Forschungsberichte aus dem Institut für Thermische Strömungsmaschinen
Prof. Dr.-Ing. H.-J. Bauer, Ord.

Erik Braun

Ein Beitrag zur Formoptimierung von Labyrinthdichtungen

Band 62/2016

Ein Beitrag zur Formoptimierung von Labyrinthdichtungen

Zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

von der Fakultät für Maschinenbau des
Karlsruher Instituts für Technologie

genehmigte

Dissertation

von

Dipl.-Ing. Erik Braun
aus Bonn

Tag der mündlichen Prüfung: 12. Mai 2016
Hauptreferent: Prof. Dr.-Ing. H.-J. Bauer, Ord.
Korreferent: Prof. Dr.-Ing. D. Peitsch

Ein Beitrag zur Formoptimierung von Labyrinthdichtungen

von

Dipl.-Ing. Erik Braun

Königswinter 2016

Vorwort des Herausgebers

Der schnelle technische Fortschritt im Turbomaschinenbau, der durch extreme technische Forderungen und starken internationalen Wettbewerb geprägt ist, verlangt einen effizienten Austausch und die Diskussion von Fachwissen und Erfahrung zwischen Universitäten und industriellen Partnern. Mit der vorliegenden Reihe haben wir versucht, ein Forum zu schaffen, das neben unseren Publikationen in Fachzeitschriften die aktuellen Forschungsergebnisse des Instituts für Thermische Strömungsmaschinen der Universität Karlsruhe (TH) einem möglichst großen Kreis von Fachkollegen aus der Wissenschaft und vor allem auch der Praxis zugänglich macht und den Wissenstransfer intensiviert und beschleunigt.

Flugtriebwerke, stationäre Gasturbinen, Turbolader und Verdichter sind im Verbund mit den zugehörigen Anlagen faszinierende Anwendungsbereiche. Es ist nur natürlich, daß die methodischen Lösungsansätze, die neuen Meßtechniken, die Laboranlagen auch zur Lösung von Problemstellungen in anderen Gebieten - hier denke ich an Otto- und Dieselmotoren, elektrische Antriebe und zahlreiche weitere Anwendungen - genutzt werden. Die effiziente, umweltfreundliche und zuverlässige Umsetzung von Energie führt zu Fragen der ein- und mehrphasigen Strömung, der Verbrennung und der Schadstoffbildung, des Wärmeübergangs sowie des Verhaltens metallischer und keramischer Materialien und Verbundwerkstoffe. Sie stehen im Mittelpunkt ausgedehnter theoretischer und experimenteller Arbeiten, die im Rahmen nationaler und internationaler Forschungsprogramme in Kooperation mit Partnern aus Industrie, Universitäten und anderen Forschungseinrichtungen durchgeführt werden.

Es sollte nicht unerwähnt bleiben, daß alle Arbeiten durch enge Kooperation innerhalb des Instituts geprägt sind. Nicht ohne Grund ist der Beitrag der Werkstätten, der Technik-, der Rechner- und Verwaltungsabteilungen besonders hervorzuheben. Diplomanden und Hilfsassistenten tragen mit ihren Ideen Wesentliches bei, und natürlich ist es der stets freundschaftlich fordernde wissenschaftliche Austausch zwischen den Forschergruppen des Instituts, der zur gleichbleibend hohen Qualität der Arbeiten entscheidend beiträgt. Dabei sind wir für die Unterstützung unserer Förderer außerordentlich dankbar.

Im vorliegenden Band der Schriftenreihe befasst sich der Autor mit der systematischen Optimierung der Gestalt von Labyrinthdichtungen mit und ohne Anstreifbeläge zur Minimierung des Durchflussverhaltens auf der Basis von Data-Mining Methoden. Aufgrund hoher Umfangsgeschwindigkeiten und teils sehr hoher Temperaturen der Komponenten bzw. des Arbeitsmediums werden in Turbomaschinen berührungsfreie Dichtungen eingesetzt. Labyrinthdichtungen sind dabei die am häufigsten eingesetzte Dichtungsart. Gerade bei Flugtriebwerken ist die minimal realisierbare Spalthöhe durch die Gefahr des Anstreifens der Dichtspitzen an die Gehäusewand begrenzt. Um dennoch sehr geringe Spaltweiten realisieren zu können, werden Anstreifbeläge eingesetzt, die ein Anstreifen bzw. Eindringen der Labyrinthspitzen erlauben, ohne dass diese dabei beschädigt werden. Nach dem Anstreifen sind die Beläge dauerhaft verformt bzw. durch Abrasion im Anstreifbereich abgetragen, so dass sich im Betrieb nach einer gewissen Einsatzzeit eine endgültige minimale Spaltweite selbständig einstellt. Mit Hilfe der vorgestellten Methodik ist es grundsätzlich möglich, die Vielzahl realer Einflussfaktoren auf das Durchsatzverhalten

von Labyrinthdichtungen in einen gemeinsamen Ansatz zusammenzufügen und einer allgemein anwendbaren multikriteriellen Optimierungsmethodik zugänglich zu machen. Dabei werden sowohl neue Dichtungsgeometrien als auch der im Betrieb zu erwartende Verschleiß gleichermaßen in die Optimierungsaufgabe einbezogen. Damit wird es dem Konstrukteur ermöglicht, den für einen konkreten Anwendungsfall relevanten Lebenszyklus bei der Auslegung einer Labyrinthdichtung zu berücksichtigen. Die Optimierungsstrategie stützt sich dabei im Wesentlichen auf datenbasierte Zielfunktionen wie Künstliche Neuronale Netzwerke oder Gaußsche Prozesse.

Karlsruhe im Juli 2016 Hans-Jörg Bauer

Vorwort des Autors

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit am Institut für Thermische Strömungsmaschinen des Karlsruher Instituts für Technologie, ehemals Universität Karlsruhe (TH), mit finanzieller Unterstützung durch das Forschungsprogramm KW21 “Kraftwerke des 21. Jahrhunderts”.

Mein ganz besonderer Dank gilt natürlich Herrn Prof. Dr.-Ing. Hans-Jörg Bauer, dem Leiter des Instituts, für die Unterstützung und Förderung meiner Arbeit, sowie für die Übernahme des Hauptreferats. Bei Herrn Prof. Dr.-Ing. Dieter Peitsch möchte ich mich für sein Interesse an meiner Arbeit und für die Übernahme des Korreferats herzlich bedanken.

Für die fachliche Betreuung meiner Arbeit möchte ich Dr.-Ing. Klaus Dullenkopf und Dr.-Ing. Corina Höfler danken. Ihre Unterstützung hat maßgeblich zum vorliegenden Ergebnis beigetragen.

Ohne kritische Diskussionen zum Thema und teils weit darüber hinaus wäre diese Arbeit nicht in ihrer jetzigen Form entstanden. Dafür möchte ich mich bei allen Kollegen und Mitarbeitern am Institut bedanken, die sich stets bereitwillig an Kaffeerunden und Flurgesprächen beteiligt haben. Besondere Erwähnung verdient hier meine langjährige Bürokollegin Natalia García Víllora. Für seine Vorarbeit und Unterstützung im Bereich des maschinellen Lernens danke ich Tim Pychynski.

Meinen zahlreichen studentischen Mitarbeitern, Studien- und Diplom- sowie in jüngerer Zeit Bachelor- und Masterarbeitern möchte ich für die Übernahme vieler kleinerer “Baustellen” danken, deren Bewältigung mich wesentlich weitergebracht hat. Nicht unerwähnt bleiben dürfen auch die vielfältigen Unterstützungsleistungen durch die mechanischen, elektrischen und IT-Werkstätten des Instituts sowie die Sekretariate und Administration. Ohne diese wäre eine sinnvolle Forschungsarbeit unmöglich.

Meinen Eltern danke ich besonders dafür, dass sie mir das Studium ermöglicht haben und für die zuverlässige Unterstützung auf diesem Weg.

Da die vorliegende Arbeit fast ausschließlich mit der Hilfe freier, quelloffener Software entstanden ist, erscheint es angebracht, hier auch einen großen Dank an die unzähligen freiwilligen Mitarbeiter der “Community” auszusprechen.

Königswinter im Juli 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Symbol	Einheit	Beschreibung
Lateinische Symbole		
a	m	Kammerhöhe
c	m/s	Geschwindigkeit
C_D	–	Durchflussbeiwert
CR	–	Crossover-Konstante
F	–	Verstärkungsfaktor
H	m	Höhe
k	J/kg	Turbulente kinetische Energie
L	m	Länge
M	–	Zielfunktionsanzahl
$\dot{m}$	kg/s	Massenstrom
N	–	Populationsgröße
n	–	Spitzenzahl
N_H	m	Nuthöhe
N_S	m	Nutversatz
N_W	m	Nutweite
p	Pa	Druck
R	m	Radius
R_{Gas}	J/kgK	spezifische Gaskonstante
s	m	Spaltweite
S_B	m	Spitzenbreite
S_H	m	Spitzenhöhe
ST_H	m	Stufenhöhe
ST_S	m	Stufenversatz
t	m	Teilung
T	K	Temperatur
Tu	–	Turbulenzgrad
U	m/s	Umfangsgeschwindigkeit
Griechische Symbole		
γ	$^\circ$	Spitzenneigungsswinkel
θ	$^\circ$	Spitzenöffnungswinkel
κ	–	Verhältnis der Wärmekapazitäten

π	–	Druckverhältnis, Kreiszahl
ρ	kg/m^3	Dichte
σ	–	Standardabweichung

Indizes

0	am Einlass
alt	im verschlissenen Zustand
i	innen
id	ideal
neu	im Neuzustand
nom	nominal
s	statische Größe
t	totale Größe
z	am Auslass

1 Einleitung

Der innerdeutsche Luftverkehr ist gemessen an Passagierkilometern nach dem Report Energieeffizienz und Klimaschutz 2013 des Bundesverbands der Deutschen Luftverkehrswirtschaft (BDL (2013)) seit 1990 um 216% gestiegen. Damit einher ging eine Steigerung des Kerosinverbrauchs um immerhin 77%. Eine weitere Steigerung des Verkehrsaufkommens ist über Deutschland hinaus auch global absehbar. Der damit wachsende Verbrauch fossiler Brennstoffe führt zu einem erhöhten Ausstoß an Treibhausgasen wie Kohlendioxid, Wasserdampf und Stickoxiden sowie Ruß und Schwefeldioxid mit entsprechenden negativen Umwelteinflüssen. Eine Senkung des Treibstoffverbrauchs von Flugtriebwerken ist daher dringend gefordert.

Zusätzlich verursachen die Treibstoffkosten immerhin gut ein Drittel der Betriebskosten einer Fluggesellschaft. Dazu können in Zukunft weitere Kosten für Emissionszertifikate hinzukommen, die derzeit auch für den Flugverkehr im Gespräch sind. Neben Umwelt- und Klimaschutzgedanken spielt also insbesondere der Kostenfaktor eine erhebliche Rolle in den Bestrebungen nach einer Senkung des Treibstoffverbrauchs von Flugtriebwerken.

Auch stationäre Gasturbinen sind von diesen Bestrebungen betroffen. Hier ist zwar aufgrund der fehlenden Randbedingung “Lufttüchtigkeit” und den damit verbundenen Gewichtseinschränkungen zum Beispiel eine Abgasnachbehandlung oder gar eine Abscheidung von Treibhausgasen grundsätzlich möglich; die Brennstoffkosten sind jedoch immer noch maßgeblich für den Betrieb.

Eine Möglichkeit zur Erhöhung des Gesamtwirkungsgrades ist die Steigerung des thermischen Wirkungsgrades. Dieser steht in direktem Zusammenhang mit dem Druckverhältnis (Overall Pressure Ratio - OPR) und der Turbineneintrittstemperatur. Für eine feste Turbineneintrittstemperatur existiert ein idealer OPR-Wert, sodass eine Steigerung des thermischen Wirkungsgrades eine Steigerung beider Größen erfordert. Ein höheres Druckverhältnis stellt jedoch höhere Forderungen an die Dichtungstechnologie.

Die Steigerung der Turbineneintrittstemperatur erfordert eine verbesserte Kühlung der thermisch hochbelasteten Turbinenkomponenten. Die durch das Schaufelmaterial vorgegebenen maximalen Materialtemperaturen liegen derzeit mehrere hundert Kelvin unter der Heißgastemparatur. Nur durch massiven Kühllufteinsatz kann ein Abschmelzen der Bauteile verhindert werden. Der Anteil der im Verdichter abgezweigten Kühlluft kann bereits über 20% des Gesamtmassenstroms bei einem Druckverhältnis von 24 ausmachen (Coppinger et al. (2002)). Dieser nimmt nicht an der Verbrennung teil und kann somit in der Turbine nur einen Bruchteil der Arbeit, verglichen mit dem Heißgas, verrichten. Dieser Sachverhalt führt in Kombination mit den auftretenden Verlusten im Sekundärluftsystem zu einer Senkung des thermischen Wirkungsgrades und der Nutzleistung im Vergleich zum theoretisch Möglichen.

Effizientere Kühlungsmethoden oder ein höherer Kühlluftmassenstrom können die steigenden Anforderungen an die Kühlung bei steigender Turbineneintrittstemperatur erfüllen. Durch ein erhöhtes Druckverhältnis steigt die Temperatur der am Verdichteraustritt abgezweigten Kühlluft, die somit ein geringeres “Kühlpotential” aufweist. Dieser Sachverhalt bewirkt eine weitere Erhöhung des Kühlluftbedarfs, um die Wärme abzuführen und die Materialtemperaturen in

einem erträglichen Bereich zu halten. Daher ist es von größter Bedeutung, dass die Kühlluft möglichst verlustfrei durch das Sekundärluftsystem an die zu kühlenden Stellen geleitet wird. Die Dichtungen und insbesondere Labyrinthdichtungen spielen hier die entscheidende Rolle.

Dichtungen haben die Aufgabe, die Leckage eines Mediums zwischen zwei Räumen über ein Druckgefälle zu vermeiden. Zur Abdichtung zwischen rotierenden und stehenden Bauteilen kommen in Turbomaschinen meist berührungsfreie Dichtungen zum Einsatz. Aufgrund des verbleibenden Spaltes sind diese jedoch nie frei von Leckage. Die Hauptaufgabe beim Entwurf solcher Dichtungen ist die Minimierung dieses Leckageverlustes.

Die Verbesserung von Labyrinthdichtungen und das Streben nach dem Verständnis der strömungs- und thermodynamischen Vorgänge in diesen sind seit Jahrzehnten Gegenstand der Forschung am Institut für Thermische Strömungsmaschinen (beispielsweise Dörr (1985), Jacobsen (1987), Schelling (1988), Waschka (1991), Scherer (1994), Willenborg (2007), Denecke (2007), Schramm (2010), Weinberger (2014)). Die vorliegende Arbeit reiht sich in die vergangenen Untersuchungen ein und ist ein wichtiger Schritt bei der Weiterentwicklung der rechnergestützten Formoptimierung von Labyrinthdichtungen.

In den vorangegangenen Arbeiten wurden Korrelationen entwickelt, die die Vorhersage des Leckageverlustes, des Wärmeübergangs oder rotordynamischer Einflüsse von Labyrinthdichtungen zum Ziel hatten. Die untersuchten Dichtungen waren zumeist Geometrien im fabrikneuen Zustand ohne Verschleißerscheinungen. Der Einfluss von abgerundeten Labyrinthspitzen wurde vereinzelt zwar untersucht, jedoch ist in der Literatur noch keine Optimierung einer Labyrinthdichtung gleichzeitig für den Neu- wie verschlissenen Zustand zu finden. Diese Lücke soll mit der vorliegenden Arbeit geschlossen werden. Es wird ein rechnergestütztes Formoptimierungswerkzeug entwickelt, das bei der Suche nach einer idealen Labyrinthgeometrie für einen bestimmten Bauraum und vordefinierte Randbedingungen den Verschleiß der Dichtung beim Betrieb mit berücksichtigt und durch Kombination verschiedener Vorhersagemethoden die Genauigkeit der Vorhersage verbessert.

2 Grundlagen und Stand der Wissenschaft

In diesem Kapitel soll ein Überblick über die für diese Arbeit notwendigen Grundlagen gegeben werden. Dazu werden zunächst die untersuchten Labyrinthdichtungen, ihr Durchflussverhalten, ihr Einfluss auf die Rotordynamik der Turbomaschine sowie mögliche Verschleißarten beschrieben. Des Weiteren werden moderne Optimierungsmethoden vorgestellt, aus denen später geeignete Algorithmen ausgewählt werden. Im letzten Abschnitt des Kapitels wird eine Übersicht über verschiedene Methoden zur Modellierung des Durchflussverhaltens von Labyrinthdichtungen, die später zum Einsatz kommen, gegeben.

2.1 Labyrinthdichtungen

Die Betriebsbedingungen in Turbomaschinen stellen hohe Anforderungen an die Dichtungstechnologie. Hohe Relativgeschwindigkeiten zwischen Rotor und Stator beziehungsweise zwischen Rotoren mit unterschiedlicher Drehrichtung, hohe Temperaturen sowie der Versatz der Dichtungen durch radiale und axiale Dehnungen verbieten den Einsatz berührender Wellendichtungen. In stationären Gas- und Dampfturbinen kommen inzwischen vermehrt Bürstendichtungen zum Einsatz. Da die Robustheit dieser Dichtungen insbesondere bei transienten Betriebspunkten in Bezug auf Verschleiß und Wärmeeintrag in den Rotor beim Anstreifen jedoch noch nicht ausreichend nachgewiesen ist, sind Labyrinthdichtungen in Flugzeugtriebwerken weiterhin die bevorzugte Dichtungsvariante. Des Weiteren kommen Labyrinthdichtungen als Sicherheitsdichtung hinter Bürstendichtungen zum Einsatz, um deren Aufgabe im Versagensfall zu übernehmen. Eine weitere kritische Stelle, an der Labyrinthdichtungen benötigt werden, ist die Abdichtung von Lagerkammern im Triebwerk. Möglicher Abrieb oder ganze Borsten von metallischen Bürstendichtungen würden in den eingesetzten Lagern und im Ölsystem zu erhöhtem Verschleiß, verstopften Leitungen und somit irreversiblen Schäden führen. Die Gefahr, die von durch Abrieb blockierten Kühlluftführungen ausgeht, ist auch im Sekundärluftsystem zu beachten. Für einen umfassenden Überblick über die Grundlagen der Labyrinthdichtungen wird auf die Arbeit von Denecke (2007) verwiesen.

Dichtungen im Sekundärluftsystem haben nach Campbell (1978) folgende Aufgaben:

- Minimierung ungewünschter Leckage zwischen rotierenden und stehenden Bauteilen
- Verhinderung von Heißgaseintritt in die Passagen zwischen Wellen, Scheiben und Radseitenräumen
- Verhinderung von Leckage des Turbinenkühlsystems
- Lagerschubausgleich
- Schnittstelle zum Lagerdichtsystem

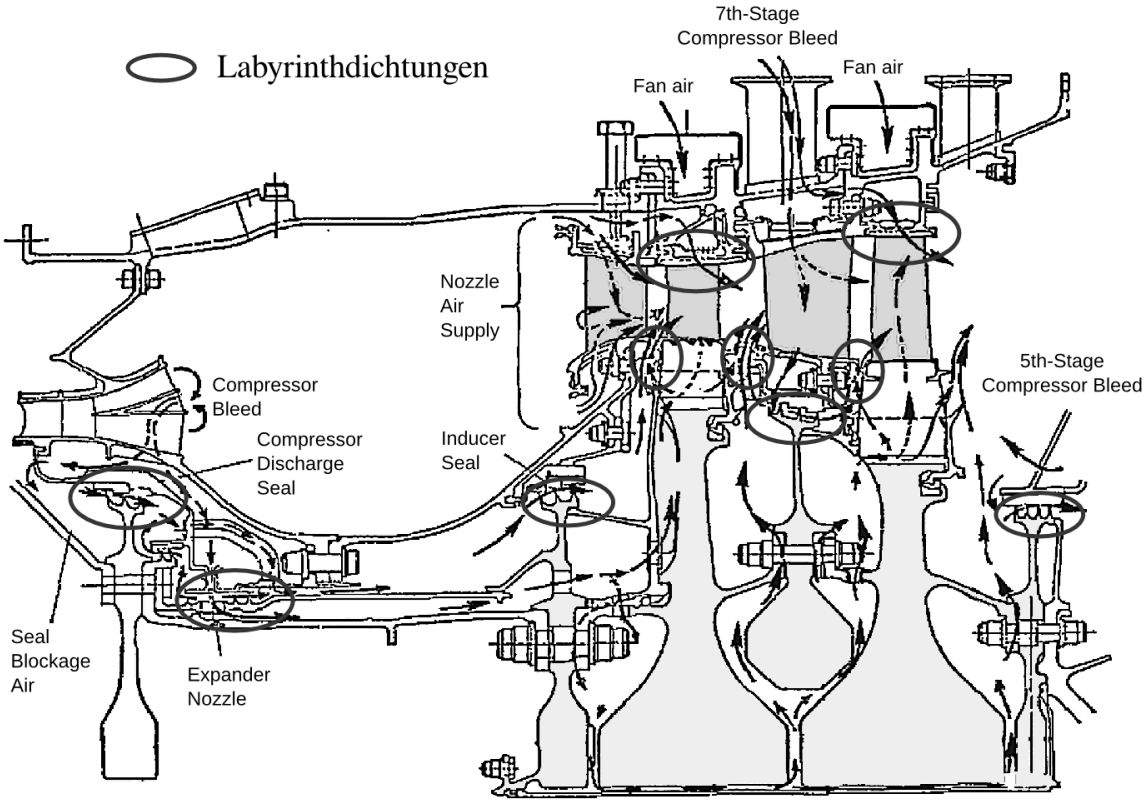

Abb. 2.1: Einbauorte von Labyrinthdichtungen im Triebwerk am Beispiel der Hochdruckturbine des E3-Triebwerks (Halila et al. (1982))

Außerdem kommen Labyrinthdichtungen an den Spitzen von Turbinenleit- und -laufschaufeln sowie an Verdichterleitschaufeln zum Einsatz, um die Ausgleichsströmung zwischen den Stufen zu verringern. Sie spielen weiter eine entscheidende Rolle bei der Einstellung von Kühlluftmassenströmen - beispielsweise zur Kühlung von Rotorscheiben.

Bei der Bauart von Labyrinthdichtungen kann zwischen zwei Hauptgruppen unterschieden werden: solchen, die sich zum Einbau in ungeteilte Gehäuse eignen und solchen, bei denen dies nicht der Fall ist. In die erste Kategorie fallen Durchblick- und Stufenlabyrinthdichtungen (siehe Abbildung 2.2). In Flugzeugtriebwerken kommen fast aussschließlich diese zum Einsatz, da die Gehäuse hier aus Gründen der Gewichtsersparnis häufig einteilig ausgeführt sind. „Echte“ und Kammnutlabyrinthe (siehe Abbildung 2.3) sind nicht axial montierbar. Die bessere Dichtwirkung dieser Geometrien wird in stationären Gas- und Dampfturbinen, deren Gehäuse teilbar ist, ausgenutzt.

Der mögliche Bewegungsspielraum in axialer Richtung ist ein weiteres wichtiges Merkmal bei der Auswahl einer geeigneten Dichtungsbauart. Der Leichtbau in Flugzeugtriebwerken und die Lageranordnung sorgen dafür, dass die Turbinenwellen sich bei Erhöhung des Schubs radial und axial dehnen (Ludwig und Bill (1980)) und sich somit die auf den Turbinenscheiben angebrachten Dichtspitzen relativ zum Stator verschieben. Axiale Durchblicklabyrinthdichtungen bieten die größte axiale Verschiebbarkeit, gefolgt von Stufenlabyrinthen. Echte und Kammnutlabyrinthe

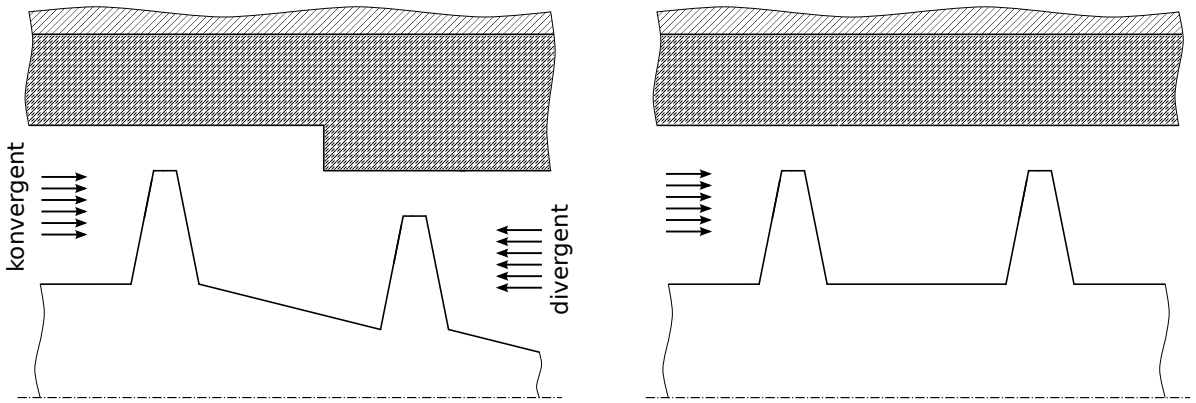

Abb. 2.2: Stufen- (links) und Durchblicklabyrinth (rechts)

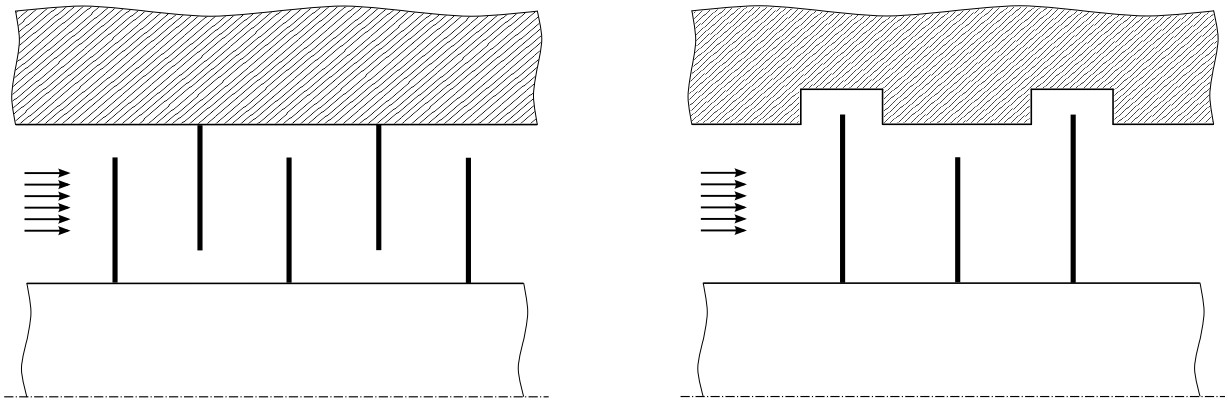

Abb. 2.3: Echtes (links) und Kammnutlabyrinth (rechts)

müssen für eine große axiale Verschiebbarkeit mit entsprechend großer Teilung gefertigt werden. Dichtungen in radialer Ausführung kommen nur selten zum Einsatz, da sich bei Axialverschiebung die Spaltweite in erheblichem Maße vergrößert, was die Dichtwirkung entsprechend mindert.

Zur Regulierung des Radialspalts kommt an den Schaufelspitzen im Turbinenbereich häufig eine aktive Spaltregelung, auch Active Clearance Control (ACC) genannt, zum Einsatz, bei der das Gehäuse durch Zapfluft erwärmt oder durch Außenluft gekühlt wird, um die Gehäusedehnung zu beeinflussen (Lattime und Steinetz (2002)). Es handelt sich dabei aber nicht um eine echte Regelung, da die Spaltweite nicht kontinuierlich gemessen wird. Vielmehr wird die Gehäusedehnung aufgrund bekannter Daten über das Dehnungsverhalten von Rotor und Stator dahingehend eingestellt, dass sich ein möglichst kleiner Spalt ohne Anstreifen ergibt. Trotz aufwendiger Methoden wie dieser und Arbeiten an einer echten Regelung mit Spaltweitenmessung (Binghui und Xiaodong (2011)) kann bei der geforderten möglichst geringen Spaltweite ein Anstreifen der Spitzen am Stator nicht gänzlich verhindert werden. Insbesondere bei Flugmanövern mit hohen Beschleunigungskräften oder dem Landestoß ist ein Anstreifen kaum zu vermeiden.

Um Beschädigungen und übermäßigem Verschleiß der Dichtspitzen beim Anstreifvorgang vorzubeugen, werden am Stator Anstreifbeläge angebracht. Zur Auswahl stehen dabei poröse, abrasive Materialien wie gesinterte oder aufgesprühte Metalle, die bei Berührung abgerieben werden,

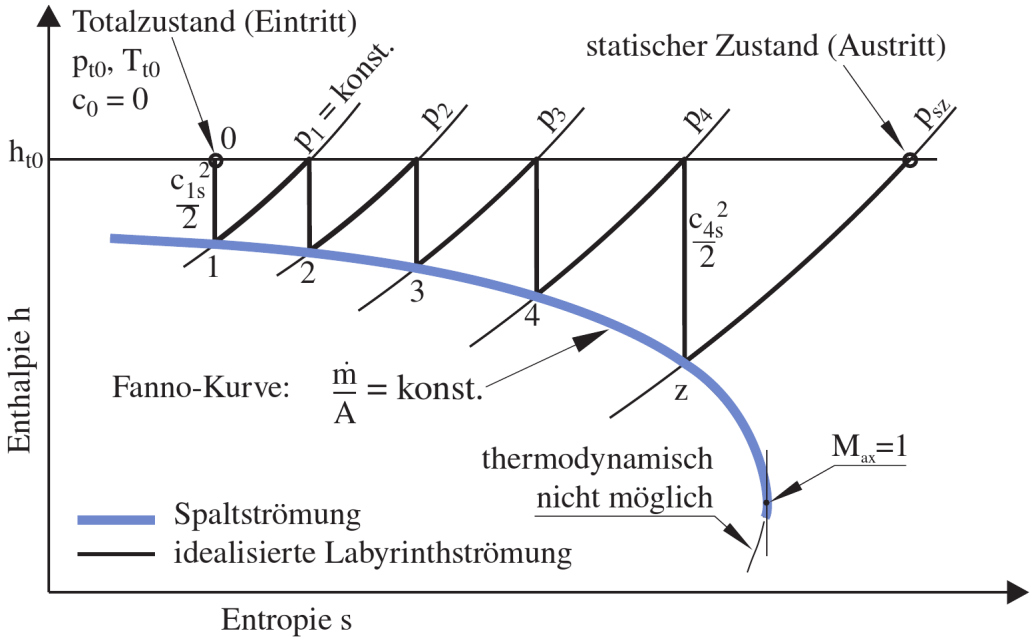

Abb. 2.4: Ideale Zustandsänderung der Labyrinthströmung (Fannokurve) (Denecke (2007))

Metallfilze, die an der Kontaktfläche gestaucht werden, Materialien mit geringer Scherfestigkeit wie aufgesprühtes Aluminium oder Graphit sowie Honigwabenstrukturen, in die die Dichtspitzen einlaufen können (Ludwig und Bill (1980)). Allen Methoden gemeinsam ist, dass beim Einlaufen eine Geometrieänderung des Stators hervorgerufen wird, die die Strömung und somit die Dichtwirkung der Labyrinthdichtung beeinflusst.

2.1.1 Durchflussverhalten

Die ideale Durchströmung von Labyrinthdichtungen kann durch wiederholte isentrope Beschleunigung und Einschnürung im Spalt, bei der statischer Druck abgebaut wird, und isobarer Verwirbelung in den Labyrinthkammern unter Dissipation kinetischer Energie beschrieben werden. In Abbildung 2.4 ist dieser Vorgang in einem h-s-Diagramm dargestellt. Bei der idealen Labyrinthströmung gilt die Energie- und Massenerhaltung einer reibungsbehafteten Strömung. Es ergibt sich daraus eine konstante Massenstromdichte ($\frac{\dot{m}}{A} = konst.$), die sich in der sogenannten Fannokurve wiederspiegelt (Friedrich (1933)).

In Abbildung 2.5 ist die Verteilung des statischen Drucks in einem Durchblicklabyrinth bei einem Druckverhältnis von $\pi \approx 1{,}5$ dargestellt. Die Strömungsrichtung ist, wie auch in allen folgenden Abbildungen, von links nach rechts. Der Leckagemassenstrom wird dabei maßgeblich durch den Spaltquerschnitt, bei gegebenem Radius durch die Spaltweite, festgelegt.

Die Strömungseinschnürung („vena contracta“), die durch eine Ablöseblase hinter der stromauf liegenden Kante hervorgerufen wird, bewirkt eine Verkleinerung des effektiv durchströmten Querschnitts und somit eine Verringerung des Leckagemassenstroms. Die Spaltfläche über der Labyrinthspitze wird im Fall von schmalen Spitzen ($S_B < s$) nicht über die gesamte Höhe durchströmt (Keller (1934)).

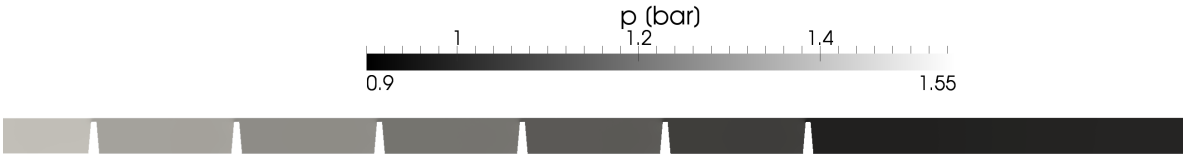

Abb. 2.5: Verteilung des statischen Drucks in einer Durchblicklabyrinthdichtung (CFD-Simulation)

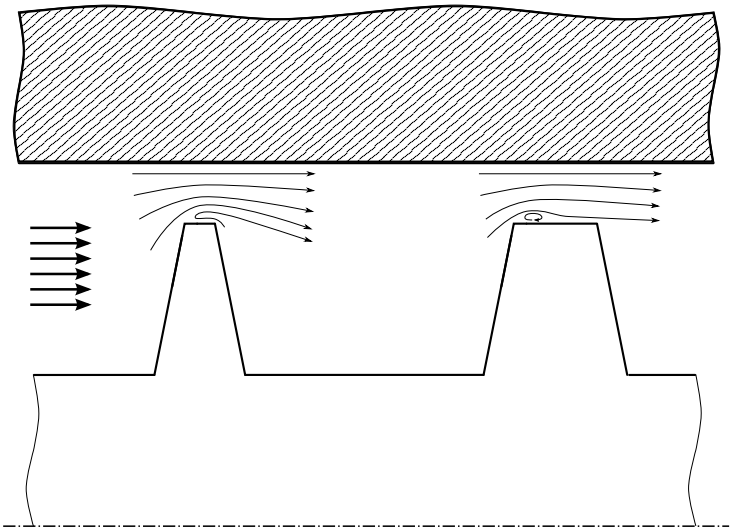

Abb. 2.6: Vena contracta über schmaler Spitze (links). Wenig Einschnürung über breiter Spitze (rechts).

Die relative Größe des Ablösegebiets bezogen auf die Spaltweite ist abhängig von der Spaltweite selbst, der Anströmrichtung und dem Kantenradius. Im Idealfall bleibt die Strömung über die gesamte Spaltlänge abgelöst, wie an der linken Spitze von Abbildung 2.6 dargestellt ist. Bei großen Verhältnissen S_B/s wird die Ablösung im vorderen Bereich des Spaltes gestaucht und die Strömung kann sich stromab wieder anlegen. Dies wird an der rechten Spitze von Abbildung 2.6 gezeigt. Die Ablösung fällt am stärksten aus, wenn die Hauptströmung den Eintritt in den Spalt von unten erreicht (vgl. konvergentes Stufenlabyrinth in Abb. 2.7). Durch die plötzliche Umlenkung in den Spalt ist die Einschnürung hier am höchsten. Beim Durchblicklabyrinth liegen alle Spalte auf demselben Radius. Dieser Umstand führt dazu, dass der im ersten Spalt sich ausbildende Strahl mit geringer Auffächerung über den Kammerwirbel hinwegströmt und parallel zur Statorfläche auf den nächsten Spalt trifft. Dadurch wird die Ablösung verkleinert. Abgerundete Kanten an der Spitze haben den stärksten Einfluss auf die Einschnürung. Schon geringe Radien können eine Strömungsablösung über der Spitze vollständig verhindern, sodass der gesamte Spaltquerschnitt durchströmt wird (Zimmermann et al. (1994), Rhode und Allen (1999)).

In den Labyrinthkammern zwischen den Spitzen wird die Strömung verwirbelt und die kinetische Energie in Wärme dissipiert. Idealerweise beginnt die Durchströmung des folgendenen Spaltes wieder mit Geschwindigkeit Null. Im realen Fall hat die Strömung aber bereits eine Vorge-

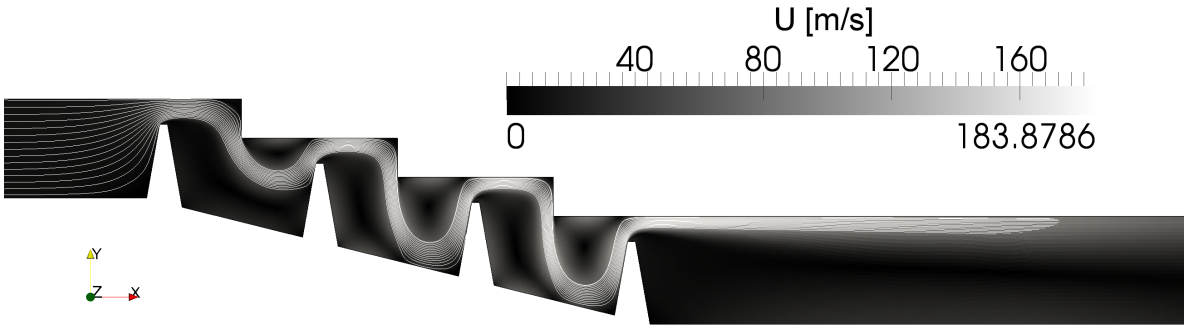

Abb. 2.7: Hauptströmung durch ein konvergentes Stufenlabyrinth

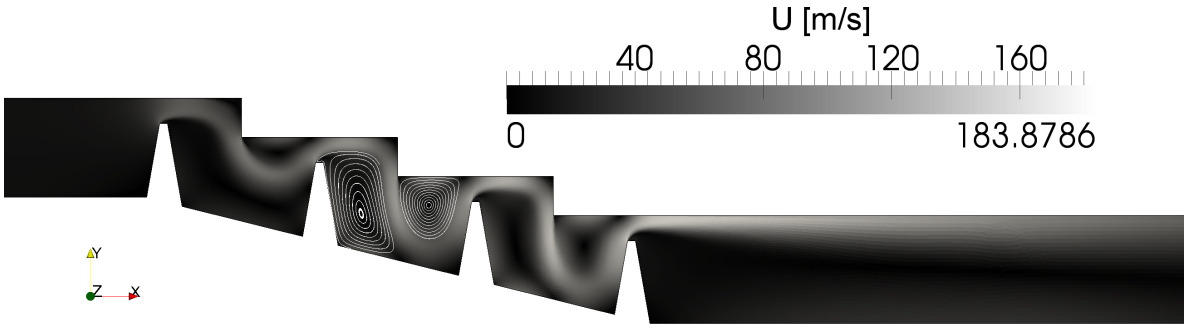

Abb. 2.8: Kammerwirbel in einem konvergenten Stufenlabyrinth

schwindigkeit. Insbesondere beim Durchblicklabyrinth ist dieser sogenannte Carry-Over-Effekt stark ausgeprägt und verringert die Dichtwirkung, da ein großer Teil des Fluids nicht in den Kammerwirbel gelangt (Komotori (1957), Zimmermann und Wolff (1998)).

Je nach Bauart der Dichtung können sich ein oder mehrere primäre Kammerwirbel ausbilden. Diese können gleich- oder gegensinnig drehen. In der Nähe des Spitzenfußes können sich zusätzlich Sekundärwirbel ausbilden. In Abb. 2.8 sind beispielhaft die beiden gegensinnig drehenden Kammerwirbel der mittleren Kammer eines Stufenlabyrinths mit vier Spitzen dargestellt.

Durch die Rotation der meist auf dem Rotor angebrachten Dichtspitzen wird auch die Fluidschicht in dessen Nähe in Rotation versetzt. Diese bewirkt zunächst eine Umlenkung der Labyrinthströmung in Umfangsrichtung, sodass die Überströmung der Spitzen nicht mehr senkrecht zu den diesen erfolgt und sie der Strömung somit breiter erscheinen. Zusätzlich wird durch die Fliehkräfte ein Pumpeffekt hervorgerufen, der bewirkt, dass die Strömung nach außen strebt. Es bildet sich ein Druckgefälle von innen nach außen, das sich dem Druckgefälle über die Dichtung überlagert. Dieser Effekt vergrößert die Ablösung über der Labyrinthspitze mit entsprechender Verbesserung der Dichtwirkung. Bei gestuften Labyrinthen in konvergenter Bauart wird die Dichtwirkung zusätzlich dadurch verbessert, dass die Strömung nun gegen den Pumpeffekt strömen muss (das effektive Druckverhältnis π wird kleiner). Der Einfluss der Rotation wurde am ITS von Waschka (1991) untersucht. Dieser wird erst ab hohen Umfangsgeschwindigkeiten relevant für die Leckage. Ab einem Verhältnis von Umfangs- zu Axialgeschwindigkeit $c_u/c_{ax} \approx 1$ ist der Einfluss messbar. Auch die Arbeiten am ITS von Denecke (Denecke et al. (2004a), Denecke et al. (2004b), Denecke

et al. (2005a), Denecke et al. (2005b) und Denecke (2007)) bestätigen diesen Sachverhalt.

Zur Beschreibung der Dichtwirkung einer Labyrinthdichtung wird sehr häufig der Durchflussbeiwert C_D verwendet. Er ist definiert durch

$$C_D = \frac{\dot{m}}{\dot{m}_{ideal}} \tag{2.1}$$

wobei $\dot{m}$ den Massenstrom durch die Labyrinthdichtung und $\dot{m}_{ideal}$ den Massenstrom durch eine verlustfrei durchströmte Düse gleichen geometrischen Querschnitts darstellen. Der ideale Massenstrom wird berechnet durch

$$\dot{m}_{ideal} = \frac{Q_{ideal} \cdot p_{t,0} \cdot A}{\sqrt{T_{t,0}}} \tag{2.2}$$

mit

$$Q_{ideal} = \sqrt{\frac{2\kappa}{R(\kappa-1)}\left[1-\left(\frac{1}{\pi}\right)^{\frac{\kappa-1}{\kappa}}\right]}\left(\frac{1}{\pi}\right)^{\frac{1}{\kappa}} \tag{2.3}$$

wobei

$$\pi = \frac{p_{t,0}}{p_{s,z}} \tag{2.4}$$

als Druckverhältnis für unterkritische Werte und $\pi = 1{,}893$ für überkritische Werte verwendet werden.

2.1.2 Rotordynamik

Selbsterregte Rotorschwingungen in modernen Turbomaschinen können zum Teil durch die eingesetzten Dichtungen hervorgerufen werden. Benckert (1980) untersuchte den Einfluss von Labyrinthdichtungen, insbesondere von durch eine exzentrische Lage hervorgerufenen Druckquerkräften, auf die Rotordynamik. Dabei führte die Betrachtung des Drallverlaufs im Labyrinth zu der Erkenntnis, dass sich drallvermindernde Maßnahmen positiv auf die Rotorstabilität auswirken. Eingehende Untersuchungen in der Folge eines Totalschadens am Haupttriebwerk des Space Shuttles ermittelten ein instabiles dynamisches Rotorverhalten in der Kraftstoff-Turbopumpe, das durch die eingesetzten Labyrinthdichtungen hervorgerufen wurde, als Fehlerquelle. Die Änderung der Dichtungsgeometrie konnte das Problem beheben (Hendricks et al. (2004b)). Eine gute Übersicht über den Einfluss von Labyrinthdichtungen auf die Rotordynamik bietet die Arbeit von Kwanka (2001), die Auslegungsansätze und Vergleiche dieser mit experimentellen Untersuchungen liefert.

2.1.3 Verschleiß

Zimmermann et al. (1994) untersuchten die Auswirkung von verschlissenen Labyrinthspitzen auf die Leckage von Durchblick- und Stufenlabyrinthen. Sie verwendeten dazu Originalgeometrien der Dichtspitzen aus Triebwerken. Das Verschleißbild beim Anstreifen des Rotors an abrasiven

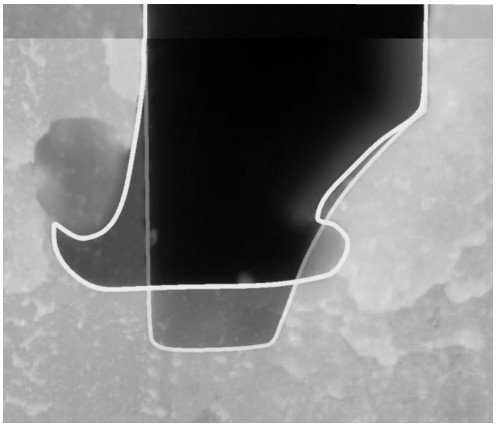

Abb. 2.9: Vergleich zwischen Originalgeometrie und verschlissener Dichtspitze (Herrmann (2013))

Statorbeschichtungen war ein Verrunden der Dichtspitzen. Jedoch konnte auch bei fabrikneuen Labyrinthdichtungen bereits ein geringer Kantenradius nachgewiesen werden, der im Vergleich zu einer scharfen Kante eine kleinere Ablösung über der Spitze und somit eine geringere Strahleinschnürung („Vena contracta“) bewirkte, was zu einem höheren Leckagemassenstrom führte. Die Experimente zeigten einen kleineren Einfluss auf den Durchfluss bei Durchblicklabyrinthen (verschlissen ca. 15% höherer Leckagemassenstrom als bei einer scharfen Kante) als bei Stufenlabyrinthen (ca. 35% größer für die untersuchten Geometrien).

Bereits Wittig et al. (1986) untersuchten die Auswirkungen von abgerundeten Spitzen an Durchblicklabyrinthen und ermittelten dabei vergleichbare Ergebnisse wie Zimmermann et al. (1994). Besondere Beachtung erhielt die erste Spitze. Eine Abrundung dieser Spitze bewirkte einen größeren Anstieg des Leckagemassenstroms als die Abrundung aller weiteren Spitzen. Diesem Effekt liegt die besonders große Einschnürung über der ersten Dichtspitze zugrunde, die durch einen Kantenradius stark verringert wird. Durch die ohnehin schwächere Einschnürung über den stromab liegenden Spitzen, die durch die fast axiale Anströmung hervorgerufen wird, ist der Effekt des Radius' hier weniger ausgeprägt.

Rhode und Allen (1999) weisen in ihrer Untersuchung von abgerundeten Spitzen darauf hin, dass bereits fabrikneue Labyrinthspitzen, insbesondere bei Einsatz von panzernden Beschichtungen, große Radien aufweisen können, die eine entsprechende Leckageerhöhung im Vergleich zu perfekt scharfkantigen Dichtspitzen nach sich ziehen.

Beim Anstreifen von ungepanzerten Labyrinthspitzen an einem glatten Stator ohne Anstreifbelag konnten Herrmann et al. (2013) ein „Aufpilzen“ der Spitzengeometrie beobachten. Beim Anstreifen wurde das Spitzenmaterial stark aufgeheizt und erweicht und wich in axialer Richtung in und gegen Strömungsrichtung aus. Dabei wurde die Spitze breiter und abgerundet, gleichzeitig verringerte sich die Spitzenhöhe S_H durch die Materialverlagerung, wie Abbildung 2.9 dargestellt wird. Sowohl die Spitzenverbreiterung, die -abrundung und die durch Verringerung der Spitzenhöhe hervorgerufene Spaltvergrößerung bewirken einen Anstieg des Leckagemassenstroms um bis zu 50%.

Nutenbildung ist das zweite hauptsächliche Verschleißbild, das in Labyrinthdichtungen auftreten

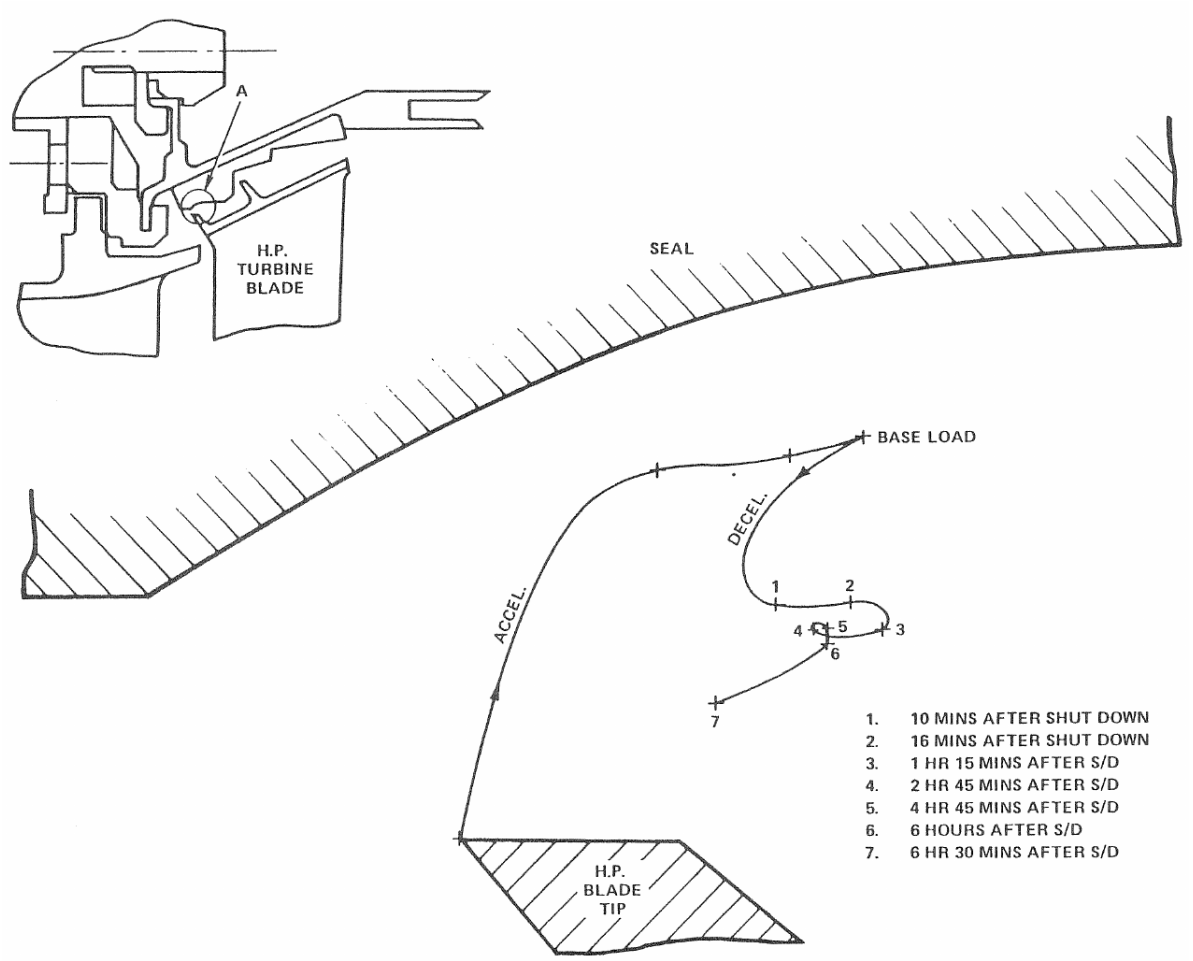

Abb. 2.10: Bewegung einer Schaufelspitzendichtung beim Anfahren verschiedener Betriebspunkte und nach dem Abschalten (Stewart und Brasnett (1978))

kann. Insbesondere bei den üblicherweise in Gasturbinen eingesetzten Anstreifbelägen wie Honigwaben oder abrasiven Materialien entstehen die Anstreifnuten durch streifenden Kontakt des Rotors mit dem Stator. Da sich gerade Turbinenscheiben nicht nur radial, sondern teils maßgeblich in axialer Richtung bewegen (vgl. z.B. Stewart und Brasnett (1978) und Chougule et al. (2006)), sind die Nuten nur in erster Näherung rechteckig. Meist weisen sie eine hügelartige Kontur mit einem oder mehreren Maxima auf (Zimmermann et al. (1994), Hendricks et al. (2004a) und Chougule et al. (2006)). Die Bewegung einer Dichtspitze einer Schaufelspitzendichtung einer Hochdruckturbine ist in Abbildung 2.10 dargestellt.

Zum Verständnis des Einflusses von Anstreifnuten auf den Strömungsverlauf untersuchten Rhode und Allen (1998) und Rhode und Allen (1999) maßstäblich stark vergrößerte Labyrinthdichtungen in einem ebenen Wasserkanal. Dabei wurde Aluminiumglitter zur Visualisierung der Strömung verwendet. Bei großen Spaltweiten war der Einfluss auf die Strömung derart, dass der Strömungswiderstand durch Anstreifnuten um 40…200% vergrößert wurde. Bei kleinen Spaltweiten sinkt der Strömungswiderstand, da die effektive Spaltweite vergrößert wird. Die Nutbreite N_W hat dabei einen großen Einfluss, da diese den Strömungswinkel des Strahls in der folgenden Labyrinthkammer verändert. Ein interessantes Ergebnis der Arbeiten ist der minimale

Unterschied im Strömungswiderstand zwischen abgerundeten und rechteckigen Anstreifnuten (Rhode und Allen (1998)). Nuten, deren Außenkontur (Winkel der Seiten und dergleichen) unterschiedlich sind, können jedoch auch sehr unterschiedliche Strömungsbilder in der Kammer und somit auch unterschiedliche Leckagemassenströme hervorrufen (Xu et al. (2004)).

Bei der Betrachtung möglichst verschleißfester Labyrinthdichtungen darf der Wärmefluss durch die Spitze in den Rotor im Falle des Anstreifens nicht außer Acht gelassen werden. Ludwig und Bill (1980) beschreiben die Möglichkeit unkontrollierter lokaler Ausdehnung des Rotors durch Anstreifen eines Winkelsegments und starker Erwärmung in diesem Bereich. Wenn die entstehende Reibungswärme nicht abgeführt werden kann, kann dies zur Schädigung der Spitzen führen. Dieser Sachverhalt ist derzeit auch am ITS Forschungsgegenstand und wird zum Beispiel von Pychynski et al. (2013) thematisiert.

2.2 Optimierung

In diesem Kapitel sollen Begrifflichkeiten geklärt und ein Überblick über verschiedene Optimierungsverfahren gegeben werden, die im Laufe der vergangenen Jahrzehnte entwickelt wurden. Dabei muss zwischen Einzelziel- und Mehrzieloptimierung unterschieden werden. Nicht jeder Algorithmus eignet sich für beide Verfahren, wohl aber können Mehrzielprobleme in Einzelzielprobleme überführt werden (Dréo (2006)). Die hier vorgestellten Sachverhalte dienen der Entscheidungsfindung für die Auswahl geeigneter Algorithmen zur Optimierung von Labyrninthdichtungsgeometrien hinsichtlich des Durchflussverhaltens in Kapitel 4.2.

2.2.1 Einführung

Im Laufe einer Optimierung, d.h. der Suche nach dem Optimum, wird eine Anzahl Optimierungsparameter variiert. Ein „Individuum“ oder auch eine Konfiguration ist eine Kombination aus Parametern mit bestimmten Werten. Dies entspricht in der vorliegenden Arbeit einer definierten Labyrinthgeometrie mit einer bestimmten Spitzenzahl, Teilung, Stufenhöhe und weiteren Parametern.

Die Optimierungsparameter, die ein Individuum beschreiben, sind gleichzeitig die Eingabevariablen $\vec{x} = (x_1, x_2, \ldots, x_N)^T$ der „Zielfunktion“. Eine Zielfunktion liefert als Ausgangsgröße die Güte, also die zu optimierende Größe, eines Individuums zurück. Bei evolutionären Algorithmen wird auch von der „Fitness“ eines Individuums gesprochen. Die Zielfunktion kann eine beliebige Methode sein, die in der Lage ist, die Güte zu beschreiben. Im Falle von Labyrinthdichtungen sind dies zum Beispiel die numerische Strömungsmechanik (CFD), Bulk-Flow-Modelle, Korrelationen, aber auch die experimentelle Bestimmung der Güte.

Eine einfache Optimierungsaufgabe ist die Minimumssuche der ersten Testfunktion von De Jong (1975)

$$f(\vec{x}) = \sum_{i=1}^{N} x_i^2 \tag{2.5}$$

Hier ist $f(\vec{x})$ die Zielfunktion und der Vektor $\vec{x} = (x_1, x_2, \ldots, x_N)^T$ mit $x_i \in \mathbb{R}$ das jeweilige Individuum. Bei Labyrinthdichtungen besteht der Vektor der Eingangsparameter zum Beispiel aus $\vec{x}_{Labyrinth} = (s, t, S_H, S_B, ST_H, ST_S, \theta, \gamma, \pi)^T$ im Falle eines Stufenlabyrinths.

Bei Einzelzielproblemen, sogenannten monokriteriellen Problemen, wird pro Individuum genau eine Zielfunktion abgefragt. Die optimale Lösung des Problems ist dann das Individuum mit der besten Güte - bei einer Minimumssuche dasjenige mit dem kleinsten Zielfunktionswert. In Mehrzielproblemen, sogenannten multikriteriellen Problemen, werden pro Individuum mehrere Zielfunktionen abgefragt. Für die Leckageoptimierung von Labyrinthdichtungen, die später betrachtet wird, werden zwei Zielfunktionen verwendet. Eine beschreibt die Leckage im Neuzustand, die zweite die Leckage mit verschlissener Geometrie. Da für dasselbe Individuum sehr gute Werte bei einer Zielfunktion, aber sehr schlechte bei einer weiteren Zielfunktion ermittelt werden können, ist es nicht trivial, die *eine* beste Lösung zu finden. Eine Möglichkeit, das Problem zu vereinfachen, besteht darin, Gewichtungsfaktoren für die einzelnen Zielfunktionen zu vergeben und somit einen neuen Gütewert basierend auf den Ergebnissen der Zielfunktionen zu berechnen. Diese Vorgehensweise vereinfacht das Mehrzieloptimierungsproblem zu einem Einzelzielproblem. Sie setzt aber Vorkenntnisse über die Größe des Einflusses jeder einzelnen Zielfunktion voraus, die im Vorfeld nicht immer gegeben sind.

Eine andere Variante der Lösung multikriterieller Probleme ist die Optimierung unter Verwendung der Pareto-Dominanz-Methode (Voorneveld (2003)). Für ein neu erzeugtes Individuum werden zunächst die Werte aller Zielfunktionen bestimmt. Es wird zur Menge der "nicht-dominierten" Individuen hinzugefügt, wenn es in mindestens einem Zielfunktionswert besser ist als alle bereits vorhandenen Individuen. Ein Individuum wird aus der Menge gelöscht, sobald ein neues Individuum in mindestens einem Zielfunktionswert besser und dabei nicht schlechter in allen anderen Zielfunktionswerten ist. Aus der Menge der pareto-optimalen Individuen kann schließlich der beste Kompromiss ausgewählt werden. In Abbildung 2.11 ist dieser Sachverhalt anhand zweier Zielfunktionen ZF1 und ZF2 dargestellt, die minimiert werden sollen. Die hellgrauen Individuen sind Mitglieder des Pareto-optimalen Sets. Eine Verringerung eines Zielfunktionswerts führt innerhalb dieser Menge automatisch zur einer Vergrößerung und somit Verschlechterung bei der anderen Zielfunktion.

Der Suchraum, in dem das globale Optimum gefunden werden soll, ist bei theoretischer Betrachtung eines Optimierungsproblems meistens unbeschränkt. In der vorliegenden Arbeit sollen jedoch reale Probleme aus dem ingenieurswissenschaftlichen Bereich bearbeitet werden, bei denen eine freie Suche in aller Regel nicht uneingeschränkt möglich ist. Dies bedeutet im Fall der Optimierung einer Labyrinthdichtung, dass die gesuchte Labyrinthdichtung nur optimal, also leckageminimal, in den Grenzen eines bestimmten Bauraums und für bestimmte Randbedingungen sein soll. Dabei kann jeder Parameter individuell durch eine obere und untere Schranke begrenzt werden, aber auch durch Kombinationen aus Parametern. Somit wird der Suchraum für die Optimierung vorgegeben.

Die Testfunktion von De Jong aus Gleichung 2.5 ist eine einfache mehrdimensionale Parabel, die im gesamten Suchraum $\mathbb{R}^n$ definiert und differenzierbar ist. Da sie nur einen Extrempunkt besitzt, findet die große Mehrzahl aller Optimierungsalgorithmen das Optimum zuverlässig (Weise (2009)). Bei realen Optimierungsaufgaben ist dies leider oft nicht der Fall. Da reale

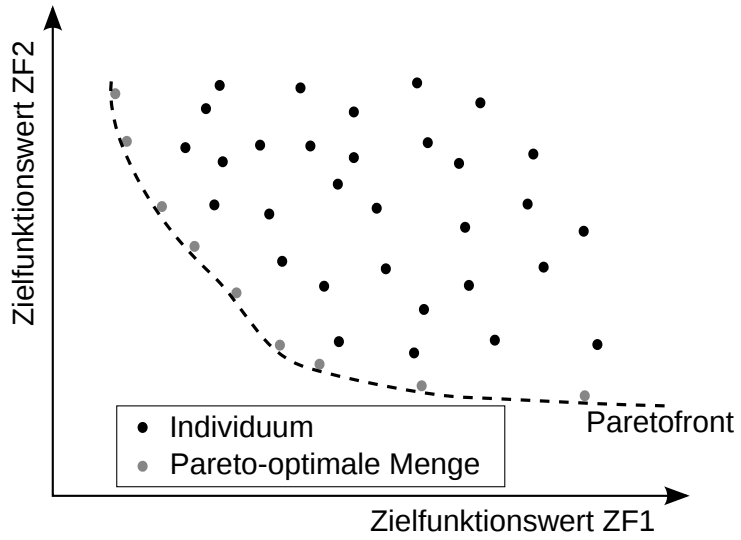

Abb. 2.11: Pareto-optimale Menge als Menge bester Kompromisse

Zielfunktionen häufig aus abschnittsweise definierten algebraischen Gleichungen (z.B. Korrelationen) bestehen und numerische Methoden nicht in jedem Fall eine Lösung liefern, eignen sich nicht alle Algorithmen gleichermaßen für solche Aufgaben. In den folgenden Abschnitten wird eine Auswahl verschiedener Methoden vorgestellt.

2.2.2 Gradientenbasierte Verfahren

Optimierungsalgorithmen werden in den Ingenieurswissenschaften und insbesondere im Maschinenbau bereits seit einigen Jahrzehnten eingesetzt. Die Zahl der Veröffentlichungen über den Einsatz von Optimierungsalgorithmen zum Entwurf von Turbomaschinenkomponenten ist in den letzten zehn Jahren stark gestiegen. In einer Vielzahl der Fälle wurden einfache Algorithmen wie gradientenbasierte Verfahren eingesetzt, die sich durch eine geringe Anzahl notwendiger Zielfunktionsaufrufe auszeichnen, was gerade bei aufwendigen Zielfunktionen wie der numerischen Strömungssimulation eine wichtige Eigenschaft ist. Gradientenbasiert bedeutet, dass die Auf- oder Abstiegsrichtung, je nachdem, ob es sich um ein Maximierungs- oder Minimierungsproblem handelt, anhand der Größe der ersten Ableitung der Zielfunktion bestimmt wird. Die Bestimmung der Gradienten kann insbesondere bei hochdimensionalen Problemen und nicht oder nur abschnittsweise differenzierbaren rechentechnisch sehr aufwendig oder unmöglich sein.

Ein Beispiel für ein gradientenbasiertes Verfahren ist der „Hill-Climbing“-Algorithmus (Winston (1987)): Der Hill-Climbing- oder auch Bergsteiger-Algorithmus versucht, auf schnellstem Wege den höchsten Wert der Zielfunktion bei einem Maximierungsproblem zu erreichen. Dabei sucht er in der Nachbarschaft des aktuellen Ortes nach dem Ort mit der größten Verbesserung, indem jeweils ein Parameter variiert wird und übernimmt diesen als neue Position, von der aus weitergesucht wird. Um ein Optimum exakt zu treffen, kann die Schrittweite bei fortschreitender Suche langsam eingeschränkt werden. Dieser Algorithmus hat mehrere Nachteile, die nach Winston (1987) folgendermaßen beschrieben werden:

- Vorgebirgsproblem: Der Algorithmus verfängt sich in lokalen Optima.
- Plateauproblem: Einzelne Extrema auf einer ebener Fläche werden nicht gefunden.
- Gratproblem: Beim Suchen in Einheitsrichtungen ist keine Verbesserung möglich.

Insbesondere die ersten beiden Probleme sind für klassische gradientenbasierte Verfahren schwierig zu bewältigen. Abhilfe kann eine Variation der Startposition schaffen, um auf diese Weise den durchsuchten Raum zu erweiterten und so eventuell beim ersten Lauf übersehene Optima zu finden. Dies erhöht allerdings die notwendige Anzahl an Zielfunktionsaufrufen unter Umständen erheblich, da mehrere Suchläufe erforderlich sind, und auch die Definition der Abbruchbedingung ist nicht einfach.

Ein Verfahren, das durch seine hohe Recheneffizienz in letzter Zeit sehr beliebt geworden ist, ist das *Verfahren der konjugierten Gradienten (Conjugate Gradients, CG)*. CG ist eine Variante des Verfahrens der konjugierten Richtungen (Conjugate Directions, CD), das einen Ersatz für das aufwendigere Newtonverfahren zur Minimierung quadratischer und nicht-quadratischer Funktionen darstellt. Der CD-Algorithmus, der in Hestenes (1980), S. 108f erläutert wird, führt n eindimensionale Minimierungen in n linear unabhängigen Richtungen durch und findet so ein Minimum in n Dimensionen auf wenig rechenintensive Weise. Der Spezialfall CG benutzt Gradienten, um die eindimensionalen Minimierungen durchzuführen und die konjugierten Vektoren für das CD-Verfahren zu bestimmen. Es ist somit auch eine Methode des steilsten Abstiegs mit den oben angesprochenen Nachteilen (Hestenes (1980)).

Bei der Optimierung von strömungsmechanischen Bauteilen mittels numerischer Methoden (CFD) besteht das Problem, dass die Lösung eines realistischen Falls (z.B. instationäre Simulation einer mehrstufigen Turbine) sehr zeitaufwendig sein kann. Einfache gradientenbasierte Verfahren verwenden Approximationen mittels Finiter-Differenzen-Verfahren, um die Gradienten zu bestimmen, die zur Auswahl der Abstiegsrichtung erforderlich sind. Dazu sind mehrere Zielfunktionsevaluationen pro Dimension des Problems notwendig, was einen entsprechenden Rechenaufwand nach sich zieht. Durch die Integration einer Sensitivitätsanalyse in die Eulergleichungen, die zur Berechnung der Strömungsgrößen hier Verwendung fanden, konnten Burgreen und Baysal (1994) die Optimierung so beschleunigen, dass die Rechenzeit, die für die Optimierung benötigt wurde, fast der einer einfachen CFD-Rechnung entsprach, die Rechenzeit inklusive Optimierung also nur verdoppelt wurde. Für die Lösung der linearen Gleichungssysteme sowohl der CFD-Simulation als auch der Sensitivitätsanalyse kam das vorkonditionierte konjugierte Gradientenverfahren (preconditioned conjugate gradients, PCG) zum Einsatz (Burgreen und Baysal (1994)). Auch am ITS wurde das angesprochene Verfahren durch Noll (1986, 1992) eingehend untersucht.

Der große Vorteil dieser Methode ist, dass die Zielfunktion während der laufenden CFD-Rechnung „mitgelöst“ wird. Die Zielfunktion und die Randbedingungen müssen dabei in geeigneter Form in die Strömungsgleichungen integriert werden. In einer weiteren Arbeit von Burgreen und Baysal (1996) wird die dreidimensionale Konturoptimierung eines Tragflügels für ein Verkehrsflugzeug im transsonischen Geschwindigkeitsbereich gezeigt. Dabei wurde die Flügelgeometrie mittels Bezierkurven diskretisiert. Die Position der einzelnen Stützpunkte für die Bezierkurven stellten

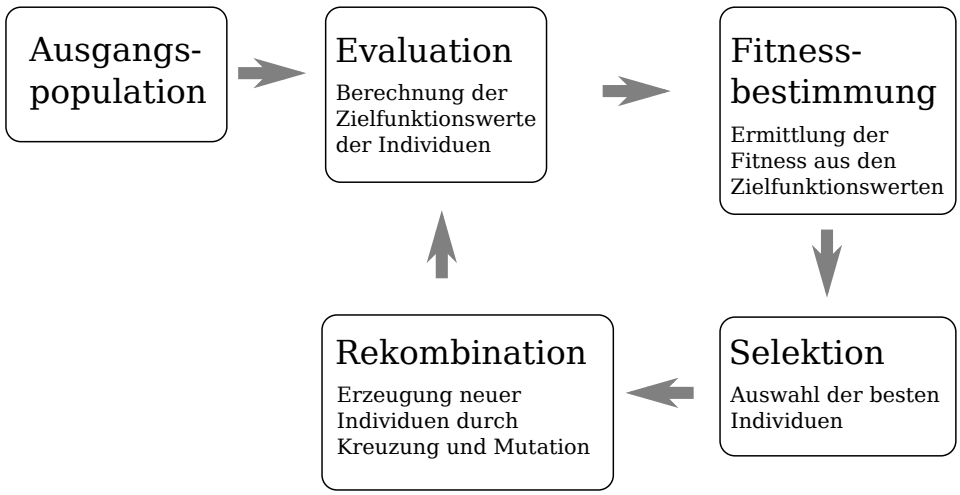

Abb. 2.12: Ablauf eines evolutionären Algorithmus nach Weise (2009)

die Optimierungsparameter dar. Als Zielgröße wurde die Gleitzahl als Verhältnis aus Auftriebs- zu Widerstandsbeiwert c_A/c_W verwendet. Die Auswahl geeigneter Formulierungen der Randbedingungen der Optimierung war offenbar nicht trivial. Mehrere Versuche mit unterschiedlichen Randbedingungen waren notwendig, um eine vorzeitige Konvergenz des Optimierers in einem lokalen Optimum zu verhindern. Die Untersuchungen zeigten, dass der Suchraum für einen transsonischen Tragflügel „topologisch komplex und voller lokaler Maxima“ (Burgreen und Baysal (1996)) ist. Kleine Änderungen bei der Formulierung des Problems führten schon zu großen Änderungen im Ergebnis. Diese Tatsachen heben die vorher genannten Vorteile des Verfahrens zumindest teilweise wieder auf und verhindern eine robuste Methode, die ohne große Interaktion mit einem Benutzer zuverlässig funktioniert.

2.2.3 Evolutionäre Algorithmen

Evolutionäre Algorithmen, auch: “genetische” Algorithmen, sind Algorithmen, die mit einem Satz an Individuen, Population genannt, arbeiten. Um neue Individuen zu erzeugen, werden aus einer Elternpopulation einzelne Individuen gewählt (*Selektion*), die Nachkommen erzeugen (*Rekombination*). Durch *Mutation* erhält die Suche einen stochastischen Anteil. Alle drei Mechanismen *Selektion*, *Rekombination* und *Mutation* wurden von der natürlichen Evolution der Lebewesen (Darwin (1859)) inspiriert. Eine Übersicht über den Ablauf ist in Abbildung 2.12 gegeben.

- Selektion: Wahl der Individuen für die Rekombination aus der Elternpopulation und Wahl der Individuen, die durch die Nachkommen ersetzt werden.
- Rekombination: Weitergabe oder Vererbung von Elternmerkmalen an die Nachkommengeneration
- Mutation: Zufällige Variation von Nachkommenmerkmalen

Unterschiedliche Methoden bei der Anwendung dieser Mechanismen haben eine große Bandbreite an evolutionären Algorithmen entstehen lassen. Zusätzlich führt eine große Anzahl einstellbarer Parameter je Algorithmus dazu, dass eine große Variation zwischen schneller Konvergenz und gründlicher Suche im gesamten Suchraum möglich ist. Durch diese Flexibilität lassen sich Algorithmen dieser Klasse an verschiedenste Optimierungsprobleme anpassen. Allgemein zeichnet sich diese Klasse von Algorithmen durch eine gründliche Suche bei entsprechend hohem Aufwand aus. Jedoch wird das globale Optimum gerade bei Problemen mit unbekannter Zielfunktion mit größerer Wahrscheinlichkeit gefunden als zum Beispiel mit gradientenbasierten Methoden. Die Gefahr der Konvergenz in ein lokales Optimum ist auch hier gegeben, wenn eine ungeeignete Wahl der Parameter des Algorithmus zu einem hohen Selektionsdruck führt, wodurch sich ein gutes, aber nicht optimales Individuum zu stark vermehren kann. Für eine umfassende Einführung in Metaheuristiken, zu denen die evolutionären Algorithmen gehören, sei auf Dréo (2006) verwiesen.

Ein bekannter Vertreter dieser Klasse der evolutionären Algorithmen ist *Differential Evolution (DE)* von Storn und Price (1995). DE ist in der Lage, auch in nichtlinearen und nichtdifferenzierbaren Suchräumen mit hoher Wahrscheinlichkeit das globale Optimum zu finden. Die Methode ist ausgesprochen robust, und die Zielfunktionsberechnung lässt sich gut parallelisieren, indem alle Individuen einer Population parallel berechnet werden. Die geringe Anzahl einstellbarer Parameter

- Populationsgröße
- Verstärkungsfaktor für die Mutation F und
- Crossover-Konstante CR

sorgt für eine einfache Konfiguration und hat zu der großen Popularität der Methode beigetragen.

Von Differential Evolution existieren verschiedene Varianten für die multikriterielle Optimierung. Eine Erweiterung ist *Generalized Differential Evolution (GDE)*, deren dritte Generation GDE3 von Kukkonen und Lampinen (2005) vorgestellt wurde. Diese Variante verwendet zum Vergleich von Individuen das Konzept der Paretodominanz. Dabei kann es vorkommen, dass die Population im Laufe einer Iteration über den eingestellten Wert anwächst. Zum Ende einer Iteration wird diese aber wieder ausgedünnt, indem nur diejenigen Individuen verbleiben, die zueinander einen möglichst gleichen Abstand besitzen, um eine gleichmäßige Verteilung der Individuen auf der Paretofront zu erreichen. GDE3 degeneriert für eine monokriterielle Optimierung wieder zum klassischen DE (Kukkonen und Lampinen (2005)).

Der *Non-Dominated Sorting Genetic Algorithm (NSGA-II)* von Deb et al. (2002) ist ein multikriterieller Optimierungsalgorithmus, der aufgrund seiner, im Vergleich zu vielen anderen Algorithmen, geringeren Komplexität und der gleichmäßigen Besetzung der Paretofront große Verbreitung gefunden hat. Die algorithmische Komplexität wird bei einem multikriteriellen Optimierer stark durch die Sortierung der Individuen beeinflusst. Eine spezielle Vorgehensweise konnte bei der Methode von NSGA-II die Komplexität von $\mathcal{O}(MN^3)$, wobei M die Anzahl der

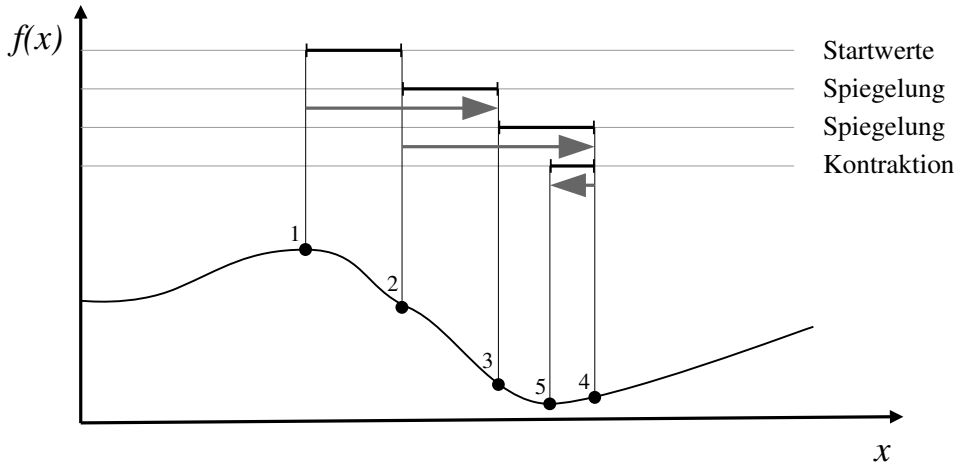

Abb. 2.13: Eindimensionale DS-Optimierung

Zielfunktionen und N die Größe der Population ist, auf $\mathcal{O}(MN^2)$ senken. Diese Sortierungsmethode, die auch die Behandlung von Schranken beinhaltet, wurde auch in GDE3 übernommen. Eine detaillierte Beschreibung, Vergleiche mit anderen Algorithmen und Ansätze zu geeigneten Einstellungen für bestimmte Probleme können bei Deb et al. (2002) nachgelesen werden.

2.2.4 Weitere Methoden

Die *Downhill-Simplex-Methode (DS)* von Nelder und Mead (1965) ist eine geometrische Optimumssuche. In Abbildung 2.13 ist der Ablauf anhand einer eindimensionalen Zielfunktion dargestellt. Für den Suchraum einer eindimensionalen Zielfunktion $f(x)$ ist ein Simplex ein Polyeder mit zwei Ecken, also eine Linie. Der Algorithmus beginnt mit einer zufälligen Verteilung zweier Startpunkte (1, 2) und berechnet die Zielfunktionswerte ($f(x_1)$, $f(x_2)$) an diesen Punkten. Anschließend wird der Punkt mit dem schlechtesten Zielfunktionswert (in diesem Fall Punkt 1, da $f(x_1) > f(x_2)$) am besseren Punkt gespiegelt. Der neu entstandene Punkt (3) wird evaluiert, also der Zielfunktionswert an der Stelle berechnet, und als neuer Punkt übernommen, falls eine Verbessung eingetreten ist. Andernfalls wird der nächstschlechteste Punkt der ursprünglichen zwei Punkte gespiegelt, evaluiert und gegebenenfalls als neuer Punkt übernommen. Hier ist jedoch eine Verbesserung zu sehen, weswegen nun Punkt 2 an Punkt 3 gespiegelt wird, was im wiederum verbesserten Punkt 4 resultiert. Zur Anpassung der Schrittweite der Suche kann die Strecke, um die ein Punkt bei der Spiegelung verschoben wird, gestreckt oder gestaucht werden, falls sonst keine Verbesserung eintritt, sodass der Simplex schließlich im Optimum auf einen Punkt zusammenschrumpft. Dies ist in unserem Beispiel im schließlich letzten Schritt von Punkt 4 nach Punkt 5 dargestellt.

Die Partikel-Schwarm-Optimierungsmethode (*Particle Swarm Optimizer, PSO*) von Kennedy und Eberhart (1995) simuliert das Schwarmverhalten von Vögeln oder Fischen auf der Suche nach Futter. Dabei wird die Erkenntnis ausgenutzt, dass die Information des Ortes einer Futterstelle

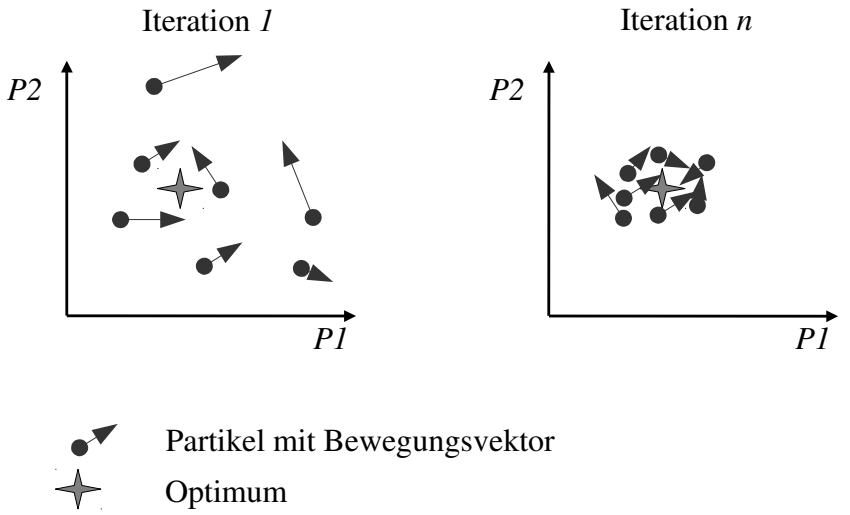

Abb. 2.14: Particle Swarm Optimization

innerhalb des Schwarms weitergegeben wird und so die einzelnen Individuen, hier auch Agenten genannt, von dem Auffinden einer solchen durch ein Individuum des Schwarms profitieren.

Zu Beginn der Suche wird eine Population mit fester Anzahl an Agenten, der Schwarm, im Suchraum mit bestimmten Positionen und Geschwindigkeiten initialisiert. Die grundlegende Regel, nach der sich die Agenten bewegen, ist, ihre Geschwindigkeit und Bewegungsrichtung an die der direkten Nachbarn anzupassen. Dieses allein bewirkt bereits die klassische Schwarmbewegung (Kennedy und Eberhart (1995)). Für die Futtersuche wird zusätzlich ein „Maisfeld“-Vektor eingeführt. Dieser stellt das soziale „Gedächtnis“ des Schwarms dar. Er zeigt auf die beste bisher vom Schwarm gefundene Position, und der Bewegungsvektor jedes Agenten wird in jeder Iteration mit einer geringen stochastischen Variation in die Richtung dieser Position verändert. Gleichzeitig erinnert sich jedes Individuum an die beste Position, die es bisher selbst besucht hat und ändert seinen Bewegungsvektor auf dieselbe Weise in dessen Richtung. Durch die Kombination dieser Mechanismen verbreitet sich die Information über eine vielversprechende Position, sodass eine globale Optimumssuche möglich wird. Je nach Gewichtung des Schwarmgedächtnisses und des „persönlichen“ Gedächtnisses kann der Algorithmus zwischen schneller Suche mit der Gefahr zu schneller Konvergenz in lokalen Optima und umfassender Suche mit langsamer Konvergenz konfiguriert werden. Der Verlauf einer solchen Optimierung ist in Abbildung 2.14 skizziert.

Von PSO existiert auch eine Variante für die multikriterielle Optimierung, die sich einen Ansatz mittels dynamischer Gewichtung der einzelnen Zielfunktionen zunutze macht (Parsopoulos und Vrahatis (2002)). Ein anderer Ansatz wird von Coello Coello und Lechuga (2002) vorgeschlagen, der ohne Gewichtung auskommt und zur Bestimmung der Flugrichtung der Partikel das Paretodominanzkonzept verwendet.

Der Algorithmus der Simulierten Abkühlung (*Simulated Annealing Method, SA*) ist ein Optimierungsverfahren, das auf Analogien aus den Werkstoffwissenschaften basiert (Kirkpatrick et al. (1983)). Bei der Erstarrung einer Schmelze hängt die erreichbare Ordnung der Kristallstruktur stark von der Abkühlgeschwindigkeit ab. Sehr schnelle Abkühlung der Schmelze resultiert in

einem ungeordneten, amorphen Zustand, während eine hinreichend langsame Erstarrung das Erreichen des thermodynamischen Gleichgewichts in einer geordneten Struktur ermöglicht. Dieser Gleichgewichtszustand entspricht dem Optimalzustand mit geringster innerer Energie.

Im SA-Algorithmus wird dieser Vorgang folgendermaßen modelliert. Eine Startkonfiguration wird in einem Parameter variiert und das resultierende Individuum wird als neue Lösung akzeptiert, falls sich eine Verbesserung im Zielfunktionswert ergeben hat. Mit einer gewissen Wahrscheinlichkeit, die von dem Parameter „Temperatur“ abhängig ist, werden jedoch auch Lösungen mit einem schlechteren Zielfunktionswert akzeptiert. Ist die Temperatur hoch, ist die Wahrscheinlichkeit hoch, dass dies geschieht. Im Verlauf der Optimierung wird die Temperatur nach bestimmten Vorgaben gesenkt und die anfangs eher chaotische Suche („random walk“) wird eingeschränkt, indem schlechtere Lösungen immer seltener akzeptiert werden und die Suche schließlich in einem Optimum konvergiert (Kirkpatrick et al. (1983)). Bei der Einstellung der „Abkühlgeschwindigkeit“ ist immer zwischen einer schnellen Suche, die möglicherweise in einem lokalen Optimum endet, und einer langsameren mit größerer Wahrscheinlichkeit, das globale Optimum auf Kosten von mehr Zielfunktionsevaluationen zu finden, abzuwägen (Dréo (2006)). Seit der Veröffentlichung des ursprünglichen Algorithmus wurden mehrere Verbesserungen vorgeschlagen (z.B. von Ingber (1996)). Am ITS wurde der SA-Algorithmus unter anderem zur Formoptimierung von einfachen Labyrinthdichtungen verwendet (Schramm (2010)).

2.3 Modellierung des Durchflussverhaltens

Im vorigen Kapitel wurden verschiedene Methoden zur Optimumssuche in Zielfunktionen vorgestellt. Außer einer einfachen mehrdimensionalen quadratischen Funktion, der Testfunktion F1 von De Jong (1975) (Gl. 2.5), wurde noch nicht weiter auf Zielfunktionen eingegangen. In diesem Abschnitt sollen daher mehrere Möglichkeiten zur Vorhersage des Leckageverhaltens von Labyrinthdichtungen vorgestellt werden. Der Abschnitt bildet damit die Grundlage für die Auswahl geeigneter Methoden, die anschließend in der Optimierungsumgebung eingesetzt werden sollen.

2.3.1 Integrale Korrelationen

Korrelationen beschreiben, wie verschiedene Ein- und Ausgangsgrößen in Wechselwirkung miteinander stehen. Im Falle von Labyrinthdichtungen soll in den meisten Fällen der Einfluss von Eingangsgrößen wie den Geometrieparametern und strömungs- und thermodynamischen Randbedingungen auf die Ausgangsgröße Durchflussbeiwert beschrieben werden. Andere Vorhersagen, wie z.B. der Einfluss der Dichtung auf die Rotordynamik, sind auch denkbar. Anhand von umfangreichen Messprogrammen werden diverse Eingangsparameter variiert, die Ausgangsgröße gemessen und schließlich auf Basis der Messergebnisse die Eingangs- mit der Ausgangsgröße durch eine Regressionsanalyse korreliert. Dabei werden üblicherweise physikalische Zusammenhänge in die Korrelation integriert, im Gegensatz zur rein datenbasierten Modellbildung.

Martin (1967) untersuchte am damaligen Versuchskraftwerk der Technischen Hochschule Karlsruhe (heute Institut für Thermische Strömungsmaschinen am Karlsruher Institut für Technologie) eine Vielzahl unterschiedlicher Labyrinth- und Spaltdichtungen, wie sie in Dampfturbinen verwendet wurden. Der Fokus lag neben dem Verständnis der Strömungsformen in den Dichtungen auf der Bestimmung von Leckageverlusten. Insgesamt wurden bei den Versuchen ca. 8000 Messpunkte erzeugt.

Bei der theoretischen Herleitung der Haupteinflussparameter der Geometrie auf den Leckagemassenstrom konnte die relative Spaltweite s/a als wichtigster Parameter gefolgt von der relativen Teilung t/a (mit $a = S_H + s$ als Gesamthöhe der Dichtung) identifiziert werden. Nach Martin (1967) sind diese beiden Parameter, unabhängig von der Bauform der Dichtung, maßgeblich für die Leckage bei gleichem Druckverhältnis. Seine Formel für die Leckagemassenstromberechnung lautet wie folgt.

$$\dot{m} = \frac{1}{\sqrt{A + k\sqrt{B \cdot s/a + C}}} \cdot \sqrt{\frac{s/a}{t/a}} \cdot \frac{2\pi R_i s}{\sqrt{n}} \cdot \sqrt{\frac{\rho \cdot (p_{t0}^2 - p_{sz}^2)}{p_{t0}}} \tag{2.6}$$

Die Koeffizienten A, B, C und k sind empirische Parameter, die anhand der Messdaten für die jeweiligen Geometrien bestimmt wurden. Durchblicklabyrinthdichtungen mit glattem Stator besitzen den Untersuchungen nach die Koeffizienten $A = 0{,}062$, $B = 0{,}00678$, $C = -0{,}000652$ und $k = 1$. Den Gültigkeitsbereich gibt Martin mit $0{,}1 < s/a < 0{,}55$ an. Er deckt somit den Bereich größerer Spaltweiten ab, der für Dampfturbinen ausreichend ist. In Gasturbinen, insbesondere Flugtriebwerken, kann dieser Bereich jedoch im Verlaufe einer Optimierung in Richtung geringerer Spaltweiten verlassen werden.

Dörr (1985) entwickelte am ITS eine Korrelation zur Beschreibung des Leckageverlustes von ebenen Durchblicklabyrinthdichtungen auf Basis von experimentellen Untersuchungen. Dabei wurden maßstäblich vergrößerte Modelle verwendet, wobei der Einfluss der Vergrößerung ebenfalls untersucht wurde. Die Korrelation, die in Gleichung 2.7 sowie 2.8a bis 2.8d wiedergegeben ist, wird von Doerr selbst als „unhandlich“ bezeichnet. Sie gibt zwar die Messergebnisse des Autors mit guter Genauigkeit wieder, ist jedoch trotz ihrer Komplexität nur für Durchblicklabyrinthdichtungen geeignet.

$$C_D = C_{D1} \cdot C_{D2} \cdot C_{D3} \cdot C_{D4} \tag{2.7}$$

$$C_{D1} = 1 - exp(-(-10 \cdot s/t + 10{,}4) \cdot (\pi + 0{,}778 \cdot s/t - 0{,}873)) \tag{2.8a}$$

$$\begin{aligned} C_{D2} = {} & 0{,}179\,\pi + ((0{,}354\,n^2 - 4{,}49\,n + 8{,}02) \cdot (s/t)^2 \\ & + (-0{,}118\,n^2 + 1{,}622\,n - 2{,}876) \cdot s/t \\ & + (0{,}01276\,n^2 - 0{,}216\,n + 1{,}03)) \end{aligned} \tag{2.8b}$$

$$C_{D3} = 1 - exp(0{,}544\,\pi - 2{,}8) \tag{2.8c}$$

$$\begin{aligned} C_{D4} = {} & 1 + (-0{,}0975 + 4{,}907\,s/t - 0{,}00783\,n) \cdot (\pi - 1)^{1{,}754\,s/t + 0{,}0167\,n)} \\ & \cdot exp((-0{,}37 - 97{,}3\,s/t + 0{,}458\,n) \cdot (\pi - 1)) \end{aligned} \tag{2.8d}$$

Waschka (1991) erweiterte die Korrelation von Dörr (1985) um den Einfluss der Rotation auf die Durchflussbeiwerte und den Wärmeübergang. Eine Änderung des Durchflussbeiwertes ist erst ab Impulsverhältnissen $I = u/c_{ax} > 1$ messbar gewesen. Oberhalb dieses Wertes konnten kleinere C_D-Werte ermittelt werden, bevor bei sehr hohen Impulsverhältnissen dieser Abfall wieder abflachte. In Gleichung 2.9 bis 2.11 ist die Korrelation für die Änderung des Durchflussbeiwertes wiedergegeben.

$$\frac{C_D}{C_{D_0}} = C_1 + C_2 \cdot x + C_3 \cdot x^2 + C_4 \cdot x^3 + C_5 \cdot x^4 \tag{2.9}$$

mit

$$x = log\left(\frac{u}{c_{ax}} \cdot K\right) \tag{2.10}$$

falls

$$\frac{u}{c_{ax}} \cdot K \geq 1 \tag{2.11}$$

Die Konstanten gibt Waschka mit $C_1 = 0{,}999$, $C_2 = -0{,}071$, $C_3 = 0{,}1457$, $C_4 = -0{,}494$ und $C_5 = 0{,}1714$ an. Der Koeffizient K wurde für die unterschiedlichen Labyrinthformen folgendermaßen ermittelt:

- Durchblicklabyrinth: $K = 1{,}17$
- Konvergentes Stufenlabyrinth: $K = 1{,}06$
- Divergentes Stufenlabyrinth: $K = 1{,}08$
- Kammnutlabyrinth: $K = 1{,}04$

Die Erweiterung um die Rotationskorrektur erhöht die Komplexität von Dörrs Korrelation weiter. Sie stellt damit ein gutes Beispiel für eine Korrelation mit in ihrem Parameterraum exzellenter Vorhersagegenauigkeit dar. Der Aufwand, der zur Erstellung und zum Einpflegen weiterer Daten notwendig wäre, macht jedoch die Nachteile solcher “monolithischer” Verfahren deutlich.

2.3.2 Datenbasierte Modellbildung

Die datenbasierte Modellbildung ist ein Bereich des „Data Mining“. Sie soll hier als allgemeinere Methode der Korrelationserstellung vorgestellt werden. Data Mining umfasst Methoden, um aus einer bisweilen sehr großen Menge an Daten Informationen zu extrahieren. In unserem Fall interessiert besonders die Regression – neben der Klassifikation ein Bereich des überwachten Lernens. Beim überwachten Lernen sind sowohl die Eingangs- als auch die Ausgangsparameter a priori bekannt (z.B. Messdaten). Durch Regression der bekannten Lerndaten wird die Vorhersage einer kontinuierlichen Ausgangsgröße in Abhängigkeit der Eingangsparameter möglich. „Datenbasiert“ sagt aus, dass in die Modellbildung keinerlei Wissen um die physikalischen Zusammenhänge, die den Daten zugrundeliegen, einfließt. Die angewandten Methoden sind daher auch unabhängig von der Art des Vorgangs, der modelliert werden soll.

Künstliche Neuronale Netze (KNN) sind eine Variante der datenbasierten Modellbildung. Sie simulieren die Art und Weise, wie das menschliche Gehirn arbeitet. Dabei werden einfache Recheneinheiten (Neuronen) durch gewichtete Verbindungen verknüpft. Die Neuronen verknüpfen ihre jeweiligen Eingangsgrößen über bestimmte mathematische Funktionen mit einer Ausgangsgröße. Bei sogenannten Multilayer-Perceptron- (MLP-) Netzwerken sind die Neuronen in unterschiedlich vielen, mindestens jedoch drei Schichten angeordnet. In der Eingabeschicht werden die Eingabeparameter eingelesen. Die Anzahl der Neuronen in dieser Schicht ist gleich der Zahl der Eingabeparameter. In der zweiten und ggf. weiterer Schichten können unterschiedlich viele Neuronen enthalten sein. In der letzten, der sogenannten Ausgabeschicht sind wiederum soviele Neuronen enthalten, wie Ausgabegrößen berechnet werden sollen. In Abbildung 2.15 ist der Aufbau eines solchen MLP-Netzwerks vereinfacht dargestellt. Durch die Vielzahl an möglichen Kombinationen von Neuronen, Gewichtungsfaktoren, Schichtanzahl und Aktivierungsfunktionen der Neuronen können auch sehr komplexe Zusammenhänge durch diese Art der Modellbildung abgebildet werden.

Nach der Wahl der Neuronen- und Schichtenanzahl müssen die Aktivierungsfunktionen ausgewählt werden. Es kann sich bei diesen um eine Sprungfunktion, einen linearen Anstieg oder eine sigmoide Funktion handeln. Anschließend müssen die Gewichtungsfaktoren der Neuronen so eingestellt werden, dass sie vorhandene Lerndaten bestmöglich wiedergeben. Dabei ist eine Überanpassung unbedingt zu vermeiden, da ansonsten zwar die Lerndaten bei der Vorhersage perfekt getroffen werden, die echte zugrundeliegende Funktion jedoch zwischen den Lerndatenpunkten nur schlecht wiedergegeben wird - die Fähigkeit zur Interpolation geht in diesem Fall verloren. Für das Training der KNN, also der Einstellung der Gewichtungsfaktoren, stehen verschiedene Strategien zur Verfügung, die für sich wieder ein Optimierungsproblem darstellen. Eine Variante, die in dieser Arbeit verwendet wurde, beinhaltet das Zurückhalten von Daten aus dem Lerndatensatz, die nicht zum Training verwendet werden. Diese dienen nach dem Training als Testdatensatz, um Überanpassung zu erkennen. Diese Vorgehensweise wird Kreuzvalidierung genannt. Einen guten Überblick über diese Thematik und KNN im Allgemeinen bietet Yegnanarayana (2009).

Pychynski et al. (2010) verwendeten Künstliche Neuronale Netze, um die Leckage von Labyrinthdichtungen zu modellieren. Die zugrundeliegenden Lerndaten bestanden aus experimentellen

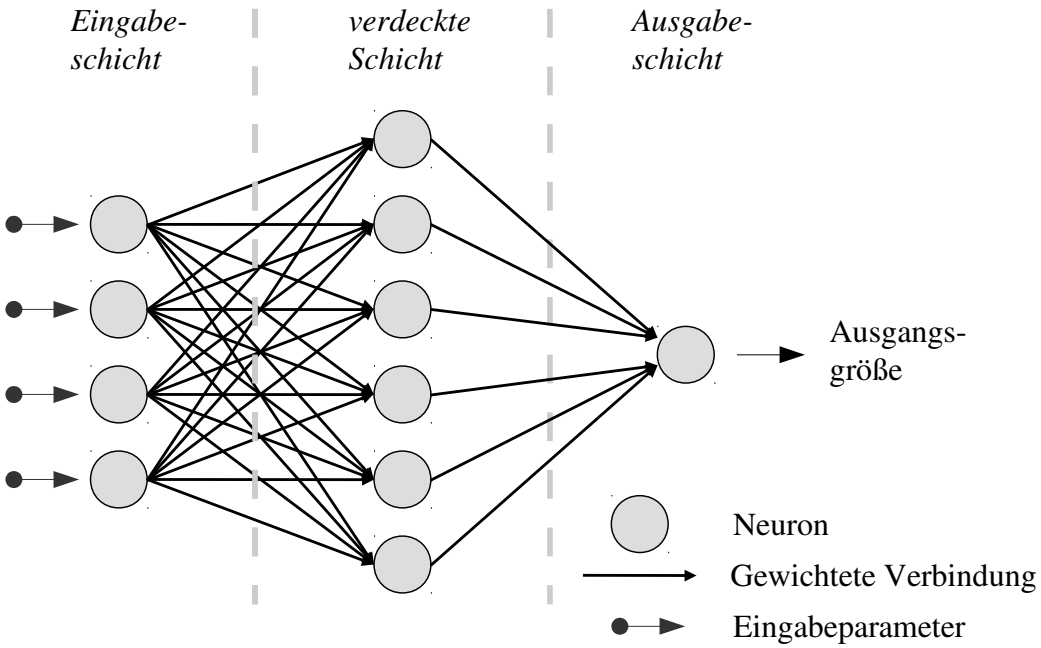

Abb. 2.15: Einfaches MLP-Netzwerk

und numerischen Ergebnissen, die im Rahmen einer umfangreichen Literaturrecherche und aus Aufzeichnungen von am Institut für Thermische Strömungsmaschinen in der Vergangenheit durchgeführten Untersuchungen zusammengetragen wurden. Dabei konnte demonstriert werden, dass ein KNN, über alle Messdatenpunkte und somit über verschiedene Labyrinthtypen gebildet, eine mittlere Abweichung von den Messdaten von 6,5% erreichen konnte. Ein KNN, das nur für Durchblicklabyrinthdichtungen erzeugt wurde, von denen der größte Anteil an Daten vorlag, erreichte 4% mittlere Abweichung und ein lokales Modell für konvergente Stufenlabyrinthe gar 1,4%. Das Training der KNN dauerte dabei jeweils weniger als 5 Minuten.

Die größten Unsicherheiten in der Vorhersage traten in den Bereichen auf, in denen nur eine geringe Abdeckung des Parameterraums durch Messdaten gegeben war. Eine möglichst umfassende Abdeckung des Raums ohne größere Lücken ist daher notwendig für eine präzise Vorhersage. Dabei sollen die Messdaten möglichst gleichmäßig verteilt sein, um eine Überbewertung besonders dicht besetzter Bereiche durch das Modell zu vermeiden. Wie bei anderen Formen der Modellbildung sollten die Messdaten frei von Ausreißern sein (Pychynski et al. (2010)).

Asok et al. (2007) untersuchten die Eignung von KNN zur Optimierung von Labyrinthdichtungen mit quadratischer oder gekrümmter Kammergeometrie, wobei sich die Krümmung auf die stromab liegende Kammerwand bezieht. Die Güte der Dichtungen wurde mittels eines Wirbelverlustkoeffizienten beschrieben. Eine Ausgangsgeometrie konnte um 75% im Druckverlustbeiwert verbessert werden. Dies ist vermutlich auf die deutlich verringerte Spitzenbreite zurückzuführen, die bei der Originalgeometrie etwa gleich der Kammerbreite und somit erheblich breiter war. Die Ablösung und Einschnürung über der Spitze fällt also größer aus, was die Labyrinthleckage verringert.

Gaußsche Prozesse (GP) werden in den letzten Jahrzehnten neben Verfahren wie KNN als

Regressionsmodell verwendet. GP können als eine Generalisierung einer Gaußschen Wahrscheinlichkeitsverteilung über einen finiten Vektorraum hin zu einem Funktionsraum unendlicher Dimension betrachtet werden (MacKay (1997)). Wie eine Gaußsche Wahrscheinlichkeitsverteilung über einen Mittelwert und eine Kovarianzmatrix beschrieben werden kann, wird ein GP über die Mittelwertsfunktion aller Funktionen und eine Kovarianzfunktion definiert. Ein GP stellt also die Generalisierung der eindimensionalen Gaußverteilung zum Unendlichdimensionalen dar. An jedem Datenpunkt liefert ein GP für eine Abbildung von $\mathbb{R}^n \rightarrow \mathbb{R}$ den Mittelwert aller Funktionswerte an dieser Stelle sowie eine Standardabweichung.

Viele Regressionsmodelle, auch Künstliche Neuronale Netzwerke, mit gegen unendlich gehender Anzahl an Neuronen können auf GP zurückgeführt werden (MacKay (1997)). Einen sehr guten Überblick und mathematische Hintergründe zu Gaußschen Prozessen bietet Rasmussen (2006). GP werden bei der Verwendung als Regressionsmodell auf Basis eines vorhandenen Datensatzes gebildet. Die einzelnen Datenpunkte können dabei mit einem Unsicherheitsmaß, dem sogenannten Konfidenzintervall, versehen sein, was insbesondere bei Messdaten sinnvoll ist, bei denen der geschätzte Messfehler angegeben wird.

In Abbildung 2.16 ist der Verlauf eines Gaußprozesses mit durch CFD-Simulationen erzeugten Daten dargestellt. Die CFD-Methode wurde hier als exakt betrachtet, d.h. die Daten weisen keinen Fehler auf. Im grauen Bereich verlaufen alle Funktionen des GP, die in das 95%-Konfidenzintervall fallen. An den bekannten Datenpunkten geht die Schar exakt durch den Funktionswert des Punktes, da der Fehler Null beträgt. Zwischen den bekannten Punkten ist die Unsicherheit größer und die Funktionsschar verbreitert sich, was sich in einem größeren Konfidenzintervall niederschlägt. Außerhalb der bekannten Daten wächst die Unsicherheit besonders stark an. Die Vorhersage des Gaußprozesses ist die Mittelwertsfunktion, die durch die durchgezogene Linie dargestellt wird.

El-Beltagy und Keane (2001) zeigten, dass sich Gaußsche Prozesse als Ersatzmodelle für aufwendige Zielfunktionen in der Optimierung eignen. Im Laufe der Optimierung wird die eigentliche rechenintensive Zielfunktion nur selten aufgerufen, um die Vorhersage des GP durch Hinzufügen weiterer Stützpunkte zu verbessern. Den Autoren nach hält sich der zusätzliche Aufwand für die Einarbeitung der neuen Datenpunkte in die GP in Grenzen, wenn davon ausgegangen werden kann, dass sich die Hyperparameter, die den GP definieren, durch die Zusatzinformationen nicht wesentlich ändern werden. Als Anwendungsbeispiel diente die Optimierung eines Fachwerkauslegers eines Satelliten mit einer Finite-Elemente-Methode als Zielfunktion.

Auch Büche et al. (2005) verwendeten Gaußsche Prozesse zur Modellierung einer aufwendigen (CFD-) Zielfunktion. Ihre Arbeit steht unter der Annahme, dass nur ein kleiner Teil der Population im verwendeten evolutionären Algorithmus von der eigentlichen Zielfunktion evaluiert werden muss und für den größten Teil die Schätzung mittels Gaußscher Prozesse als Ersatzmodell genügt, um das globale Optimum zu finden. Durch diese Vorgehensweise konnte die Optimierung von Kompressorschaufelprofilen hinsichtlich aerodynamischer Verluste im Vergleich zur rein CFD-basierten Optimierung beschleunigt werden.

Als Möglichkeit, die Vorteile unterschiedlicher Modelle zu kombinieren, beschreibt Oza (2009) Ensemble Data Mining Methoden, die mehrere möglichst komplementäre Modelle zusammen-

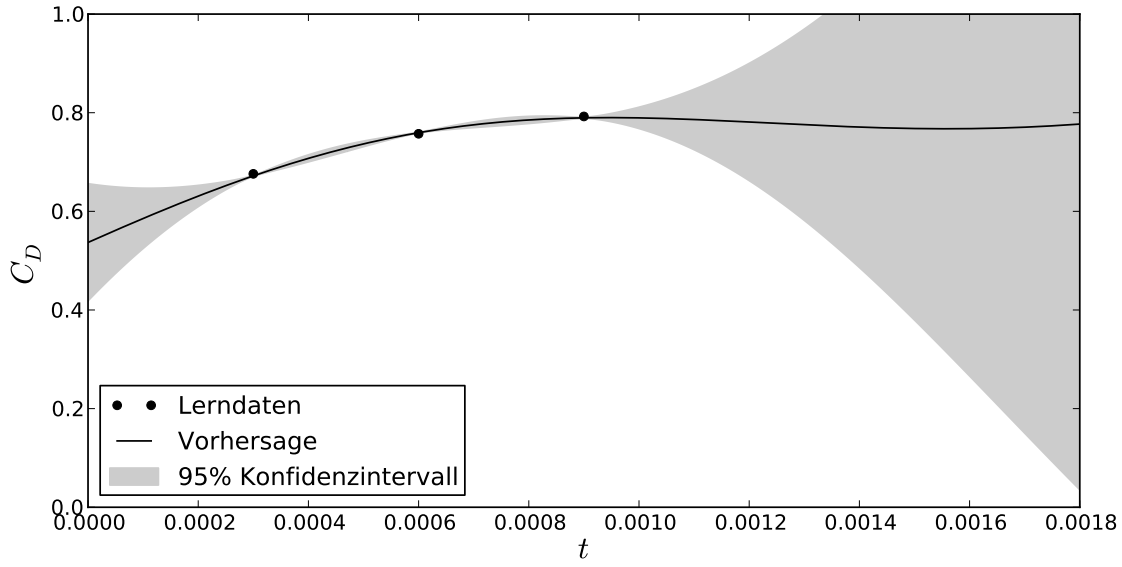

Abb. 2.16: Beispiel für die Vorhersage des Durchflussbeiwertes eines Durchblicklabyrinthes in Abhängigkeit der Teilung mittels eines Gaußschen Prozesses

führen. Er trifft dabei den Vergleich mit einem Komittee, dessen Mitglieder möglichst kompetent, aber gleichzeitig möglichst unterschiedlich sein sollten, um nicht bei jeder Entscheidung exakt übereinzustimmen (sonst wäre das Komittee unnötig), sondern Fehler eines einzelnen Mitglieds durch Überstimmung auszugleichen. Oza (2009) spezifiziert Trainingsmethoden, wie z.B. das Training mit entfernten Teildatensätzen, die bestimmte Charakteristiken enthalten, sowie auch Verfahren, um die unterschiedlichen Modelle zu kombinieren. Als Beispiele werden die Bildung des Durchschnitts aller Vorhersagen, des gewichteten Durchschnitts z.B. durch Verwendung eines „Gating Networks“ oder der Einsatz eines Basis- und mehrerer Spezialmodelle genannt.

2.3.3 Bulk-Flow-Modelle

Sogenannte „Bulk-Flow“-Modelle stellen den Übergang von den integralen Korrelationen zur numerischen Strömungsmechanik (CFD) dar. Behandeln die erstgenannten eine Labyrinthgeometrie in ihrer Gesamtheit und CFD sehr viele Kontrollvolumina einzeln (mehrere tausend bis hunderttausend schon im zweidimensionalen Fall, vgl. auch Kap. 5.3.3), liegen die Bulk-Flow-Modelle dazwischen. Sie können anhand ihrer Anzahl an Kontrollvolumen pro Labyrinthkammer in Ein-, Zwei- und Dreivolumenmodelle unterschieden werden.

Einvolumenmodelle modellieren das Strömungsverhalten in der Dichtung für jede Kammer einzeln. Die unterschiedlichen Effekte, die in einer Kammer auftreten, werden dabei zusammengefasst. Zwei- und Dreivolumenmodelle verfeinern die Betrachtung der einzelnen Effekte durch eine höhere Auflösung. So können die Strahleinschnürung über den Spitzen, insbesondere die Sonderstellung der ersten Spitze, der Wandstrahl über der Kammer und der Kammerwirbel in ihrer Wirkung gesondert berechnet werden. Die Korrelationen für die Kontrollvolumina

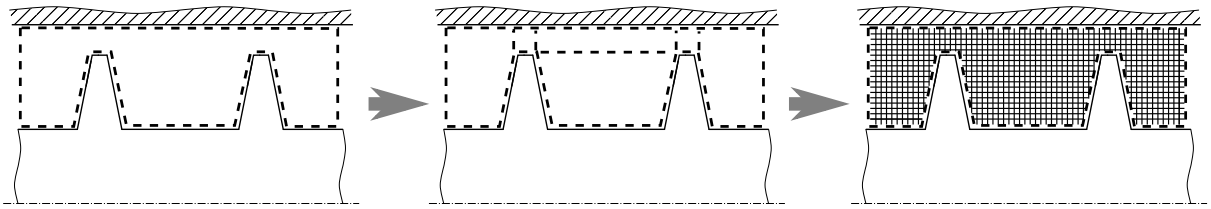

Abb. 2.17: Vergleich der Kontrollvolumina von integraler Korrelation (links), 3-Volumen-Bulkflow-Modell (Mitte) und CFD (rechts)

müssen aufwendig anhand von Modellmessungen kalibriert werden. Da der Großteil der Bulk-Flow-Modelle außerdem entworfen wurde, um den Drallverlauf der Strömung und damit das rotordynamische Verhalten von Labyrinthdichtungen zu modellieren, fand diese Form der Modellbildung in dieser Arbeit keine Verwendung. Einen guten Überblick über die Arbeitsweise einiger Bulk-Flow-Modelle und weitere Referenzen bietet Denecke (2007).

2.3.4 Numerische Strömungsmechanik (CFD)

Für einen umfassenden Überblick über den Einsatz von CFD zur Simulation der Strömung in Labyrinthdichtungen sei auf Denecke (2007) verwiesen. Die dort vorgestellten Methoden verwenden weitestgehend das k,ϵ-Turbulenzmodell. In den letzten Jahren kamen vermehrt auch andere Methoden zum Einsatz. Dies soll in diesem Abschnitt gezeigt werden.

Bianchini et al. (2013) untersuchten Honigwabendichtungen, eine Variante der Spaltdichtung mit Wabenstrukturen auf dem Stator und glattem Rotor zur Evaluierung eines neuen Versuchsaufbaus. Als numerisches Modell wurde ANSYS CFX® unter Anwendung des k,ω-SST Modells eingesetzt, das geringere Anforderungen an die Auflösung der Wandgrenzschicht stellt als das klassische k,ϵ-Modell (Menter (1993) und Menter et al. (2003)). Auch andere Autoren bevorzugen das k,ω- oder k,ω-SST-Modell (He et al. (2012) und Chougule et al. (2006)).

Großen Aufwand betrieben Jefferson-Loveday et al. (2013) mit ihren instationären Simulationen der Labyrinthströmung. Sie verglichen die Möglichkeiten von instationären Reynolds-gemittelten Navier-Stokes- (URANS-) Methoden mit hochauflösenden Modellen wie Detached Eddy Simulation (DES) und lokalem Einsatz der Grobstruktursimulation (Large Eddy Simulation - LES). Die berechneten Geschwindigkeiten und turbulenten Schwankungen deckten sich sehr gut mit experimentellen Ergebnissen. Die notwendige Rechenleistung und der Speicherbedarf für LES und DES rechtfertigen aber eine Anwendung dieser für reine Durchflussberechnungen nicht.

Zur Simulation der Strömung in Kammnut-Labyrinthdichtungen mit starren und flexiblen Spitzen wurde von Herrmann et al. (2013) das Spalart-Allmaras-Turbulenzmodell verwendet. Die Grenzschicht musste hier sehr fein aufgelöst werden, um ein y^+ von $1 \ldots 2$ zu erreichen. Eine feine Auflösung war aufgrund der speziellen Geometrie, die sich nach dem Anstreifen der Dichtspitzen ergab, für das Nachvollziehen der experimentellen Ergebnisse jedoch ohnehin notwendig. Es zeigte sich nämlich für bestimmte Geometrien, dass sich nur im Falle einer sehr präzisen Nachbildung der echten Geometrie, insbesondere der Radien an den Dichtspitzen (vgl. auch Zimmermann et al. (1994)), im CFD-Modell die gleichen Durchflussbeiwerte wie im Experiment einstellten

(Herrmann (2013). Obwohl das genannte Turbulenzmodell ursprünglich für Außenströmungen erdacht wurde (Spalart und Allmaras (1992)), deckten sich die Simulationsergebnisse sehr gut mit den Experimenten.

3 Zielsetzung

Im vorherigen Kapitel wurden die vielfältigen Möglichkeiten zur Bestimmung der Leckage von Labyrinthdichtungen erläutert. Es wurde gezeigt, wie sich diese in Aufwand, Genauigkeit und Gültigkeitsbereich unterscheiden. So eignen sich Korrelationen sehr gut zur effizienten Vorhersage innerhalb eines begrenzten Parameterraums. Bei den erwähnten Bulk-Flow-Modellen lässt der Gültigkeitsbereich durch erhöhte Komplexität bereits deutlich erweitern. CFD-Methoden bieten schließlich vor den Experimenten einen sehr guten Kompromiss zwischen Aufwand und Vorhersagegenauigkeit. Verschiedene Optimierungsverfahren wurden vorgestellt und deren Grenzen aufgezeigt. Bei der Wahl eines geeigneten Verfahrens muss zwischen der hohen Geschwindigkeit von gradientenbasierten Verfahren und der hohen Wahrscheinlichkeit, das globale Optimum zu finden, die beispielsweise den evolutionären Verfahren zugrundeliegt, abgewogen werden.

Martin (1967) beschreibt die Suche nach der optimalen Labyrinthdichtung folgendermaßen: „Gesucht ist die Dichtungsbauform, die bei gegebenem Durchmesser der abzudichtenden Welle, bei gegebener Lässigkeit sowie gegebenen Zustandsgrößen des Strömungsmediums vor und nach der Dichtstrecke die kürzeste axiale Baulänge l aufweist.“ Diese Definition kann verwendet werden, um eine Dichtung zu entwerfen, die beispielsweise den Kühlluftmassenstrom für ein bestimmtes Bauteil einstellt. In dieser Arbeit soll im Gegensatz dazu die Dichtungsgeometrie gesucht werden, die bei gegebenem Bauraum und Zustandsgrößen die beste Dichtwirkung aufweist.

Bisher wurden die Effekte, die einen Einfluss auf die Labyrinthleckage haben, in den Untersuchungen nur einzeln betrachtet. Versuche, eine Labyrinthgeometrie zu optimieren, beschränkten sich auf die Leckage im unverschlissenen Zustand. In dieser Arbeit soll daher untersucht werden, welche Methoden dazu geeignet sind, um ermitteln zu können, welche Geometrien im Neuzustand wie auch im verschlissenen Zustand gleichzeitig eine optimale Dichtwirkung besitzen. Eine Betrachtung der Gesamtheit aller Effekte gemeinsam ist daher entscheidend. Auf diese Weise soll die bestehende Lücke in Hinblick auf eine Auslegung für die gesamte Lebensdauer der Turbomaschine geschlossen werden.

Um dieses Ziel zu erreichen, sollen verschiedenste Modelle und Optimierungsalgorithmen in einem Computerprogramm zusammengefasst werden, um sie im Einzelnen und gemeinsam untersuchen zu können. Ein Hauptaugenmerk liegt dabei einerseits auf der Anwendbarkeit für eine große Anzahl unterschiedlicher Dichtungsgeometrien. Andererseits soll, wie angesprochen wurde, die Möglichkeit gegeben sein, den besten Kompromiss aus geringer Leckage sowohl im Neuzustand als auch im verschlissenen Zustand der Dichtung ermitteln zu können. Dieser Ansatz bietet als entscheidenden Vorteil, dass sich hieraus die beste Dichtungsgeometrie für den gesamten Lebenszyklus ergibt.

Um das Ziel zu erreichen, müssen folgende Fragen beantwortet werden.

- Welche Methoden sind zur Leckagevorhersage möglichst beliebiger Dichtungsgeometrien geeignet?
- Welche Optimierungsalgorithmen eignen sich für eine a priori unbekannte Zielfunktionslandschaft?

- Wie können die unterschiedlichen Programmteile am besten kombiniert werden?
- Welche Geometrie bietet letztendlich den besten Kompromiss aus geringer Leckage im neuen und verschlissenen Zustand?

4 Methodik

Im vorherigen Kapitel wurde die Zielsetzung skizziert. Es gilt, zu ermitteln, welche Dichtungsgeometrie sich innerhalb bestimmter Randbedingungen am besten dazu eignet, sowohl im Neuzustand als auch im verschlissenen Zustand eine minimale Leckage darzustellen. Dazu sind Methoden und Prozesse zu identifizieren und zu bewerten, mit deren Hilfe eine systematische Ableitung geeigneter Geometrien möglich ist. Als erste Hypothese muss davon ausgegangen werden, dass sich die optimalen Geometrien für die beiden Zustände einzeln betrachtet unter Umständen stark unterscheiden werden. Eine Geometrie, die über die gesamte Lebensdauer eine geringe Leckage besitzt, wird somit voraussichtlich ein Kompromiss sein. Es ist zu erwarten, dass dieser sowohl im Neuzustand als auch gegen Ende des Lebenszyklus etwas schlechter abschneiden wird als Dichtungen, die explizit für den Neuzustand beziehungsweise für den verschlissenen Zustand ausgelegt wurden. Diese Abweichungen gilt es zu minimieren.

Der Weg zur Lösung dieses Problems beinhaltet zunächst die Wahl von Methoden, die sich zur Berechnung des Leckagemassenstroms von Labyrinthgeometrien eignen. Zum anderen muss unter Berücksichtigung der möglichen Parameter und Zielfunktionen ein Satz von Optimierungsalgorithmen ausgewählt werden, der es möglich macht, die Bandbreite von aufwendiger globaler Suche bis hin zur schnellen Konvergenz abzudecken.

Um die Aufgabe zu lösen, wurde ein modularer Programmaufbau gewählt. Standardisierte Schnittstellen zwischen den einzelnen Modulen sollen erlauben, beliebige Vorhersageverfahren miteinander zu verknüpfen. Das Programm soll aus einem Steuerungsmodul mit grafischer Bedienoberfläche, einer Auswahl an Optimierungsmodulen sowie einer Reihe von Zielfunktionsmodulen zur Bestimmung der Labyrinthleckage bestehen. Da die Optimierungsmodule keine Kenntnis über den Charakter der Zielfunktionen haben sondern allein durch die Suche einer geeigneten Kombination von Parameterwerten danach streben, den Zielfunktionswert zu minimieren, wird es möglich, eine Vielzahl von Optimierungsproblemen zu lösen. Das Programm wird damit also keineswegs auf die Optimierung von Labyrinthdichtungen beschränkt sein. Es eröffnet die Möglichkeit, auch weitere Minimierungsprobleme, die auf eine Variation von Parametern zurückgeführt werden können, lösen zu können.

4.1 Auswahl der Methoden zur Leckageberechnung

In Kapitel 2 wurden unterschiedliche Ansätze zur Berechnung des Massenstroms beziehungsweise des Durchflusskoeffizienten von Labyrinthdichtungen unter gegebenen Randbedingungen vorgestellt (vgl. Gln. 2.1 bis 2.4). Dabei wurde unterschieden zwischen

- Korrelationen, die aus einer oder mehreren algebraischen Gleichungen bestehen und somit sehr wenig Rechenzeit beanspruchen,
- Bulk-Flow-Modellen, die die Entwicklung der Strömung in einzelnen Bereichen berechnen und etwas aufwendiger sind,

- modernen Methoden der datenbasierten Modellbildung sowie
- der numerischen Strömungsmechanik als rechnerisch aufwendigstem Modell.

Ein quantitativer Vergleich der Methoden wird in Abbildung 2.17 dargestellt. In diesem Kapitel sollen die Vor- und Nachteile der Methoden herausgestellt werden, um eine Entscheidungsbasis für die Auswahl zu erstellen.

4.1.1 Integrale Korrelationen

Streng genommen zählen Korrelationen, die auf Messdaten basieren und somit nichts anderes als einfache Regressionsmodelle darstellen, zur datenbasierten Modellbildung. Sie werden jedoch hier gesondert betrachtet, da sie in der Regel Vorwissen über die physikalischen Zusammenhänge beinhalten.

Ein einfaches Regressionsmodell ist zum Beispiel die „Summe der kleinsten Fehlerquadrate“, bei der ein Polynom derart an den Verlauf der Messdaten angepasst wird, dass die Summe der Quadrate der Differenz zwischen Vorhersage durch das Polynom und den Messwerten an jeder Messstelle minimal wird. Durch die Einflüsse der unterschiedlichen Strömungsphänomene im Labyrinth ist ein einfaches Polynom meist nicht ausreichend, um die Messdaten genau wiederzugeben und kompliziertere Gleichungen, wie zum Beispiel Gln. 2.7 bis 2.8d, sind notwendig. Die resultierenden Gleichungen sind unter Umständen recht kompliziert (vgl. Kapitel 2.3). In der Vorhersage sind diese Korrelationen jedoch sehr effizient.

Ein entscheidender Nachteil ist die schlechte Erweiterbarkeit von Korrelationen. Da die zugrundeliegenden Messdatentupel aus begrenzten Bereichen des Parameterraums stammen, ist in der Regel mindestens eine Anpassung der Polynomkoeffizienten, möglicherweise aber sogar ein Hinzufügen weiterer Terme erforderlich, um neue Messdaten aus anderen Bereichen in die Korrelation zu integrieren. Ein weiterer Nachteil bei deren Verwendung in einem globalen Optimierungsprogramm ist, dass unterschiedliche Autoren unterschiedliche Eingangsparameter und Ausgabegrößen verwenden und somit das Hinzufügen weiterer Korrelationen zur Erweiterung des Gültigkeitsbereiches einen Mehraufwand in Form notwendiger Parametertransformationen bewirkt.

Um einen Vergleich von verschiedenen Methoden zu ermöglichen, wurden in den Modulen für die Zielfunktionsevaluation zwei Korrelationen implementiert. Sowohl die Korrelation von Dörr (1985) als auch die von Martin (1967) beschreiben durch die Gleichungen 2.7 bis 2.8d sowie Gleichung 2.6 den Leckageverlust von Durchblicklabyrinthdichtungen in Abhängigkeit von der Geometrie und dem Druckverhältnis π. Beide verwenden als ersten Parameter zur Labyrinthbeschreibung die Spitzenzahl n. Die Kammergeometrie wird bei Dörr mit dem Verhältnis aus Spaltweite zu Teilung s/t und bei Martin mit der dimensionslosen Teilung t/S_H und dimensionslosen Spaltweite s/S_H beschrieben. Die Einflüsse weiterer Geometrieparameter werden von beiden Ansätzen vernachlässigt. Im Detail wurden die Arbeiten bereits in Kapitel 2 vorgestellt.

4.1.2 Datenbasierte Modellbildung

Künstliche Neuronale Netze

Künstliche Neuronale Netze (KNN) stellen eine besondere Form von Regressionsmodellen dar. Ein großer Vorteil von Künstlichen Neuronalen Netzen gegenüber Korrelationen ist deren Flexibilität. Durch die vielen Möglichkeiten, die sich bei der Vernetzung und Gewichtung zwischen den einzelnen Neuronen ergeben, lassen sich nahezu beliebige Zusammenhänge zwischen den Lerndaten wiedergeben. Da die Anlernphase automatisiert erfolgen kann, ist ein nachträgliches Hinzufügen neu gewonnener Daten ohne großen Arbeitsaufwand möglich. Mit wachsender Anzahl an Datentupeln und Neuronen wächst die benötigte Rechenzeit für das Training der KNN. Da das Training im Vergleich zur Abfrage des KNN im Rahmen der Anwendung auf Optimierungsprobleme wesentlich seltener vorkommt, fällt dieser Nachteil kaum ins Gewicht. Die Vorteile gegenüber Korrelationen überwiegen somit. Die Möglichkeit, KNN als Zielfunktion bei der Optimierung verwenden zu können, wurde mit der Integration der FANN-Bibliothek (FANN: Fast Artificial Neural Networks, Nissen et al. (2014)) geschaffen.

Gaußsche Prozesse

Gaußsche Prozesse (GP) eignen sich gut als Regressionsmodelle für Anwendungen, bei denen es sinnvoll ist, gemeinsam mit der Vorhersage auch eine Schätzung für die Genauigkeit dieser Vorhersage zu erhalten. Die Kombination aus Mittelwertsfunktion, die den wahrscheinlichsten Verlauf der Erwartungswerte wiedergibt, und Kovarianzfunktion, die das Konfidenzintervall für alle Funktionswerte beschreibt, bietet genau diesen Vorteil. In der Nähe von bekannten Funktionswerten, also Datenpunkten aus den Lerndaten, strebt das Konfidenzintervall gegen null, wenn der Datenpunkt keinen (Mess-)Fehler aufweist. Mit wachsender Entfernung zu den bekannten Punkten wächst das Konfidenzintervall. Für die Anwendung in einer Optimierungsumgebung bedeutet dies, dass ein Wert für ein Konfidenzintervall definiert werden kann, oberhalb diesem der Vorhersage des Gaußschen Prozesses nicht mehr vertraut wird. Die Vorhersage kann dann mit einem Strafwert belegt oder eine Ausweich-Zielfunktion gestartet werden, die für das betrachtete Individuum einen exakteren Wert berechnen kann.

Gaußsche Prozesse haben wie die Künstlichen Neuronalen Netze den Vorteil, dass sie in der Vorhersage sehr schnell sind. Jedoch müssen sie im Voraus auf Basis von Trainingsdaten angelernt werden. Im Vergleich zu KNN ist dieser Prozess bei den Gaußschen Prozessen erheblich aufwendiger. Die benötigte Rechenzeit und der Speicherbedarf wachsen mit $\mathcal{O}(n^3)$, wobei n die Anzahl der Lerndatentupel darstellt (MacKay (1997)). Es existieren allerdings Algorithmen, die dieses Problem teilweise umgehen, indem nicht alle Lerndaten gleichzeitig betrachtet werden.

4.1.3 Numerische Strömungsmechanik (CFD)

Die numerische Strömungsmechanik stellt die flexibelste Methode zur Vorhersage der Leckageverluste dar. Der Leckagemassenstrom beliebiger Geometrien kann ermittelt werden, ohne

dass eine Grundlage durch Messdaten vorhanden sein muss, indem die gesamte Durchströmung der Dichtung simuliert wird. Selbst mit einfachen numerischen Methoden wie stationärer, zweidimensionaler Betrachtung der Strömung und Lösung der Reynolds-gemittelten Navier-Stokes-Gleichungen (engl. Reynolds-averaged Navier-Stokes oder RANS-Gleichungen) mit einem Zweigleichungsturbulenzmodell ist der Rechenaufwand jedoch um einige Größenordnungen höher als bei den datenbasierten Methoden.

Bei der Suche nach einer optimalen Geometrie ist es essentiell, den Optimierungsalgorithmen den nötigen Freiraum zu geben, also den Parameterraum, in dem eine Suche erlaubt ist, nicht zu sehr einzuschränken. Dies würde die Möglichkeit, neue, bessere Lösungen zu finden, verringern. Ein grundsätzliches Problem bei datenbasierten Methoden ist der begrenzte Gültigkeitsbereich, der den Suchraum einschränkt. Es ist also unbedingt notwendig, bei der Optimierung zumindest eine, wenn auch aufwendige, Methode bereitzustellen, die in der Lage ist, die Güte bisher unbekannter Parameterkombinationen zu ermitteln. Aus diesem Grund wurde eine CFD-Routine implementiert, die den Arbeitsablauf von Parameterinterpretation über Erstellung des Rechennetzes, Setzen der Randbedingungen, Start des Gleichungslösers und Auswertung der Daten vollständig automatisiert. Diese Routine, die in Abbildung 5.6 dargestellt ist, kann als Ausweich-Zielfunktion für datenbasierte Modelle dienen, deren Gültigkeitsbereich verlassen wird. Eine allein auf dieser Methode basierende Optimierung ist jedoch auch möglich.

Abbildung 4.1 stellt die Rechenzeit dar, die bei Einsatz der vorgestellten Methoden erforderlich ist, um den Durchflusskoeffizienten einer bestimmten Geometrie des Typs Durchblicklabyrinth zu berechnen. Für die CFD-Methode wurde eine parallele Berechnung auf 10 Prozessorkernen angenommen. Das Diagramm macht deutlich, dass die Rechenzeiten sowohl der Korrelationen als auch der Regressionsmodelle deutlich unter einer Sekunde pro Evaluation liegen und somit im Optimierungsprozess nahezu vernachlässigbar sind. Die CFD-Methode, inklusive aller notwendigen Schritte, liegt mit ungefähr 300 Sekunden erheblich darüber. Steht eine geringere Anzahl an Prozessoren zur Verfügung, verschlechtert sich dieser Wert weiter.

Bei der Berechnung der Leckage mit der CFD-Methode stellt sich die Frage nach der Genauigkeit des Verfahrens. In Kapitel 5 wird auf die notwendige Auflösung des Rechengebiets eingegangen und ein Vergleich mit hochwertigen experimentellen Ergebnissen angestellt. Dabei zeigt sich, dass die Vorhersagen der CFD-Methode, mit den hier gemachten Einstellungen, eine exzellente Übereinstimmung liefert. Dabei “genügen” sogar relativ simple Methoden, um dieses Ergebnis zu erreichen. Einschränkungen bei der Übertragbarkeit auf andere Zielgrößen müssen jedoch gemacht werden. So ist der Wärmeübergang in Labyrinthdichtungen kein Teil der Untersuchungen gewesen. Die Erfahrungen anderer Autoren (z.B. Weinberger (2014)) legen nahe, dass für dessen Vorhersage mehr Aufwand getrieben werden muss, um ähnlich exakte Resultate zu erzielen - mit entsprechenden Nachteilen für die Optimierung.

4.2 Auswahl geeigneter Optimierungsalgorithmen

In Abschnitt 2.2 wurden drei grundlegende Probleme gradientenbasierter Optimierungsalgorithmen angesprochen. Das Vorgebirgs-, Plateau- und Gratproblem stellen diese Algorithmen vor

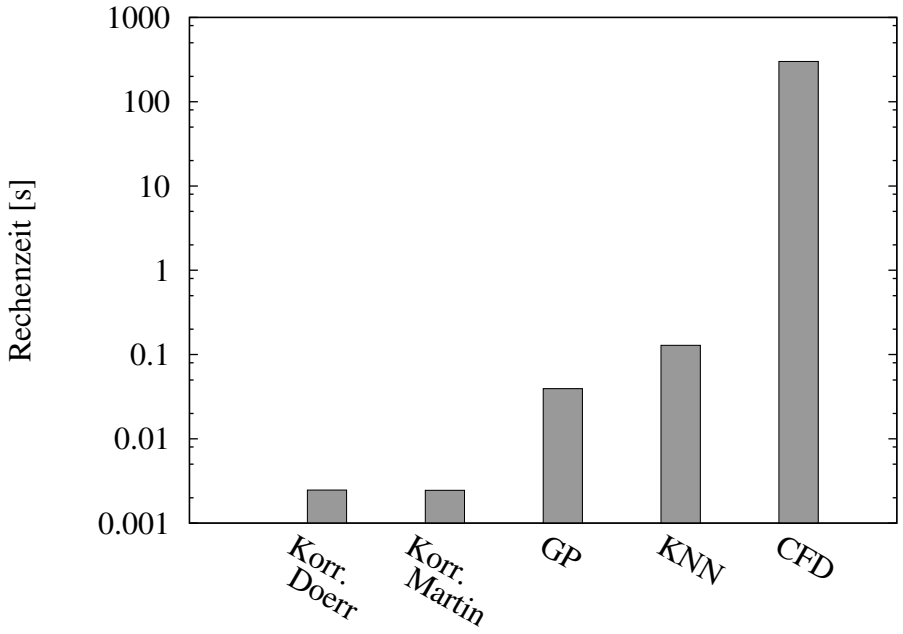

Abb. 4.1: Vergleich der Rechenzeiten unterschiedlicher Berechnungsmethoden

Schwierigkeiten. Diese können zwar durch verschiedene Methoden wie mehrfacher Neustart der Optimierung mit unterschiedlichen Startpositionen teilweise umgangen werden, dennoch überwiegen die Nachteile bei der Suche nach einem optimalen Individuum. Bei der vorliegenden Aufgabenstellung, die sich mit der Optimierung von Labyrinthdichtungen in Turbomaschinen mit mehrjährigen Entwicklungszyklen und sehr großer Einsatzdauer befasst, liegt der Fokus weniger auf der absoluten Geschwindigkeit eines Optimierungsalgorithmus. Viel wichtiger ist die Fähigkeit des Algorithmus, das globale Optimum im Suchraum zu finden.

Da die gradientenbasierten Verfahren nur durch Umwege eine sehr gute Lösung finden, wird hier von Occams Rasiermesser (auch: lex parsimoniae, MacKay (2003)) Gebrauch gemacht und auf Algorithmen ausgewichen, die von sich aus bereits mit hoher Wahrscheinlichkeit das globale Optimum erreichen. Als wichtigster Algorithmus dieser Kategorie wurde daher Differential Evolution (DE) (Storn und Price (1995)) gewählt. Dieser ist zwar relativ aufwendig im Hinblick auf Zielfunktionsevaluationen, sucht den Suchraum aber sehr gründlich ab und ist im Gegensatz zur rein zufälligen Suche oder dem SA-Algorithmus ein elitärer Algorithmus. Dies bedeutet, dass das beste im Verlauf der bisherigen Suche gefundene Individuum immer in der Population erhalten bleibt.

Um die Suche von Differential Evolution beschleunigen zu können, wurde ein Hybrid mit dem Downhill-Simplex-Algorithmus (DS) nach Fröhlig (2004) gewählt. Dieser verwendet zur Beschleunigung der Suche nach einer einstellbaren Anzahl von Generationen im DE-Algorithmus eine einstellbare Anzahl von DS-Iterationen. Weitere Konfigurationsparameter sind die Art der Wahl der zu verwendenden und zu ersetzenden Individuen aus der DE-Population. Der DS-Algorithmus hat zwar vergleichbare Nachteile wie die gradientenbasierten Algorithmen, benötigt jedoch aufgrund seiner geometrischen Vorgehensweise keine zusätzlichen Zielfunktionsevaluationen, die ansonsten zur Berechnung der Gradienten erforderlich sind. Der Hybridalgorithmus

zeichnet sich immer noch durch eine hohe Wahrscheinlichkeit der Konvergenz zum globalen Optimum auch bei komplexen Zielfunktionslandschaften aus, benötigt dafür aber weniger Zielfunktionsaufrufe als der reine Differential-Evolution-Algorithmus.

Als weiterer geeigneter Algorithmus wurde Particle Swarm Optimization (PSO) (Kennedy und Eberhart (1995)) gewählt. Im Rahmen der hier durchgeführten Optimumssuchen zeigte dieser deutlich schnellere Konvergenz zum besten Individuum hin, allerdings besteht bei diesem die Gefahr des zu schnellen Abbrechens der Suche. Die richtige Wahl der Konfigurationsparameter von PSO, angepasst auf das jeweilige Problem, ist dabei von großer Bedeutung. Zu Vergleichszwecken mit in der Vergangenheit am ITS durchgeführten Arbeiten wurde weiterhin Simulated Annealing (Kirkpatrick et al. (1983)) verwendet.

Im Bereich der Mehrzieloptimierung ist derzeit der Non-Dominated Sorting Genetic Algorithm in der zweiten Version (NSGAII) weitverbreitet. Sein größter Vorteil ist, dass die Individuen auf der Paretofront sehr gleichmäßig verteilt werden und dies eine gute Interpolation zwischen den Individuen ermöglicht. Um Vergleiche anstellen zu können, wurde außerdem PAES (Pareto Archived Evolution Strategy) von Knowles und Corne (1999) implementiert. Die Verteilung der Individuen auf der Paretofront ist bei diesem nicht so gleichmäßig ausgeprägt wie bei NSGAII.

5 Lösungsweg

In diesem Kapitel werden die Bausteine des Optimierungsprogramms im einzelnen beschrieben. Zunächst wird die Labyrinthgeometrie parametrisiert, um sie eindeutig beschreiben zu können. Die Werte der hier eingeführten Parameter werden während der Optimierung variiert, um die optimale Geometrie unter den gegebenen Randbedingungen zu finden. In Abschnitt 5.2 wird der Ablauf des Gesamtprogramms gezeigt. Die Zielfunktionsmodule werden schließlich in Abschnitt 5.3 vorgestellt.

5.1 Parametrisches Modell der Labyrinthgeometrie

Abbildung 5.1 stellt vereinfacht einen Ausschnitt aus einer konvergenten Labyrinthdichtung dar. Von oben nach unten ist zunächst das Statorgehäuse abgebildet. Darauf aufgebracht ist ein optionaler Anstreifbelag, der häufig aus Honigwaben oder porösen Metallstrukturen besteht. Im unteren Teil der Abbildung wird der Rotor mit zwei Labyrinthspitzen gezeigt. Die Strömungsrichtung von links nach rechts wird durch die Pfeile am Einlass verdeutlicht.

Die Parameter, die die Geometrie der Dichtung definieren, sind als Bemaßungen dargestellt. Die Geometrie der Labyrinthspitze wird durch die Spitzenhöhe S_H, die Spitzenbreite S_B, den Spitzenneigungswinkel γ sowie den Flankenwinkel θ definiert. Verschleißeinflüsse an der Spitze werden über den Radius R an der luvseitigen Kante der Spitze dargestellt. Zur Vereinfachung der Geometrie befinden sich die beiden Fußkanten der Spitze auf demselben Radius.

Die Geometrie der Kammer definiert sich zunächst über die Teilung t und die Spaltweite s. Ein Durchblicklabyrinth ist damit eindeutig beschrieben. Bei einem Stufenlabyrinth kommen die Stufenhöhe ST_H und der Stufenversatz ST_S hinzu. Ein divergentes Stufenlabyrinth besitzt eine positive (der Radius wird in Strömungsrichtung größer), ein konvergentes Labyrinth eine negative Stufenhöhe (der Radius verkleinert sich in Strömungsrichtung). Für die Optimierung wird später meist der relative Stufenversatz ST_S/t verwendet. Die Kammergeometrie wird durch ein Verschleißbild in Form von Anstreifnuten im Statormaterial verändert. Diese Nuten werden hier mit einer rechteckigen Form angenähert und mit den Parametern Nuthöhe N_H, Nutweite N_W und Nutversatz N_S relativ zur Mitte der gegenüberliegenden Spitze beschrieben. Der Parameter R_i bezeichnet den Rotorradius am Fuß der ersten Labyrinthspitze.

5.2 Optimierungsablauf

Bevor die Funktionsweise der Zielfunktionsmodule beschrieben wird, soll hier auf das Zusammenspiel der einzelnen Programmbausteine eingegangen werden. Einen Überblick liefert Abbildung 5.2. Da das Programm nicht allein auf die Optimierung von Labyrinthdichtungen festgelegt ist, und je nach Typ und Parametrisierung von Labyrinthdichtung auch unterschiedliche Parameteranzahlen auftreten können, muss diese Zahl zunächst festgelegt werden. Mit der anschließenden Eingabe der oberen und unteren Schranke jedes Parameters werden die ersten Randbedingungen

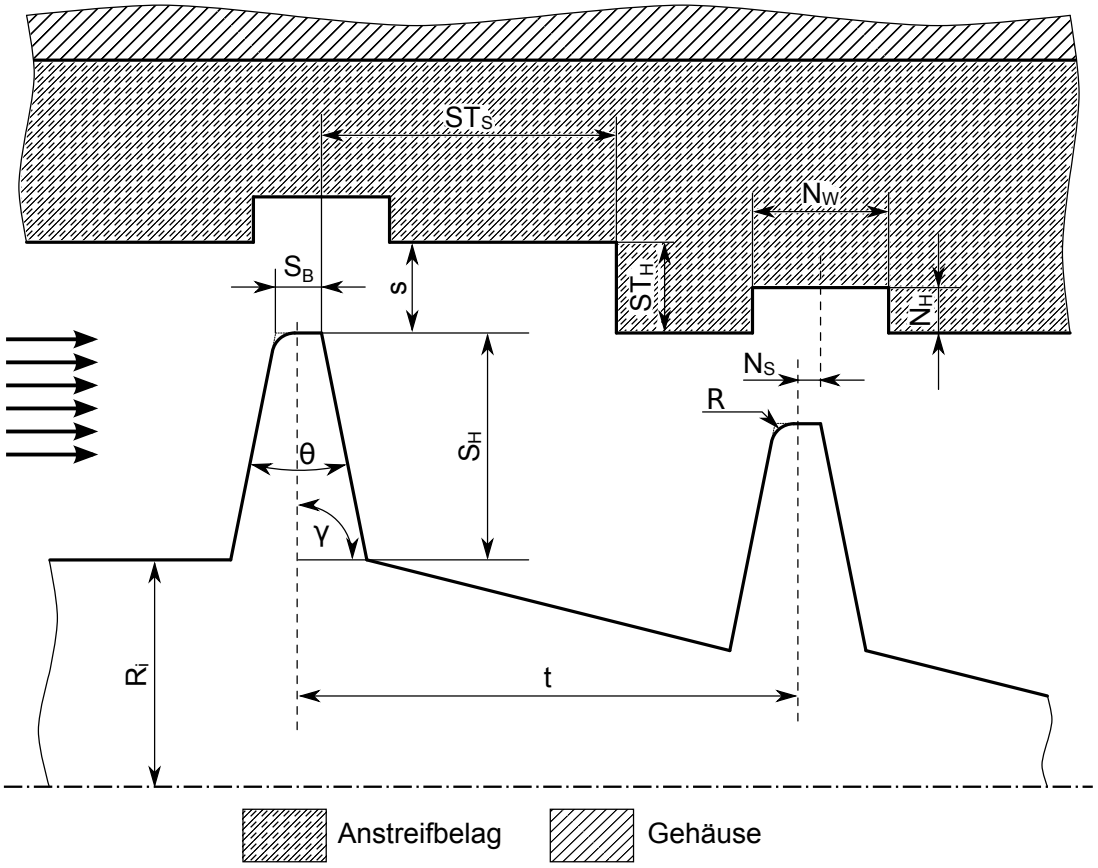

Abb. 5.1: Parameter zur Beschreibung der Dichtungsgeometrie

gesetzt. Zusätzlich können weitere Einschränkungen des Parameterraums durch die Definition von Parameterkombinationen vorgenommen werden. Dies kann zum Beispiel die Gesamtlänge einer Labyrinthdichtung in Form von $(n-1) \cdot t + S_B$ sein. Somit ist die exakte Beschreibung des zur Verfügung stehenden Bauraums möglich. Vor dem Start der Optimierung erfolgt weiterhin die Auswahl einer oder, im Falle der Mehrzieloptimierung, mehrerer Zielfunktionen und des Optimierungsalgorithmus.

Die einzelnen Parameterschranken werden dem gewählten Optimierungsalgorithmus übergeben, da deren Einhaltung direkt durch diesen überwacht wird. Je nach Algorithmus kann dies mit unterschiedlichen Methoden geschehen. Die Überwachung der Schranken von Parameterkombinationen wie oben beschrieben geschieht durch eine in das Zielfunktionsmodul eingebaute Funktion.

Abhängig von der Art des Optimierungsalgorithmus werden die Parameter auf unterschiedliche Art und Weise kombiniert, um schließlich das bestmögliche Individuum beziehungsweise die Pareto-optimale Menge zu finden. Dabei wird das Zielfunktionsmodul wiederholt aufgerufen, und die Güte des Individuums berechnet. Es kann jedoch geschehen, dass der Optimierungsalgorithmus dabei ein Individuum vorschlägt, dessen Güte von der Zielfunktion nicht hinreichend genau berechnet werden kann. Im Falle von Korrelationen und Modellen wie KNN oder Gaußschen Prozessen kann ein Individuum zum Beispiel außerhalb des Vertrauensbereichs liegen (vgl. Kapitel 4). In diesem Fall wird als Ersatzmodell die CFD-Methode gestartet, um ein Ergebnis

für dieses Individuum zu erhalten. Ist dies erfolgreich, die CFD-Rechnung also konvergiert und das Ergebnis plausibel, wird dieser Wert an den Optimierer zurückgegeben und zusätzlich im Archiv gespeichert. Das Archiv fließt bei der nächsten Modellerzeugung mit in die Lerndaten ein, um den Gültigkeitsbereich der neuen Modelle zu erweitern. Sollte die CFD-Rechnung nicht konvergieren oder Fehler im Ablauf auftreten, wird das Ergebnis verworfen und ein "Strafwert" also eine sehr geringe Güte zurückgegeben, damit das fehlerhafte Ergebnis die Optimumssuche nicht negativ beeinflusst. Die automatische Einbindung einer Ersatzfunktion ist ein neuer Ansatz, der so in der Literatur noch nicht zu finden ist.

Die Optimierungsschleife wird so lange ausgeführt, bis ein Abbruchkriterium erreicht wird. Ein Abbruchkriterium ist das Erreichen des Konvergenzkriteriums des Optimierers - oft definiert über eine gewisse Anzahl an Iterationen, über die keine weitere Verbesserung erzielt wird. Als weiteres Abbruchkriterium ist die maximale Anzahl von Zielfunktionsaufrufen konfigurierbar. Bei Erreichen eines Abbruchkriteriums wird die Optimierungsschleife gestoppt und die Parameterkombination sowie der oder die Gütewerte des besten Individuums an den Benutzer zurückgegeben. Der Quelltext der Optimierungsumgebung ist in Anhang A.1 zu finden.

5.3 Zielfunktionsmodule

Anhand eines generischen Zielfunktionsmoduls „TF“ soll kurz der Ablauf eines solchen Moduls vorgestellt werden. Wird dem Zielfunktionsmodul ein Individuum übergeben, wird zunächst geprüft, ob das Individuum alle Schranken einhält oder eine Verletzung vorliegt. Um beim Beispiel aus dem vorherigen Abschnitt zu bleiben, werden die entsprechenden Parameterwerte in die Gleichung für die Labyrinthlänge eingesetzt und geprüft, ob das Ergebnis innerhalb der definierten Schranken liegt. Ist das nicht der Fall, wird ein hoher (im Falle der Minimumssuche) "Strafwert" anstatt des berechneten wahren Zielfunktionswerts zurückgegeben, um den Optimierungsalgorithmus dazu zu bewegen, diesen Bereich zu meiden.

Besteht das Individuum die Schrankenprüfung, kann optional zunächst ein Archiv durchsucht werden, das bereits berechnete Zielfunktionswerte (Güten) aus vorherigen Berechnungen enthält. Dabei wird innerhalb einstellbarer Grenzen nach einem ähnlichen Individuum gesucht. Ist die Suche erfolgreich, wird das archivierte Ergebnis verwendet und so eine aufwendige Neuberechnung vermieden. Diese Vorgehensweise ist besonders empfehlenswert bei rechenintensiven Zielfunktionen wie der Strömungssimulation. Wird kein ähnliches Individuum gefunden, so erfolgt nun der Aufruf der eigentlichen Güteberechnung. Diese Funktion kann beliebige Berechnungsmethoden enthalten. Die einzige Bedingung aus der Sicht des Programms ist, dass die Funktion ein Individuum in Form einer Parameterliste akzeptiert und einen Gütewert zurückliefert.

Nach Abschluss der Berechnung kann optional der Abstand zu einem beliebigen Zielwert berechnet und dieser an Stelle des eigentlichen Zielfunktionswertes an den Optimierer zurückgegeben werden. Diese Vorgehensweise wird gewählt, wenn nicht das globale Optimum gesucht wird, sondern auf einen Zielwert hin optimiert werden soll. Dies kann zum Beispiel bei Labyrinthdichtungen der Fall sein, wenn die Dichtung zur Einstellung eines Kühlluftmassenstroms verwendet werden soll - im Gegensatz zur Optimierung auf eine möglichst gute Dichtwirkung. Der Quelltext

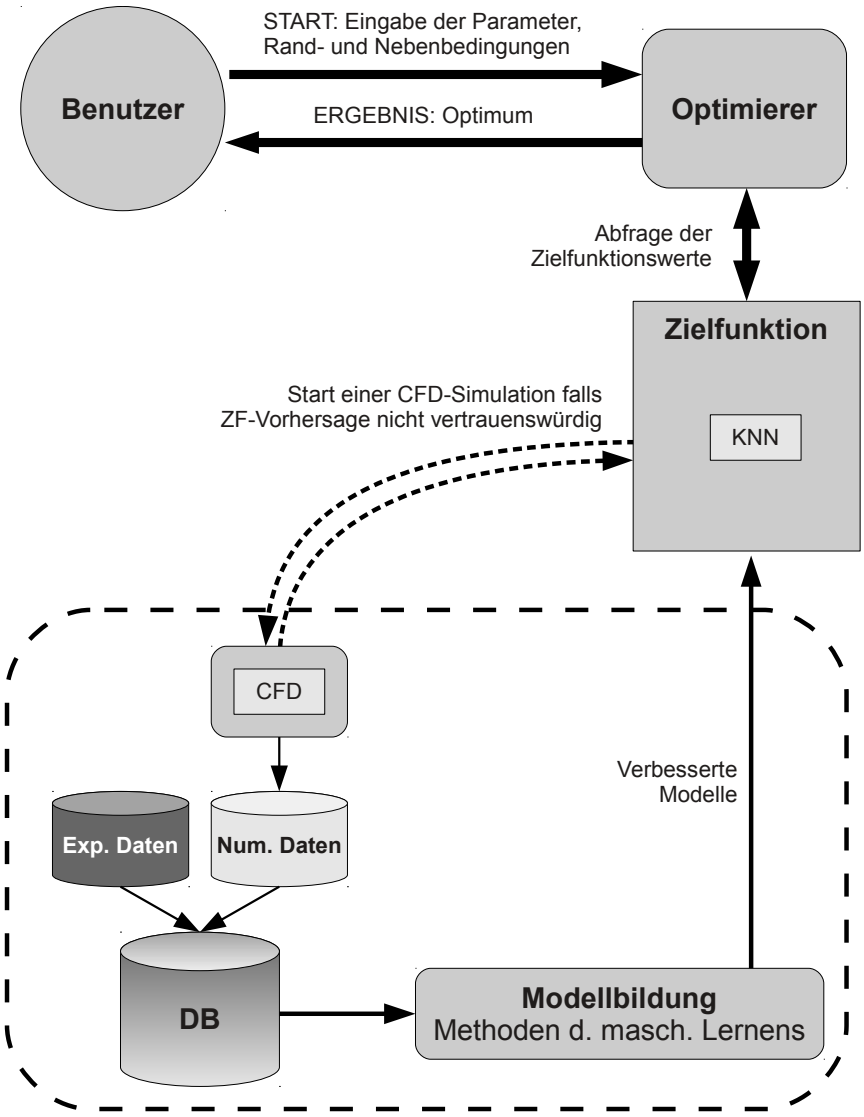

Abb. 5.2: Programmablauf bei Verwendung einer Zielfunktion mit CFD-Ausweichmethode

der implementierten Zielfunktionen ist in Anhang A.2 zu finden. Einen Überblick über den Ablauf des Programms im Allgemeinen und eines Zielfunktionsmoduls bieten Abbildung 5.2 und 5.3.

5.3.1 Künstliche Neuronale Netze

Die Erzeugung, das Training und das Auslesen Künstlicher Neuronaler Netze (KNN) wurde mit der *FANN*-Bibliothek realisiert (Nissen et al. (2014)). Diese in C geschriebene OpenSource-Bibliothek bietet Möglichkeiten zum Arbeiten mit großen Datenmengen und ist sehr schnell (vgl. auch Abbildung 4.1). Mehrere Trainingsmethoden für verschiedenen Arten von KNN stehen zur Verfügung, wobei hier ausschließlich Multi-Layer-Perceptron- (MLP-) Netzwerke zum Einsatz kamen. Ein Beispiel dieser KNN wurde in Abb. 2.15 gezeigt. Die Anbindung an die im Rahmen dieser Arbeit entstandene Optimierungsumgebung wurde mit Schnittstellen zur

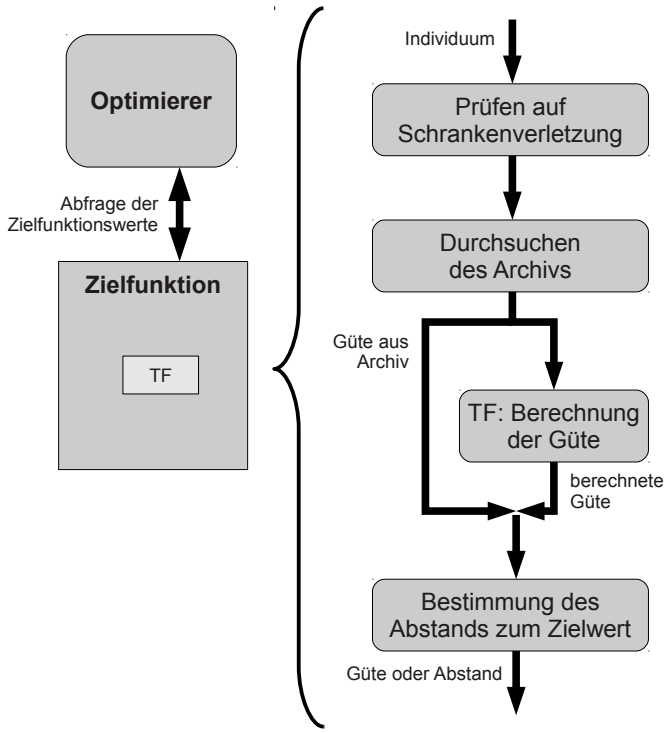

Abb. 5.3: Schema eines generischen Zielfunktionsmoduls

Python-Programmiersprache umgesetzt, die in der verwendeten Bibliothek enthalten sind.

Die KNN wurden auf Basis von vorher erstellten Trainingsdatensätzen angelernt. Dabei wurden für das Training jeweils nur die Parameter verwendet, die bei der Erstellung des zugehörigen Datensatzes variiert wurden. Bei den Durchblicklabyrinthen wurden beispielsweise das Druckverältnis, die Spaltweite, die Teilung, die Spitzenbreite und die Spitzenhöhe variiert. Bei der Verwendung in der Optimierung dürfen nur diese Parameter als variabel deklariert werden. Da sich die Datensätze und somit die KNN für Durchblick- und Stufenlabyrinthe unterscheiden, wurden mehrere Zielfunktionsmodule, jeweils eines für jeden Datensatz, implementiert. Die vom Optimierer übergebenen Parameter werden gefiltert und auf die für das Auslesen des jeweiligen KNN notwendigen Größen reduziert. Anschließend wird das KNN geladen und die Ausgangsgröße (z.B. C_D-Wert), die sich für die übergebenen Eingangsgrößen ergibt, ausgelesen. Diese wird gemäß dem in Kapitel 5.3 beschriebenen Ablauf an den Optimierer zurückgegeben.

Um zu vermeiden, dass das KNN extrapolieren muss, wobei die Vorhersagequalität leidet, ist der Parameterbereich der Trainingsdaten in der Zielfunktion hinterlegt. Bevor die Güte eines Individuums mit dem KNN evaluiert wird, wird zuerst geprüft, ob alle Parameter im Gültigkeitsbereich der Trainingsdaten liegen. Ist dies der Fall, wird das KNN wie oben beschrieben, ausgelesen. Andernfalls wird die CFD-Zielfunktion als aufwendiges aber präzises Ersatzmodell gestartet.

5.3.2 Gaußsche Prozesse

Für die Verwendung von Gaußschen Prozessen als Zielfunktion wurde das Pythonpaket *Scikit-Learn* eingebunden (Pedregosa et al. (2011)). Diese komplett in Python geschriebene OpenSource-

Bibliothek enthält viele Methoden zum maschinellen Lernen wie unter anderen Regressionsmodelle, Entscheidungsbäume und Gaußsche Prozesse. Hinzu kommen Werkzeuge zum Training und zur Validierung der erzeugten Modelle.

Um Gaußsche Prozesse als Zielfunktion verwenden zu können, müssen diese, wie auch die KNN, zunächst auf Basis von Trainingsdaten im Voraus erzeugt werden. Die Zielfunktion bedient sich anschließend dieser erzeugten Modelle, um Vorhersagen zu treffen. Die Struktur des Zielfunktionsmoduls ist dabei fast identisch mit dem, das als Modell die KNN verwendet. Der Unterschied liegt in der Art und Weise, wie entschieden wird, ob die eigentliche Zielfunktion oder die CFD-Ersatzfunktion aufgerufen wird. Die Gaußschen Prozesse liefern nach der Abfrage, wie in Kapitel 2.3 beschrieben, einen Unsicherheitswert für die Vorhersage mit. Ist dieser Wert zu hoch, wird das CFD-Ersatzmodell verwendet. Eine gewisse Extrapolation über die eigentlichen Trainingsdaten hinaus ist somit möglich.

5.3.3 Numerische Strömungssimulation

Für die Berechnung eines Lerndatensatzes zum Training von Künstlichen Neuronalen Netzen und Gaußschen Prozessen (siehe Abschnitt 5.3.1) sowie als universelle Zielfunktion für die Optimierung (siehe Abschnitt 5.3.3) wird eine flexible, robuste und möglichst präzise Methode zur numerischen Strömungssimulation benötigt. Es wurde die OpenSource-Software OpenFOAM® in Version 2.1.1 verwendet (OpenCFD (2012)).

Als Gleichungslöser wurde *rhoPorousMRFSimpleFoam* verwendet. Dabei handelt es sich um einen Gleichungslöser, der die kompressiblen, zeitinvarianten Navier-Stokes-Gleichungen löst. Die Turbulenz wird dabei mit klassischen Modellen wie k,ϵ-, k,ω-, Spalart-Allmaras- und einigen weiteren modelliert. In dieser Arbeit kam das k,ω-SST-Modell zum Einsatz, das sich für einen relativ großen y^+-Bereich eignet (Menter (1993)) und für ein stabiles Konvergenzverhalten der Simulationen sorgen konnte. Die anderen Turbulenzmodelle zeigten in Vorversuchen bei bestenfalls vergleichbarer Genauigkeit schlechtere Konvergenzeigenschaften und wurden nicht weiter verwendet.

Die SIMPLE-Druckkorrektur (Patankar und Spalding (1972)) kommt in dieser Arbeit zum Einsatz. Für den Druck wurde ein geometrisch-algebraischer Mehrgitterlöser (*GAMG*) verwendet, für die Enthalpie ein vorkonditionierter bikonjugierter Gradientenlöser (*PBiCG*) und für die Geschwindigkeit sowie die Turbulenzgrößen ein Löser mit Glättungsfunktion (*smoothSolver*). Es wurden alle Gleichungen bis zu einem Residuum von 10^{-5} gelöst, außer der Enthalpiegleichung, die nur bis 10^{-4} gelöst wurde. Als Fluid wurde Luft als perfektes Gas betrachtet. Die Viskosität des Gasgemisches wurde nach Sutherland (1895) berechnet. Am Eintritt wurden Totaldruck, statische Temperatur und die Turbulenzgrößen, einem Turbulenzgrad von 10% entsprechend, vorgegeben. Am Austritt wurde der statische Druck fixiert; Rotor und Stator wurden als adiabatische, glatte Wände mit Haftbedingung simuliert. Die Länge des Einlaufs betrug für alle Simulationen drei- bis viermal den Wert der Spitzenhöhe, der Auslauf das zehn bis 15-fache ebendieser, um eine Rückströmung durch den Austrittsquerschnitt, die sich durch den Wirbel hinter der letzten Spitzen einstellen könnte, sicher zu vermeiden. Dies wurde ebenfalls in Vorversuchen, in denen die Länge des Auslaufs bis zur 30-fachen Spitzenhöhe vergrößert wurde, nachgewiesen.

Die Diskretisierung des zweidimensionalen Rechengebiets wurde mit Gmsh (Geuzaine und Remacle (2009)) durchgeführt. Die Zellen des Netzes können mit Gmsh durch unterschiedliche Algorithmen erstellt werden. Hier kam der Frontal-Algorithmus von Remacle et al. (2013) zum Einsatz, der sehr gleichmäßige Netze erzeugt, die sehr geringe Werte für den Nicht-Orthogonalitätsindex aufweisen. Wichtigste Größe für die Definition der „Netzfeinheit“ stellt die charakteristische Kantenlänge der drei- oder viereckigen 2D-Zellen dar. Eine ausführliche Gitterunabhängigkeitsstudie führte zu dem Ergebnis, dass die Auflösung im Spalt über den Labyrinthspitzen das für die Genauigkeit entscheidende Maß ist. Um die Ablösung über der Spitze sicher aufzulösen, sollten mindestens fünf Zellen im Wirbel liegen. Somit sind je nach Spaltweite mindestens 25 Zellen in radialer Richtung über den Spalt ausreichend. In Abbildung 5.4 ist die Annäherung der errechneten C_D-Werte für Druckverhältnisse von $\pi = 1{,}3$ und $\pi = 1{,}5$ an die von Denecke (2007) experimentell ermittelten Werte für ein konvergentes Stufenlabyrinth mit vier Dichtspitzen dargestellt. Denecke gibt für die experimentellen Ergebnisse eine Messunsicherheit von $\pm 8\%$ an, welche im Diagramm durch gestrichelte Linien wiedergegeben wird. Die mit OpenFOAM berechneten Werte liegen bereits ab 15 Zellen über die Spalthöhe deutlich innerhalb der Messunsicherheit und ändern sich ab 25 Zellen nicht mehr. Die Numerik gibt die experimentellen Ergebnisse somit exzellent wieder.

In der Arbeit von Denecke (2007) wurde im verwendeten numerischen Verfahren eine starke lokale Verfeinerung des Rechengitters in der Scherzone über dem Spitzenwirbel durchgeführt. Um die dabei entstehenden hohen Gradienten in der Netzauflösung und dadurch auftretendes instabiles numerisches Verhalten zu vermeiden, wurde in der vorliegenden Arbeit darauf verzichtet und der Spaltbereich bereits im Vorfeld über eine größere Zone hinweg fein aufgelöst. Am Eintritts- und Austrittsquerschnitt kann das Netz gröber gestaltet werden. Hier genügt das Zehnfache der charakteristischen Kantenlänge im Spalt. Am Kammerboden wird die Netzfeinheit um den Faktor zwei bis vier vergröbert. Für eine Labyrinthgeometrie mit einer Spitzenhöhe von $2{,}5\ mm$ und einer Teilung von $8\ mm$, wie in Abbildung 5.5 gezeigt, ergibt sich somit eine Zellenzahl von rund 100.000 Dreieckszellen.

Die Geometrie des Rechengebietes, anhand derer das Gitter erzeugt wird, wurde mittels eines selbst entwickelten Pythonprogramms *pyGeogen* erstellt. Dieses verwendet die Parameter des Labyrinthmodells aus Abschnitt 5.1 als Eingangsgrößen. Die erzeugte Geometrie wird anschließend auf Plausibilität überprüft, damit Überschneidungen von Rotor und Stator aufgrund von Fehleingaben abgefangen werden können. Wert gelegt wurde bei der Programmierung auf einen modularen Aufbau, der es erlaubt, das Programm mit geringem Aufwand an andere Aufgaben anzupassen.

Die Steuerung und Überwachung des Ablaufs einer oder mehrerer Rechnungen von der Parametereingabe bis zur Ausgabe der Ergebnisse übernimmt ein weiteres selbstentwickeltes Pythonprogramm *labyBatch*, das sich Teile der OpenSource-Bibliothek *pyFoam* zunutze macht. Es handelt sich dabei um eine Stapelverarbeitung, die zunächst eine Liste vom Benutzer vorgegebener Parameter für eine oder mehrere Labyrinthgeometrien einliest. Auf Basis eines vorher eingerichteten Ordners, der sämtliche Daten enthält, die von OpenFOAM benötigt werden, wird eine Ordnerstruktur erstellt, in der jede Geometrie einen eigenen Unterordner erhält. Anschließend werden die Rechengitter generiert, geprüft und in ein von OpenFOAM lesbares Format umgewandelt.

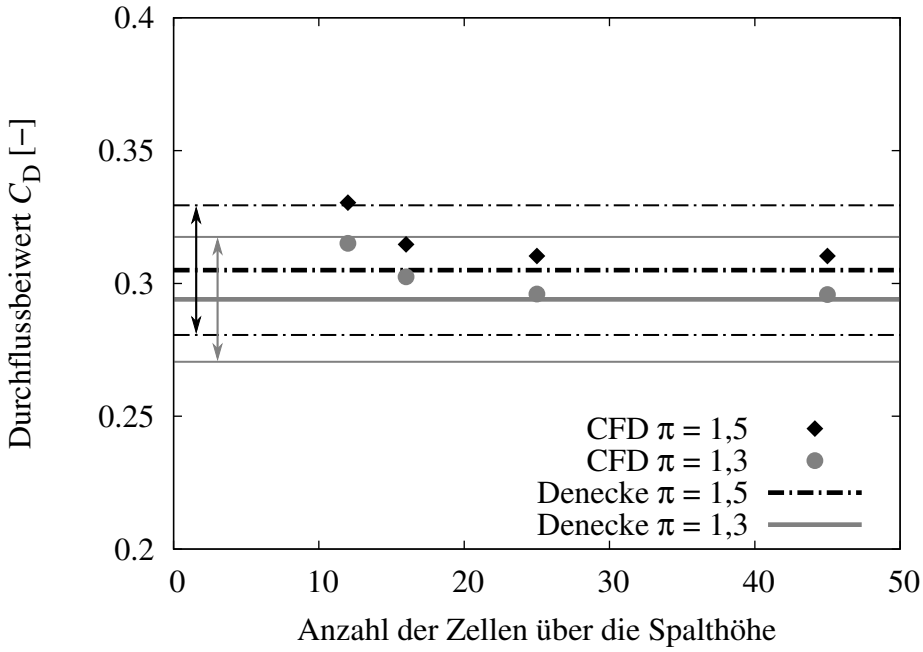

Abb. 5.4: Verlauf des Durchflussbeiwerts über der Zellenzahl bei der Netzunabhängigkeitsstudie

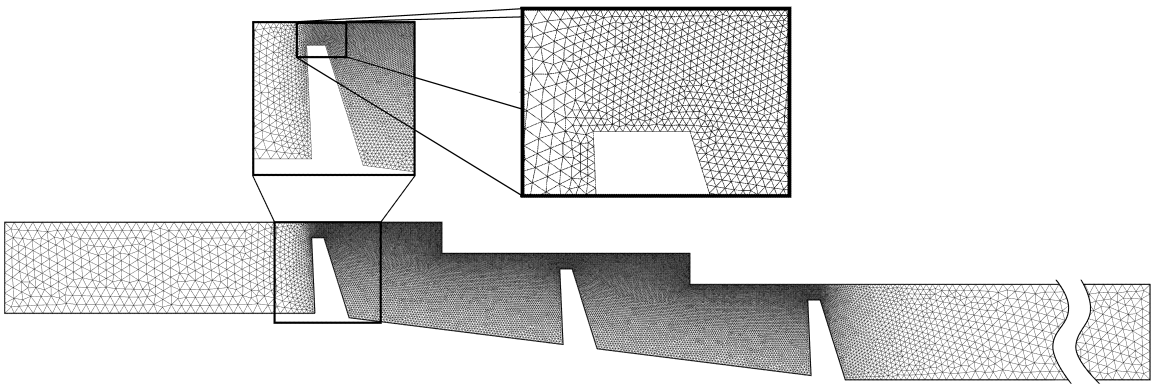

Abb. 5.5: Diskretisierung des Strömungsgebietes mit hoher Auflösung in kritischen Bereichen

Nun werden die Simulationen für die vorgegebenen Variationen der Randbedingungen (z.B. Druckverhältnis π) für jede Geometrie von OpenFOAM durchgeführt und die Ergebnisse sowohl im jeweiligen Unterordner als auch in einer gemeinsamen Datei für alle Variationen abgelegt. Konvergiert eine Rechnung nicht, wird dies ebenfalls in diesen Dateien vermerkt und mit der nächsten Rechnung fortgefahren. Soll die Stapelverarbeitung als Zielfunktion für die Optimierung verwendet werden, vereinfacht sich der Ablauf auf eine einzige Rechnung. In Abbildung 5.6 ist dieser dargestellt.

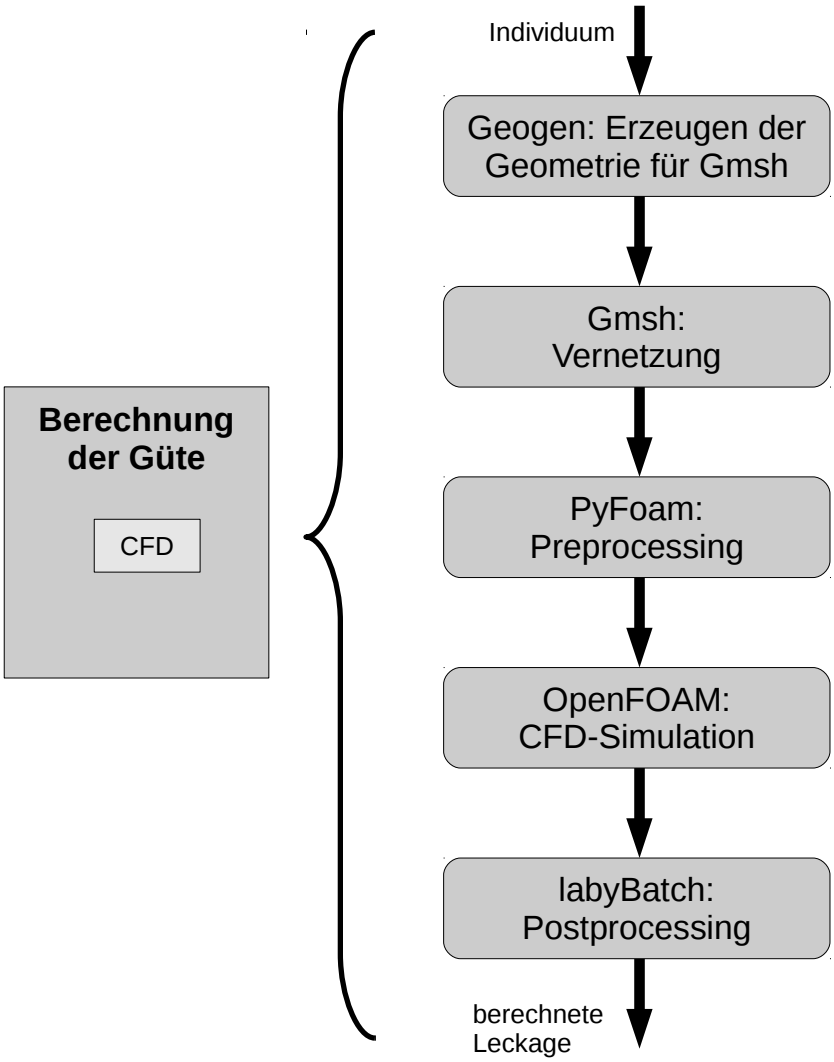

Abb. 5.6: Ablaufdiagramm des CFD-Automatismus

6 Optimierungsergebnisse

In diesem Kapitel werden drei Aufgabenbeispiele vorgestellt, anhand derer letztendlich die Wahl einer geeigneten Methode zur Lösung der im Rahmen dieser Arbeit gestellten Aufgabe ermöglicht wird. Die drei gewählten Optimierungsaufgaben sind folgende:

- Entwurf eines optimalen Durchblicklabyrinths unter Verwendung unterschiedlicher Zielfunktionsmodule
- Entwurf eines optimalen Stufenlabyrinths im Neuzustand
- Entwurf eines gegen Verschleiß robusten Stufenlabyrinths

Die erste Aufgabe dient der Demonstration und dem Test der Abläufe, die zur Durchführung einer Optimierung notwendig sind. Verschiedene Zielfunktionsmodule werden verwendet, um einen Vergleich zwischen den einzelnen Ansätzen anstellen zu können. Die zweite Aufgabe wurde gewählt, um die Komplexität der Optimierung gegenüber dem ersten Beispiel zu erhöhen. Die Schwierigkeit liegt hier in der erhöhten Anzahl variabler Parameter und des steigenden Rechenaufwands. Zunächst handelt es sich jedoch wie im ersten Beispiel um einen Versuch mit monokriteriellem Optimierungsalgorithmus. Als dritte Aufgabe soll die Eignung der multikriteriellen Methodik für die Problematik der verschleißenden Labyrinthdichtung aufgezeigt werden. Die Herausforderungen bestehen für diese sehr rechenintensive Aufgabe unter anderem in der exakten Definition der Randbedingungen und Beschränkungen, innerhalb derer sich die Optimumssuche bewegen muss.

6.1 Optimales Durchblicklabyrinth

Als erste Aufgabe soll die beste Geometrie in Bezug auf Leckageverluste für ein einfaches Durchblicklabyrinth mit drei Dichtspitzen gesucht werden. Diese dient der eingehenden Validierung der eingesetzten Methoden als erstem Schritt hin zu komplexeren Anwendungen. Als Modell für die Leckagevorhersage sollen Künstliche Neuronale Netze im Verbund mit der CFD-Ausweichmethode verwendet werden. Das Training der KNN wird auf Basis eines Lerndatensatzes durchgeführt, der vorher erstellt werden muss.

6.1.1 Erzeugung eines homogenen Datensatzes

Um einen Vergleich zwischen der “direkten” Leckageberechnung von Labyrinthdichtungen mittels CFD und der Approximation mittels künstlicher neuronaler Netze durchführen zu können, ist zunächst ein Lerndatensatz erforderlich, mit dem die KNN trainiert werden. Diese Lerndaten wurden mit der automatischen CFD-Routine erzeugt. Es wurden sechs Parameter in jeweils drei Schritten gleicher Schrittweite nach Tabelle 6.1 variiert. Die Spitzengeometrie wurde scharfkantig,

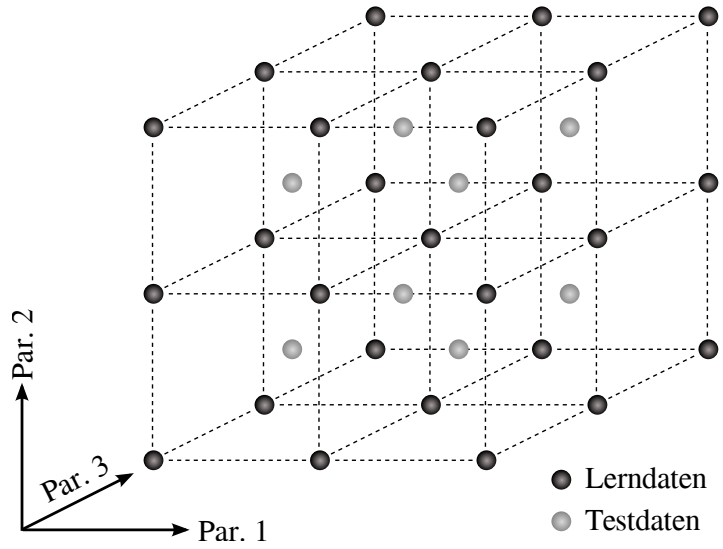

Abb. 6.1: Lern- und Testdatenverteilung im Parameterraum

Tab. 6.1: Parametervariation der KNN-Lerndaten für das Durchblicklabyrinth

Parameter	Symbol	Wert	Schrittweite
Spitzenzahl	n	3	
Spaltweite	s	$0{,}3 \ldots 0{,}9\,mm$	$0{,}3\,mm$
Spitzenbreite	S_B	$0{,}2 \ldots 0{,}8\,mm$	$0{,}3\,mm$
Spitzenhöhe	S_H	$2{,}0 \ldots 6{,}0\,mm$	$2{,}0\,mm$
Teilung	t	$4{,}0 \ldots 10{,}0\,mm$	$3{,}0\,mm$
Spitzenneigung	γ	$90{,}0° \ldots 110{,}0°$	$10{,}0°$
Flankenwinkel	θ	$15{,}0°$	
Druckverhältnis	π	$1{,}5 \ldots 1{,}9$	$0{,}2$

ohne Radius, und der Statorteil als glatte Gegenfläche erzeugt. Die Geometrien entsprechen somit einer Labyrinthdichtung im Neuzustand.

Zusätzlich zu den 729 Datentupeln, die sich aus den Kombinationen der Parameter ergeben, wurden weitere 36 Konfigurationen berechnet, die als Testdaten zur Erkennung von Überanpassung genutzt wurden. Die Testdaten liegen mittig auf den Raumdiagonalen zwischen den Lerndaten. Der Vorteil dieser Vorgehensweise gegenüber der mehrfachen Kreuzvalidierung liegt darin, dass hier alle Lerndaten zum Training zur Verfügung stehen und kein Teil der Daten zurückgehalten werden muss.

6.1.2 Vorbereitung der Optimumssuche

In dieser Optimierungsaufgabe kommen zwei Modelle zur Leckagevorhersage zum Einsatz. Die Vor- und Nachteile von Künstlichen Neuronalen Netzwerken und Gaußschen Prozessen werden unter möglichst realistischen Bedingungen untersucht und die beiden Methoden gegenübergestellt.

Für die Vorhersage der Durchflussbeiwerte in Abhängigkeit der Eingangsparameter aus Tabelle 6.1 wurde eine Methode gewählt, die sich in Vorversuchen als bester Kompromiss zwischen Trainingsaufwand und zuverlässig guter Vorhersagegenauigkeit herausgestellt hatte. Dazu wurden jeweils 100 MLP-Netzwerke mit einer verdeckten Schicht und elf Neuronen trainiert (vgl. Abbildung 2.15). Aus diesen KNN wurden die zehn Kandidaten ausgewählt, die den geringsten Fehler nach der Methode der kleinsten Fehlerquadrate in Bezug auf die Testdaten aufweisen. Für die Vorhersage wurden alle zehn KNN abgefragt und aus den Vorhersagen der einzelnen KNN der Mittelwert gebildet. Die benötigte Zeit für das Training der KNN betrug etwa 15 Minuten für die genannten 729 Datentupel und 100 KNN.

Als zweites Modell für die Leckagevorhersage wurde ein Gaußscher Prozess (GP) erstellt. Dabei wurden dieselben Lerndaten verwendet, die auch für das Training der KNN herangezogen wurden. Die Erstellung des GP war ungleich aufwendiger und benötigte auf demselben PC circa zwei Stunden. Als Maximalwert für die Unsicherheit der Vorhersage wurde die Breite des Konfidenzintervalls auf $\sigma_{max} = 0{,}01$ eingestellt, welche während der Optimierung jedoch nie überschritten wurde.

Die Schranken, innerhalb derer sich der Suchalgorithmus bewegen darf, wurden mit den Grenzen des Lerndatensatzes gleichgesetzt. Die Suche wurde mit jedem Algorithmus zehnmal durchgeführt, um einen repräsentativen Mittelwert bilden zu können. Da die Zahl der Zielfunktionsaufrufe teilweise stark schwankt, ist in den Ergebnissen zusätzlich die Standardabweichung angegeben.

Die Suche nach der besten Geometrie wurde mit unterschiedlichen Optimierungsalgorithmen durchgeführt. Der Particle Swarm Optimizer (PSO) wurde verwendet, da er eine schnelle Suche mit sehr gutem Ergebnis bei gleichzeitig relativ geringer Gefahr des frühzeitigen Abbruchs bietet. Als Kandidat für eine gründliche Suche mit dem Nachteil deutlich höherer Zielfunktionsaufrufe wurde Differential Evolution eingesetzt. Als Maß für die Güte einer Labyrinthdichtung wurde bei allen hier vogestellten Beispielen der Durchflussbeiwert C_D gewählt.

6.1.3 Ergebnisse der Optimumssuche

Die ersten Suchläufe wurden mit KNN als Zielfunktion und PSO als Optimierungsalgorithmus durchgeführt. Die Suche benötigte im Mittel 444 Zielfunktionsaufrufe mit einer Standardabweichung von 123,9 ZF-Aufrufen. Das gefundene Optimum, das in jedem Suchlauf getroffen wurde, hat die Parameterwerte $S_H = 6\,mm$, $S_B = 0{,}2\,mm$, $s = 0{,}3\,mm$, $t = 10\,mm$, $\gamma = 110°$ und $\pi = 1{,}5$. Der Durchflussbeiwert bei diesen Parameterwerten beträgt nach der Vorhersage des KNN-Ensembles $C_D = 0{,}4905$. Dieselbe Suche mit dem Differential-Evolution- (DE-) Algorithmus durchgeführt, benötigt deutlich mehr Zielfunktionsaufrufe, nämlich im Mittel 1856 mit einer Standardabweichung von 561,8 ZF-Aufrufen. Die gefundene Optimalgeometrie war mit der von PSO gefundenen identisch. Zur Übersicht sind die Ergebnisse in Tabelle 6.2 dargestellt.

Tab. 6.2: Parameterbereich und optimale Geometrie des Durchblicklabyrinths

Parameter	Symbol	Schranken	Ergebnis
Spaltweite	s	$0{,}3 \ldots 0{,}9\,mm$	$0{,}3\,mm$
Spitzenbreite	S_B	$0{,}2 \ldots 0{,}8\,mm$	$0{,}2\,mm$
Spitzenhöhe	S_H	$2{,}0 \ldots 6{,}0\,mm$	$6{,}0\,mm$
Teilung	t	$4{,}0 \ldots 10{,}0\,mm$	$10{,}0\,mm$
Spitzenneigung	γ	$90{,}0° \ldots 110{,}0°$	$110{,}0°$
Druckverhältnis	π	$1{,}5 \ldots 1{,}9$	$1{,}5$
Bester C_D-Wert (KNN)			$0{,}4905$
Bester C_D-Wert (GP)			$0{,}4753$
Bester C_D-Wert (CFD)			$0{,}4999$

Gemäß der Vorhersage des KNN ist die optimale Labyrinthgeometrie eines Durchblicklabyrinthes mit glattem Stator innerhalb des in Tabelle 6.1 angegebenen Parameterraums eine Geometrie mit möglichst schmalen Spitzen, möglichst kleiner Spaltweite, größter Spitzenhöhe und Teilung, maximal entgegen der Strömungsrichtung geneigten Spitzen und einem möglichst kleinen Druckverhältnis. Eine geringe Spitzenbreite begünstigt die Ablösung und somit die Einschnürung der Strömung über der Spitze. Eine Verkleinerung der Spaltweite sorgt bei einem Durchblicklabyrinth für einen geringeren C_D-Wert und einen geringeren Leckagemassenstrom. Dieser Sachverhalt ist bei anderen Labyrinthbauarten nicht immer gegeben. Komotori (1957), Ueda und Kubo (1967) und Shimoyama und Yamada (1957) berichten über ein optimales Verhältnis von Teilung zu Spitzenhöhe $t/S_H = 4 \ldots 5$. Dieses Verhältnis führt im vorgegebenen Parameterraum durch die Beschränkung der Teilung auf $t < 10\,mm$ zu sehr niedrigen Spitzen. Die Verwirbelung in der Kammer ist dann offenbar nicht optimal, sodass in dem hier betrachteten Fall eine große Spitzenhöhe von Vorteil ist.

Ein weiterer Faktor für die Abweichung im Optimalwert kann auch die Beschränkung der oben genannten Arbeiten auf senkrechte Spitzen sein. Die Neigung der Spitzen von $\gamma = 110°$, d.h. 20° gegen die Strömungsrichtung, sorgt für einen anders ausgeprägten Kammerwirbel als dies bei geraden Spitzen der Fall ist. Auf diesen Sachverhalt wird später noch näher eingegangen. Die gegen die Strömung geneigten Spitzen beeinflussen außerdem die Ablösung über der Spitze positiv, sodass die Suche den höchsten für γ erlaubten Wert liefert.

Bei Durchblicklabyrinthdichtungen mit geringen Spaltweiten, wie in diesem Fall, steigt der Durchflussbeiwert mit steigendem Druckverhältnis. Daher ist der kleinste zulässige Wert für π das logische Ergebnis. In der realen Anwendung ist das Druckverhältnis in aller Regel als Randbedingung vorgegeben. Hier wurde jedoch als zusätzlicher Test der Methodik π innerhalb der oben genannten Grenzen variabel angesetzt.

Die Optimumssuche mit dem Gaußprozess (GP) als Modell kommt bezüglich der Labyrinthgeometrie zu demselben Ergebnis wie die mit Künstlichen Neuronalen Netzen. Der vorhergesagte

Durchflussbeiwert liegt mit $C_D = 0{,}4753$ leicht unter dem der durch das KNN-Ensemble vorhergesagten. Der Suchlauf, durchgeführt mit dem Particle Swarm Optimizer, benötigt im Mittel 464 Zielfunktionsaufrufe mit einer Standardabweichung von 84,8 ZF-Aufrufen. Differential Evolution sucht wiederum mehr als viermal so lange. Hier werden im Mittel 1817 Zielfunktionsaufrufe bei einer Standardabweichung von 484,5 ZF-Aufrufen benötigt.

Um die oben genannten Ergebnisse zu validieren und zu vergleichen, wurde ein weiterer Suchlauf mit der CFD-Methode als Modell für die Leckagevorhersage und PSO als Suchalgorithmus gestartet. Um bereits bekannte Ergebnisse aus vorangegangenen Simulationen wiederzuverwenden, wurde das CFD-Archiv in die Suche eingebunden. Bei geringer Differenz eines jeden Parameters des aktuell zu evaluierenden Individuums zu einem im Archiv bereits vorhandenen, wird der archivierte Zielfunktionswert verwendet, anstatt eine neue CFD-Simulation zu starten. Die maximalen Differenzen wurden wie folgt definiert:

- $\Delta s = 10^{-5}\,m$
- $\Delta S_B = 10^{-5}\,m$
- $\Delta S_H = 10^{-4}\,m$
- $\Delta t = 10^{-4}\,m$
- $\Delta \gamma = 1^{\circ}$

Die CFD-Optimierung liefert dasselbe Ergebnis in Form der Parameterwerte wie sowohl KNN als auch GP bei einem Durchflussbeiwert von $C_D = 0{,}4999$. In Abbildung 6.2 ist das Geschwindigkeitsfeld für die Optimalgeometrie bei einem Druckverhältnis von $\pi = 1{,}5$ dargestellt. Über den Spitzen ist die ausgeprägte Strömungsablösung mit entsprechender Einschnürung des Strahls gut erkennbar, die die effektive Spaltweite über der ersten Spitze um annähernd 25% und über den nächsten beiden Spitzen um circa 15% relativ zur nominellen Spaltweite verringert. Hinter der Spitze bildet sich eine Ablösung aus, da der Kammerwirbel der stromab liegenden Spitzenflanke nicht bis an den Spalt folgen kann. Der Hauptkammerwirbel nimmt somit eine nahezu rechteckige Form an. Abbildung 6.3 zeigt die turbulente kinetische Energie k, die insbesondere in der Scherschicht zwischen Wandstrahl und Kammerwirbel hoch ist. In den Kammern wird die Turbulenz teilweise wieder dissipiert. Hinter der letzten Spitze ist der Geschwindigkeitsgradient in der Scherschicht besonders hoch, da der Wirbel hinter der Dichtung sehr langgestreckt ist und die Geschwindigkeit unmittelbar hinter der letzten Spitze sehr gering ist. Im Gegensatz dazu hat die Strömung durch den stärker rotierenden Kammerwirbel bereits eine größere Vorgeschwindigkeit, die eine geringere Scherung bewirkt. In Abbildung 6.4 wird die Wirbelstärke (nach Gl. 6.1) dargestellt, wodurch die Form des Kammerwirbels ersichtlich wird.

$$\vec{\omega} = \vec{\nabla} \times \vec{u} \tag{6.1}$$

Für die Suche wurden 840 Zielfunktionsaufrufe benötigt, von denen nur 394 simuliert werden mussten. Das Archiv konnte dadurch die benötigte Zeit um mehr als die Hälfte reduzieren ohne das

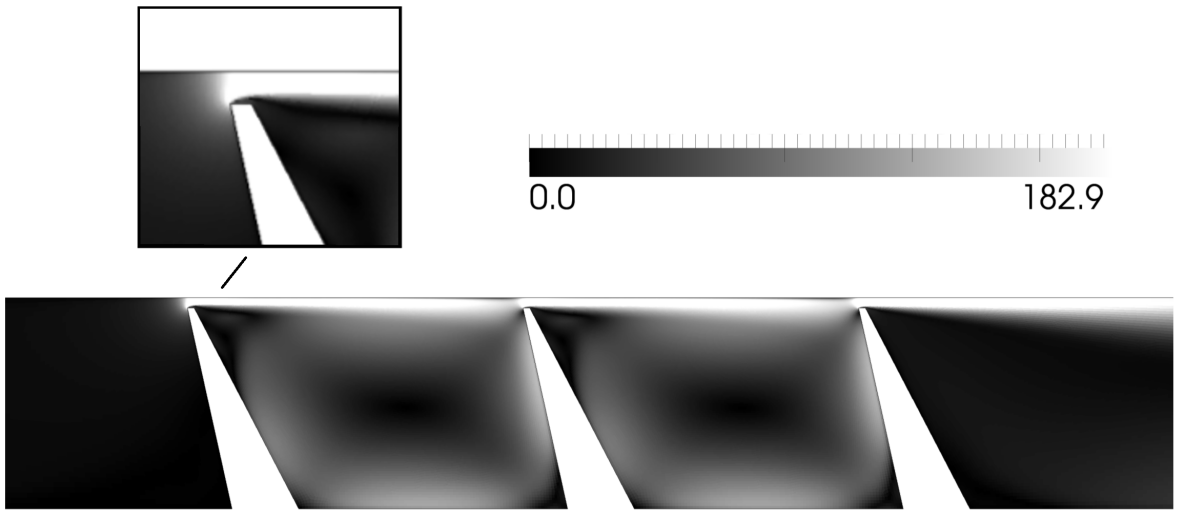

Abb. 6.2: Optimale Geometrie des Durchblicklabyrinths mit Geschwindigkeitsfeld $[m/s]$

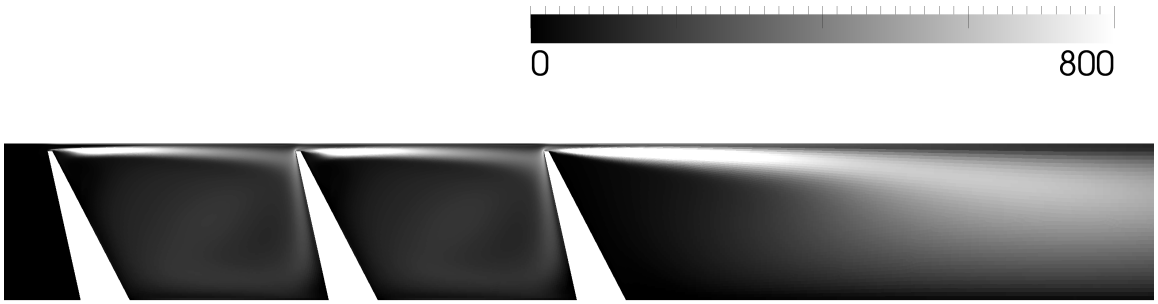

Abb. 6.3: Turbulente kinetische Energie im Strömungsfeld der Optimalgeometrie $[m^2/s^2]$

Ergebnis zu beeinflussen. Bei der Wahl der maximal zulässigen Differenzen der Parameter beim Durchsuchen des Archivs ist darauf zu achten, dass diese nicht zu hoch angesetzt werden. Höhere erlaubte Differenzen würden zwar die Suche weiter beschleunigen, in der Nähe bekannter Punkte jedoch einen konstanten C_D-Wert erzwingen und somit eine Art Diskretisierung der Zielfunktion mit geringer "Auflösung" der wahren Zielfunktionslandschaft bewirken. Hintergrund ist, dass die Archivsuche nach einem bekannten Individuum *in der Nähe* des gesuchten Individuums sucht. Allgemeingültige Regeln für die maximale Suchweite können nicht gegeben werden, jedoch bieten die hier gewählten Werte für das hier besprochene Problem eine gute Ausgangsbasis.

6.1.4 Fazit

Wird das Ergebnis der CFD-Zielfunktion als Referenz angesehen, liegt die Vorhersage des KNN mit einer Abweichung von 1,9% im Rahmen der Vorhersagegenauigkeit der CFD-Methode (vgl. Abbildung 5.4). Der Gaußprozess liefert ein etwas ungenaueres Ergebnis mit einer Abweichung von 4,9%. Beide Modelle kommen jedoch auf den gleichen optimalen Parametersatz wie die CFD-Methode. Der etwas ungenauere Gaußprozess bietet allerdings, wie in Kapitel 2 bereits

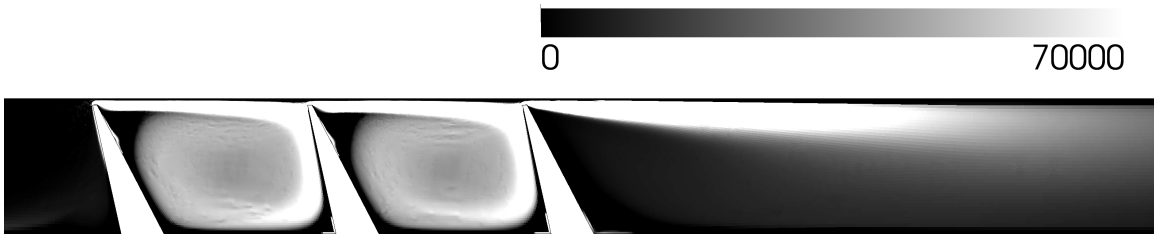

Abb. 6.4: Wirbelstärke senkrecht zur Ansichtsebene im Strömungsfeld der Optimalgeometrie $[1/s]$

erwähnt, den großen Vorteil, das Maß für die Unsicherheit der Vorhersage direkt als Auslöser für den Start der CFD-Methode bei kombiniertem Einsatz zu verwenden.

Abbildung 6.5 zeigt den Verlauf der Eingangs- und Ausgangsparameter während der Optimierung. Beispielhaft ist hier der Suchlauf dargestellt, bei dem die CFD-Methode zum Einsatz kam. Die Verläufe der Optimierungen mit KNN oder dem Gaußprozess sehen sehr ähnlich aus. Offensichtlich wird bei dieser Darstellung, dass Parameter, die einen starken Einfluss auf den Durchflussbeiwert haben, sehr schnell gegen ihren optimalen Wert streben. Dies ist hier für die Spaltweite s, das Druckverhältnis π, die Spitzenbreite S_B und die Teilung t der Fall. Die Spitzenhöhe S_H und der Spitzenneigungswinkel γ haben einen kleineren Einfluss auf die Leckage, beziehungsweise beeinflussen sich möglicherweise sogar gegenseitig, sodass die Suche nach den optimalen Werten aufwendiger ist und mehr Zielfunktionsaufrufe benötigt. Auch sind die Änderungen des C_D-Wertes als Funktion der Spitzenneigung gering, sodass der Selektionsdruck entsprechend klein ist, was zu einer geringeren Variationsgeschwindigkeit beziehungsweise Schrittweite bei der Suche führt.

Der Optimierungsalgorithmus besitzt keine Kenntnis der zugrundeliegenden Physik, die durch die CFD-Simulation abgebildet wird. Aus den Verlaufsdiagrammen der variierenden Parameter lassen sich jedoch die Korrelationen und Sensitivitäten der einzelnen Parameter ablesen. Dies gilt sowohl gegenüber dem Zielwert C_D als auch bezüglich der Beeinflussung der Parameter untereinander. Es lässt sich somit aus dem simplen Automatismus direkt ein physikalisches Verständnis ableiten.

6.2 Optimales Stufenlabyrinth

Im letzten Abschnitt wurden die im Rahmen dieser Arbeit entstandenen Methoden anhand eines relativ einfachen Beispieles vorgestellt und die erfolgreiche Anwendung demonstriert. Dabei wurden Annahmen getroffen, die in der Praxis bei der Auslegung einer neuen Dichtung weniger relevant sind, wie beispielsweise eine variable Spaltweite. Die Spaltweite ist in der Regel keine Größe, die bei der Optimierung eine Rolle spielt, da die minimale Spaltweite in Bezug auf den Leckagemassenstrom, und nicht auf den C_D-Wert, auch die beste Dichtwirkung bei ansonsten gleichen Bedingungen bewirkt. Die minimale Spaltweite wird dabei durch Größen wie thermischer und fliehkraftbedingter Ausdehnung des Rotors, exzentrischer Auslenkung des

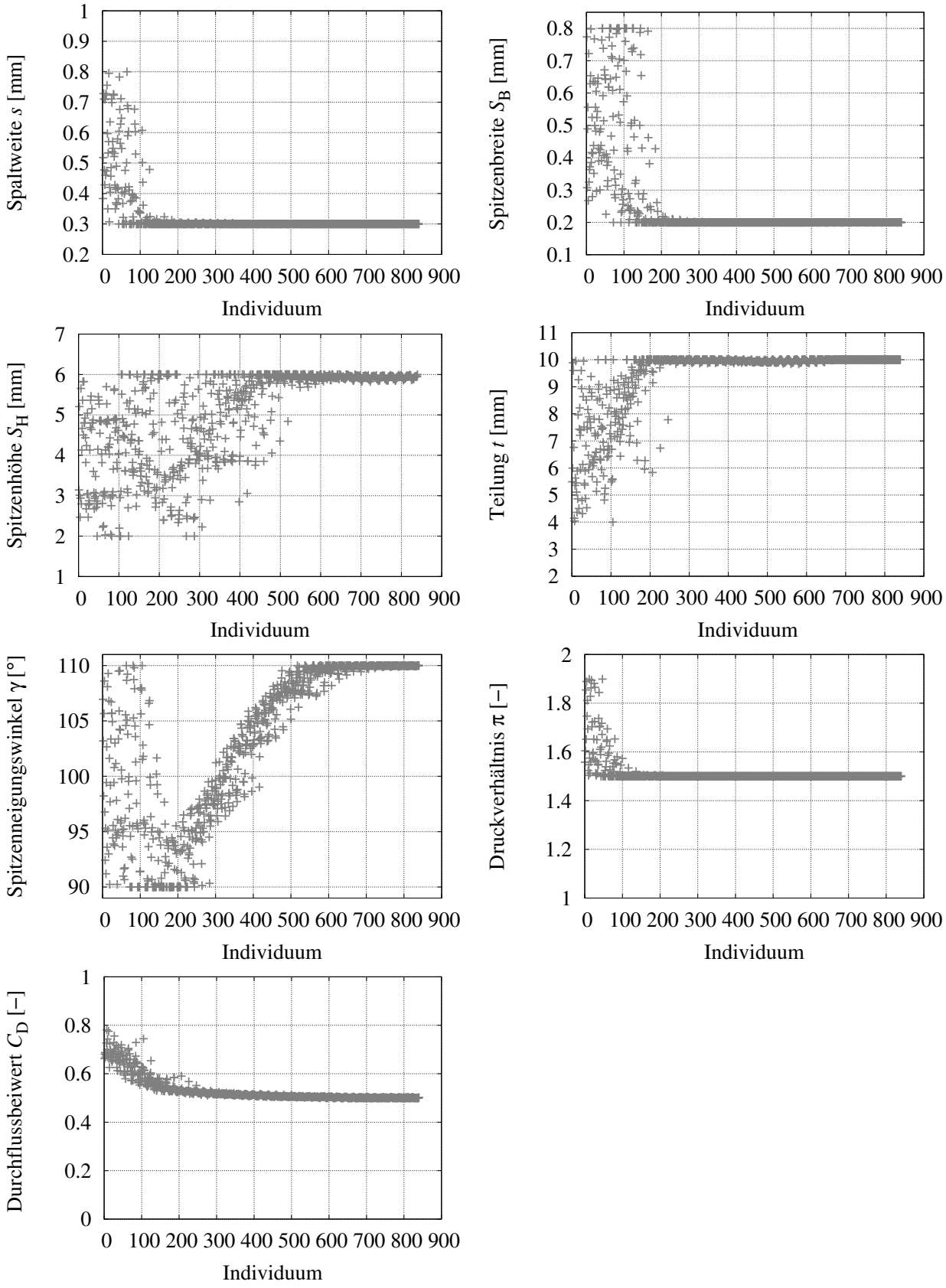

Abb. 6.5: Variation der Parameter und C_D-Wert über Optimierungsverlauf (Einzelzieloptimierung des Durchblicklabyrinths)

Rotors durch z.B. Schwingungen und akzeptabler Überschneidung zwischen Rotor und Stator beim Anstreifen bestimmt. Die Spaltweite soll daher im Folgenden konstant gehalten werden, um dieser Maßgabe gerecht zu werden.

Bei diesem Optimierungsbeispiel handelt es sich um die Suche der Geometrie eines konvergenten Stufenlabyrinths mit drei Spitzen, die die geringste Leckage aufweist. Diese Bauform findet besonders häufig Verwendung im Bereich des Sekundärluftsystems von Flugtriebwerken, in dem es neben der guten Dichtwirkung besonders auf kompakte Bauform und geringes Gewicht ankommt. In diesem Beispiel soll nur der Neuzustand der Labyrinthdichtung betrachtet werden. Es werden also scharfkantige Spitzen mit Spitzenradius $R = 0$ angenommen und der Stator als glatte Fläche simuliert.

6.2.1 Vorbereitung der Optimumssuche

Für die Suche nach der besten Geometrie wurde die Kombination aus vorher angelerntem KNN und der CFD-Methode als Ausweichverfahren für die Vorhersage der Leckage, die auch in diesem Beispiel wieder durch den C_D-Wert beschrieben wird, verwendet. In Tabelle 6.3 sind die Bereiche der Lerndatenparameter und die Schrittweite der Variationen aufgelistet. Die Gesamtheit aller Kombinationen ergibt $3^8 = 6561$ Datentupel, die mit der automatischen CFD-Methode berechnet wurden. Wie schon beim ersten Beispiel wurden auch hier wieder 100 Künstliche Neuronale Netzwerke trainiert, die zehn besten ausgewählt und zur Vorhersage das arithmetische Mittel dieses Ensembles herangezogen. Als Testdaten für die Wahl der besten KNN wurden wie schon beim vorherigen Beispiel weitere Datentupel berechnet (diesmal 128), die auf den Raumdiagonalen zwischen den Lerndaten liegen.

Der Suchraum bei der Optimierung wurde etwas größer als im voranstehenden Beispiel gestaltet. Dazu wurden die Schranken für die Parameter Spitzenneigung γ und Spitzenbreite S_B gelockert. Die Schranken der variablen Parameter sind in Tabelle 6.4 dargestellt. Hier sind auch die Ergebnisse aufgeführt. Der eingesetzte Optimierungsalgorithmus war PSO, der Particle Swarm Optimizer, der sich auch bei höherdimensionalen Problemen wie diesem, als guter Kompromiss zwischen sicherem Auffinden des globalen Optimums und geringer dafür notwendiger Anzahl an Zielfunktionsaufrufen herausgestellt hat.

6.2.2 Ergebnisse der Optimumssuche

Die beste Geometrie mit dem geringsten Durchflussbeiwert setzt sich aus den in Tabelle 6.4 gelisteten Ergebnisparametern zusammen. Als optimale Spitzenbreite S_B wurde der maximal zulässige Wert ermittelt. Für die Strömungsablösung über der Spitze ist dies in der Regel nicht ideal, da die Strömung sich wieder anlegen kann (vgl. auch das vorige Beispiel unter 6.1). Das Strömungsfeld in der "Idealgeometrie" ist in Abbildung 6.6 dargestellt und zeigt deutlich, warum in diesem Fall eine große Spitzenbreite vorteilhaft ist. Zwischen der Spitzenflanke und der Kante der Stufe am Stator ergibt sich aufgrund des sehr geringen Wertes der relativen Stufenverschiebung von $ST_S/t = 0{,}3$ und der sehr großen Spitzenbreite von $S_B = 0{,}8\,mm$ ein neuer Spalt mit kleinerem

Tab. 6.3: Parametervariation der KNN-Lerndaten für das konvergente Stufenlabyrinth

Parameter	Symbol	Wert	Schrittweite
Spitzenzahl	n	3	
Spaltweite	s	$0{,}3 \ldots 0{,}9\,mm$	$0{,}3\,mm$
Spitzenbreite	S_B	$0{,}2 \ldots 0{,}8\,mm$	$0{,}3\,mm$
Spitzenhöhe	S_H	$2{,}0 \ldots 6{,}0\,mm$	$2{,}0\,mm$
Teilung	t	$4{,}0 \ldots 10{,}0\,mm$	$3{,}0\,mm$
Spitzenneigung	γ	$90{,}0° \ldots 110{,}0°$	$10{,}0°$
Flankenwinkel	θ	$15{,}0°$	
Stufenhöhe	ST_H	$-1{,}9 \ldots -0{,}5\,mm$	$0{,}7\,mm$
rel. Stufenverschiebung	ST_S/t	$0{,}3 \ldots 0{,}7$	$0{,}2$
Druckverhältnis	π	$1{,}3 \ldots 1{,}9$	$0{,}3$

Tab. 6.4: Parameterbereich, Schranken und optimale Geometrie des konvergenten Stufenlabyrinths

Parameter	Symbol	Schranken	Ergebnis
Spitzenbreite	S_B	$0{,}2 \ldots 0{,}8\,mm$	$0{,}8\,mm$
Spitzenhöhe	S_H	$2{,}0 \ldots 6{,}0\,mm$	$6{,}0\,mm$
Teilung	t	$4{,}0 \ldots 10{,}0\,mm$	$4{,}0\,mm$
Spitzenneigung	γ	$80{,}0° \ldots 110{,}0°$	$110{,}0°$
Stufenhöhe	ST_H	$-1{,}9 \ldots -0{,}5\,mm$	$-1{,}9\,mm$
rel. Stufenverschiebung	ST_S/t	$0{,}3 \ldots 0{,}7$	$0{,}3$
Druckverhältnis	π	$1{,}3 \ldots 1{,}9$	$1{,}3$
ZF-Evaluationen			900
Bester C_D-Wert			0,15029

Querschnitt. Dass dieser für eine Leckagereduktion sorgt, ist offensichtlich und spiegelt sich im, für eine Geometrie mit nur drei Spitzen, sehr geringen Durchflussbeiwert von $C_D \approx 0{,}15$ wider. Weiterhin vorteilhaft für eine geringe Leckage ist die Richtung des Kammerwirbels, der sich entgegen dem Uhrzeigersinn (in der in Abbildung 6.6 gewählten Ansichtsebene) ausbildet, sodass das Fluid an der stromauf liegenden Flanke an die Spitze strömt und durch die Trägheit eine stark ausgeprägte Einschnürung über der Spitze bewirkt.

Aus Sicht des Optimierungsalgorithmus ist das gefundene Individuum tatsächlich das Optimum, da dessen Durchflussbeiwert am geringsten ist. Jedoch wird bei Betrachtung der Geometrie sofort ersichtlich, weswegen diese Geometrie für den Einsatz in Turbomaschinen nicht geeignet ist. Die

Labyrinthgeometrie, die sich aus der Kombination der Parameter für den geringsten C_D-Wert ergibt, besitzt ihren kleinsten Querschnitt nicht über der Spitze sondern an der stromabgewandten Flanke der Spitze. Da die Labrinthdichtungen jedoch, wie die Teile des Rotors, auf denen sie montiert sind, axialen Verschiebungen ausgesetzt sind, die mehrere Millimeter betragen können, würde es im Bereich des kleinsten Spalts zu starkem Anstreifen und schneller Zerstörung der Spitze beziehungsweise des Stators kommen. Eine Strategie, dieses Problem zu umgehen, wird im folgenden Beispiel (siehe 6.3) dargelegt.

In Abbildung 6.8 sind die Parameterwerte über den Verlauf der Optimierung aufgetragen. Die Parameter Druckverhältnis π und Spitzenneigungswinkel γ erreichen innerhalb weniger Schritte ihren finalen Wert. Die relative Stufenverschiebung ST_S/t strebt zuerst den höchsten erlaubten Wert an, bevor die Suche nach 350 Schritten auf dem kleinsten zugelassenen Wert konvergiert. Die Optimierung der Teilung t ist ebenfalls nach 350 Individuen abgeschlossen. Beim Durchflussbeiwert ist zwischen 250 und 350 Individuen bereits ein Plateau erkennbar. Der endgültige Wert der Stufenhöhe ST_H ist an diesem Punkt noch nicht gefunden, wobei die meisten Individuen an dieser Stelle bereits einen hohen (negativen) Wert besitzen, der nahe am später gefundenen Optimalwert liegen. Die Spitzenhöhe S_H liegt bei 350 Individuen in der Nähe ihres kleinsten Wertes von $2{,}0\,mm$. Bei der Spitzenbreite S_B ist zwischen 300 und circa 500 Individuen ein starker Trend von einem niedrigen auf den maximal erlaubten Wert von $0{,}8\,mm$ ersichtlich, der sich deutlich im Durchflussbeiwert widerspiegelt. Nach 500 Individuen hat sich dieser seinem endgültigen Wert bereits bis auf ca. 5% angenähert. Die Vergößerung der Spitzenhöhe von $S_H = 2{,}0\,mm$ auf $6{,}0\,mm$ bewirkt allerdings noch eine weitere Verbesserung. Diese kann auf eine weitere Verkleinerung des neu entstandenen kleinsten Spalts an der Spitzenflanke zurückgeführt werden. Nach 900 Individuen ist die Suche nach der Optimalgeometrie schließlich abgeschlossen, da sich die Individuen in der Population nicht mehr unterscheiden.

Im Verlauf der Optimierung wurde der größte Teil der Zielfunktionsevaluationen vom KNN-Ensemble übernommen. Nur 195 Individuen wurden direkt durch die CFD-Methode berechnet. Da für den hier verwendeten Suchraum bisher nur wenige CFD-Daten vorlagen, konnte das CFD-Archiv nicht zu einer Beschleunigung der Suche beitragen. Das Endergebnis lag schließlich im Gültigkeitsbereich der KNN. In Zeitstunden benötigte die Optimumssuche unter Verwendung von zehn Kernen eines AMD-Opteron-6727-Prozessors nur 12 Stunden. Die Arbeitszeit für das Einrichten der Suche kann auf weniger als eine halbe Stunde beziffert werden.

6.2.3 Fazit

Das Beispiel zeigt deutlich, dass die Ergebnisse von Optimierungsalgorithmen zur rechnergestützten, automatischen Optimierung nicht ohne Überprüfung verwendet werden dürfen. Von größter Bedeutung zum Erhalt “sinnvoller” Lösungen ist die exakte Definition des Problems und, wie es in diesem Beispiel nicht geschehen ist, der Randbedingungen. Bei der Vorgabe des erlaubten Parameterraums wurde keine Einschränkung hinsichtlich Abständen von Teilen der Geometrie vorgenommen. Dies führte zu der gezeigten Geometrie (Abb. 6.6) mit den angesprochenen Problemen. Eine klare Anweisung, die einen kleinen Bereich des Suchraums ausschließt, der solche problematischen Geometrien enthält, kann die Erzeugung dieser Geometrien vermeiden und

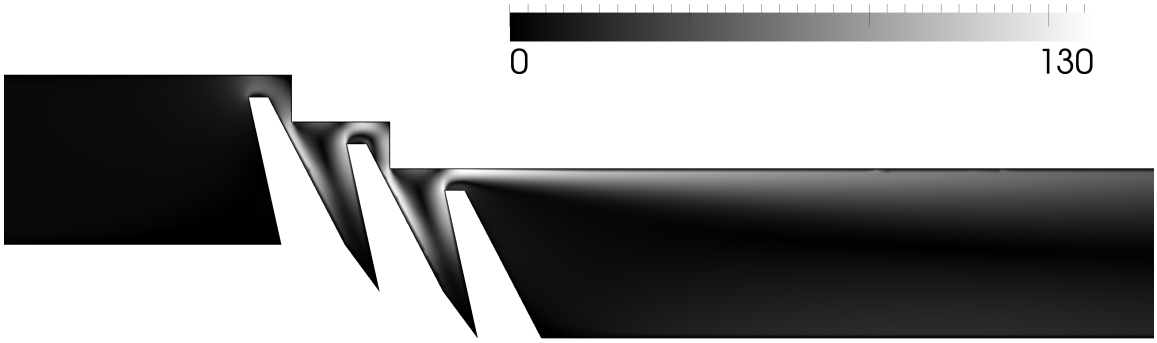

Abb. 6.6: Optimale Geometrie des konvergenten Labyrinths mit Geschwindigkeitsfeld $[m/s]$

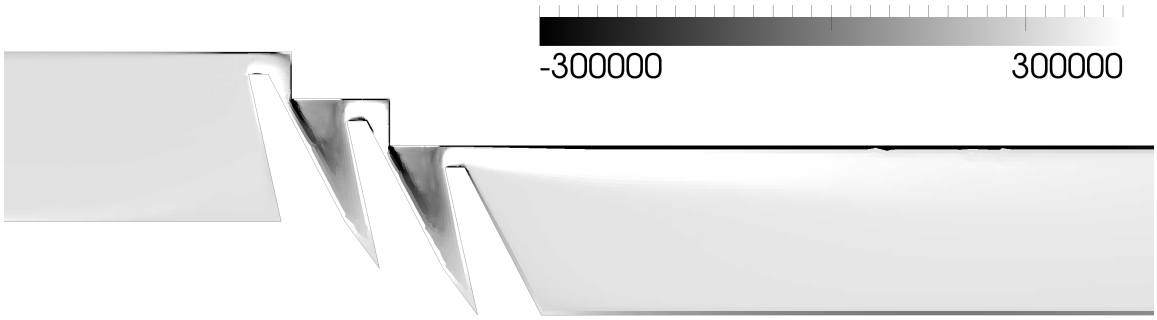

Abb. 6.7: Wirbelstärke senkrecht zur Ansichtsebene im Strömungsfeld der Optimalgeometrie $[1/s]$

eine praktikable Lösung bevorzugen. Dies soll im nächsten Optimierungsbeispiel berücksichtigt werden.

Andererseits zeigt das Beispiel, dass die rechnergestützte Optimumssuche neuartige Lösungen finden kann. Da der vorgegebene Suchraum *alle* möglichen Lösungen enthält, ist keine menschliche Kreativität erforderlich, um neue Geometrien zu entwickeln. Ein geeigneter Optimierungsalgorithmus findet die global beste Lösung. Er muss nur durch sinnvolle Einschränkungen an der Erzeugung ungeeigneter Lösungen gehindert werden.

6.3 Mehrzieloptimierung: Robustes Stufenlabyrinth

In den beiden vorangegangen Beispielen wurden jeweils nur die Geometrien im Neuzustand der Labyrinthdichtung untersucht. Die Spitzen wurden als perfekt scharfkantig und der Anstreifbelag am Stator als unverletzt, also ohne Anstreifnuten, betrachtet. Wie in Kapitel 2 erläutert wurde, ist dieser Zustand jedoch bereits nach wenigen Betriebsstunden nicht mehr anzutreffen. Die Kanten verrunden durch Anstreifvorgänge und die Anstreifbeläge werden ausgeschliffen. Ein Vorgang, der sich mit wachsender Einsatzdauer verstärkt. Eine gute Labyrinthdichtung muss daher eine geringe Leckage nicht nur im Neuzustand sondern auch im verschlissenen Zustand gewährleisten, um den Abfall der Leistung und des Wirkungsgrades der Turbomaschine durch erhöhte Leckage zu minimieren.

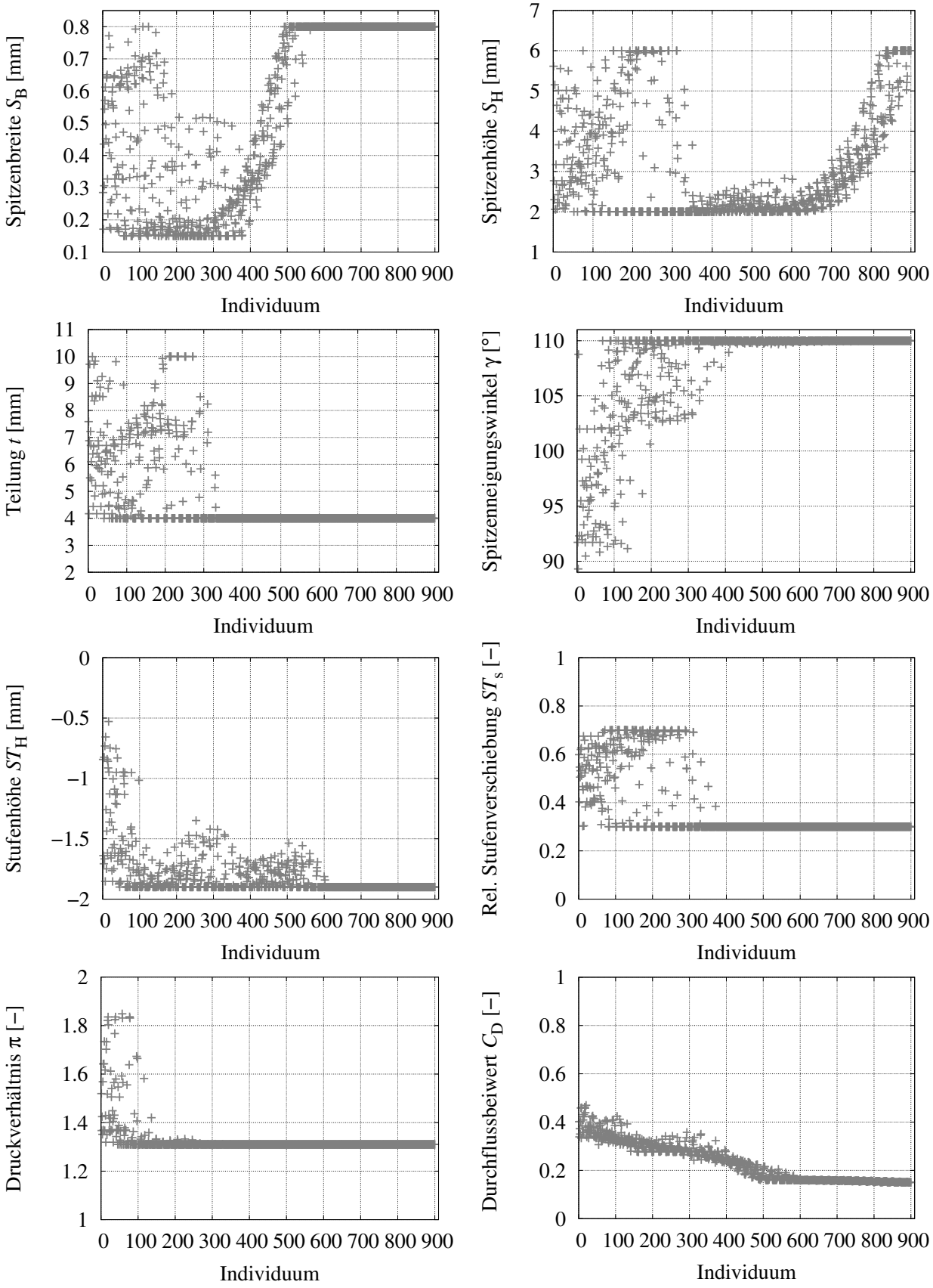

Abb. 6.8: Variation der Parameter und C_D-Wert über Optimierungsverlauf (Einzelzieloptimierung des konvergenten Stufenlabyrinths)

In diesem Beispiel soll daher untersucht werden, ob sich bei der Suche nach einer Optimalgeometrie eines konvergenten Stufenlabyrinths unterschiedliche Ergebnisse für den verschlissenen und Neuzustand ergeben. Die Optimumssuche hat daher zwei Zielfunktionen:

- ZF1: Neuzustand der Labyrinthdichtung - scharfkantige Spitzen, glatter Stator
- ZF2: Verschlissene Labyrinthdichtung - Kantenradius und Einlaufnuten

Das Ziel der Optimierung ist die Wahl des besten Kompromisses bei möglichst geringer Leckage in beiden Zuständen. Beim Einsatz eines Mehrziel-Optimierungsalgorithmus' wird als Ergebnis die Menge pareto-optimaler Individuen ausgegeben. Aus dieser Menge muss schließlich eine geeignete Lösung gewählt werden.

Mögliche Kompromisse würden sich ergeben, wenn sich die optimale Neu-Geometrie sehr von der optimalen verschlissenen Geometrie unterscheidet. In diesem Fall muss für die Auswahl der Anwendungsfall berücksichtigt werden. Bei Maschinen, bei denen von einer sehr kurzen Einsatzdauer ausgegangen werden kann, aber kurzzeitig Höchstleistungen verlangt werden, ist eine Wahl nahe der idealen Neugeometrie empfehlenswert. Kommt es mehr auf eine hohe Lebensdauer und robustes Verhalten an, z.B. im stationären Einsatz, wird die Wahl eher auf die optimale verschlissene Geometrie fallen. Jedoch spielt auch die Dauer bis sich ein entsprechender Verschleiß einstellt, eine entscheidende Rolle. Bei Flugtriebwerken beispielsweise, bei denen eine Einlaufphase üblicherweise in den ersten Betriebsstunden erfolgt und in der sich die Betriebsspalte einstellen, geschieht ein erster Verschleiss bereits frühzeitig.

Daher wird im Rahmen der Optimierung auf eine Gewichtung bewusst verzichtet - die Auswahl einer geeigneten Lösung sollte aus der pareto-optimalen Menge unter Berücksichtigung bisher gemachter Erfahrungen und des geplanten Einsatzrahmens sowie weiterer Nebenbedingungen durch geschultes Personal erfolgen. Dadurch können auch unvorhersehbare Ergebnisse in der Entscheidung berücksichtigt werden, was bei einer vorher vorgenommenen Gewichtung so nicht möglich wäre.

6.3.1 Vorbereitung der Optimumssuche

Um dieses Optimierungsbeispiel möglichst realistisch zu gestalten, werden zunächst einige Annahmen getroffen:

- Zu mininieren ist der Leckageverlust.
- Der Bauraum, den die Dichtung einnehmen darf, ist vorgegeben.
- Die Zahl der Spitzen darf variiert werden.
- Das Druckverhältnis ist durch die Drücke auf beiden Seiten der Dichtung vorgegeben.
- Der Verschleißmechanismus besteht aus Kantenverrundung und Nutenbildung.

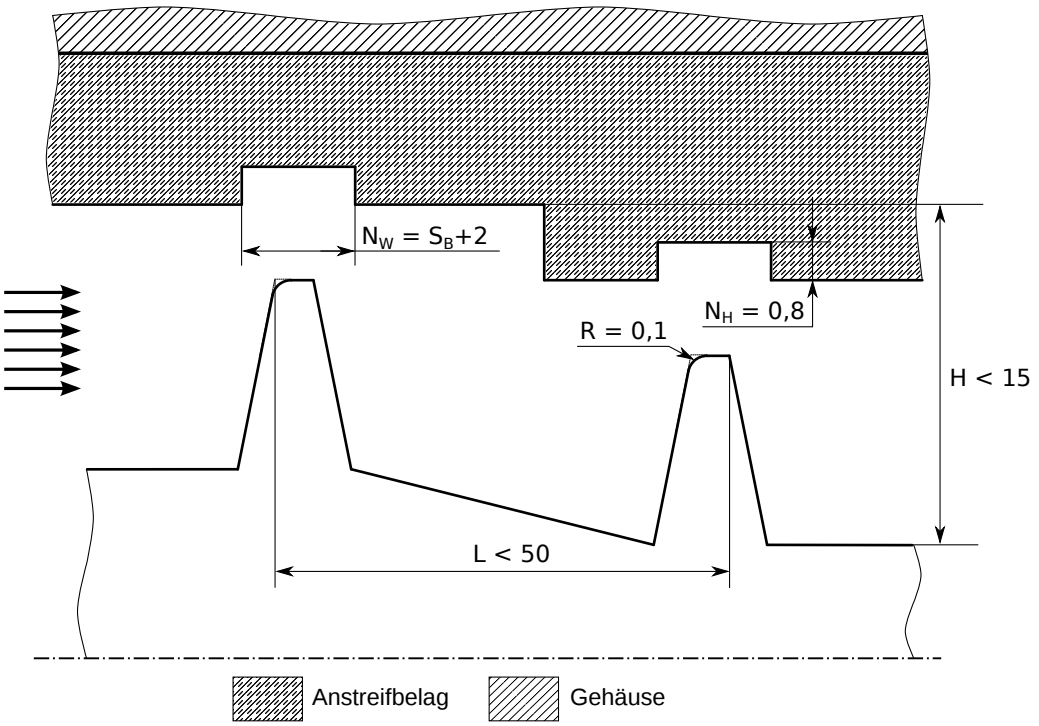

Abb. 6.9: Definition des verfügbaren Bauraums für die Mehrzieloptimierung mit Verschleiß (alle Maße in mm)

Der Bauraum, den die Dichtung einnehmen darf, setzt sich zusammen aus einer maximalen Gesamtlänge L, gemessen von der stromaufwärts liegenden Kante der ersten Spitze bis zur stromabwärts liegenden Kante der letzten Spitze, berechnet sich also nach $L = (n-1) \cdot t + S_B$. Sie ist hier begrenzt auf $50\,mm$. Die maximale Höhe der Dichtung H wird gemessen vom Statorradius über der ersten Spitze bis zum Fuß der letzten Spitze und berechnet sich nach $H = -ST_H \cdot n + S_H + s$. Der höchste Wert ist hier $15\,mm$. Zur Verdeutlichung sind die beiden Größen in Abbildung 6.9 eingezeichnet.

Innerhalb der oben genannten Schranken soll die Suche nach der beziehungsweise den Optimalgeometrien möglichst wenig eingeschränkt werden. Die Wertebereiche der einzelnen Parameter werden daher möglichst groß gewählt. Außerdem kommen weitere variable Parameter hinzu. Die optimale Spitzenzahl n und der Flankenwinkel θ werden zusätzlich zu den optimalen Werten der übrigen Parameter gesucht. Als fixe Parameter wird die Spaltweite mit $s = 0{,}5\,mm$ und das Druckverhältnis mit $\pi = 1{,}1$ festgelegt. In Tabelle 6.5 sind alle erlaubten Werte aufgeführt.

Als Gütemaß für die Optimierung wird in diesem Beispiel wieder der Durchflussbeiwert C_D verwendet. Durch die festgelegte Spaltweite ist dieser äquivalent zum Leckageverlust. Die in den beiden Zielfunktionen erzeugten Geometrien besitzen jedoch aufgrund der Nutenbildung für den verschlissenen Fall unterschiedliche Durchströmquerschnitte bei ansonsten gleichen Parameterwerten. Diese Querschnitte werden üblicherweise für die Berechnung des C_D-Wertes herangezogen. Zur besseren Vergleichbarkeit zwischen “alt” und “neu” wurde die Berechnung jedoch in diesem Beispiel auch für die verschlissenen Geometrien mit vergrößertem Querschnitt mit dem nominellen, also dem Neuzustand entsprechenden, Spaltquerschnitt ausgeführt. Dies

Tab. 6.5: Parameterbereiche und Beschränkungen des konvergenten Stufenlabyrinths

Parameter	Symbol	Schranken
Spitzenzahl	n	$3 \ldots 10$
Spitzenbreite	S_B	$0.3 \ldots 1.0\ mm$
Spitzenhöhe	S_H	$2.0 \ldots 5.0\ mm$
Teilung	t	$5.5 \ldots 15.0\ mm$
Spitzenneigung	γ	$90.0° \ldots 110.0°$
Flankenwinkel	θ	$10.0° \ldots 20.0°$
Stufenhöhe	ST_H	$-0.1 \ldots -4.5\ mm$
rel. Stufenverschiebung	ST_S/t	$0.4 \ldots 0.6$
Gesamtlänge	L	$0.0 \ldots 50.0\ mm$
Gesamthöhe	H	$0.0 \ldots 15.0\ mm$

kann dazu führen, dass Durchflussbeiwerte von $C_D > 1$ berechnet werden.

Als Verschleißmechanismus wird eine Kombination aus Kantenverrundung an der Spitze und Nutenbildung im Anstreifbelag des Stators festgelegt. Dabei wird von einer konstanten Rotordehnung, d.h. einer festen sich einstellenden Nuthöhe $N_H = 0{,}8\,mm$ ausgegangen. Die ausgeschliffene Anstreifnut hängt bei immer gleicher Axialverschiebung des Rotors von der Spitzenbreite ab. Die Axialverschiebung wurde mit $\pm 1\,mm$ angesetzt, die Nutweite ergibt sich somit aus $N_W = S_B + 2\,mm$. Die Spitze soll sich im Betriebszustand mittig unterhalb der Nut befinden. Der Kantenradius wird im verschlissenen Zustand auf $R = 0{,}1\,mm$ gesetzt – ein Wert, der sich am unteren Ende der im Betrieb beobachteten orientiert, aber bereits in Experimenten einen großen Einfluss zeigte (Wittig et al. (1986)). Er wird nur an der stromaufwärts liegenden Kante simuliert, da zum einen festgestellt wurde, dass der Einfluss eines Radius an der stromabwärts liegenden Kante auf die Leckage verschwindend gering ist. Zum anderen wird durch die hohe Feinheit des Rechengitters an dieser Stelle eine kleine instationäre Ablösung aufgelöst, die bei der stationären Simulation ein Konvergieren der Lösung verhindert.

Die Evaluation der Individuen, die durch den Optimierungsalgorithmus vorgeschlagen werden, muss in diesem Beispiel ausschließlich durch die CFD-Methode übernommen werden. Der große Parameterbereich und insbesondere die große Anzahl an Parametern verhindern die Berechnung einer sinnvollen Menge an Lerndaten zur Erzeugung von Künstlichen Neuronalen Netzen oder Gaußprozessen. Die Parameterzahl für die Lerndaten wird in diesem Fall schließlich nicht nur durch die acht variablen Optimierungsparameter bestimmt sondern zusätzlich um die Verschleißparameter Nutgeometrie und Kantenradius erweitert. Da diese nur gemeinsam auftreten, ergibt sich die Anzahl der mindestens erforderlichen Lerndaten somit zu $3^8 \cdot 2 = 13122$ Datentupeln. Diese Menge war im Vorfeld nicht mit vertretbarem Aufwand zu berechnen, sodass hier direkt auf die CFD-Methode zurückgegriffen wird. Für eine gewisse Verringerung der Zahl an notwendigen CFD-Simulationen wird das Archiv herangezogen.

Die Suche nach der Paretofront wird mit dem Non-Dominated-Sorting-Genetic-Algorithm II (NSGA2) umgesetzt. In vorangegangen analytischen Testfällen hatte sich gezeigt, dass die Individuen der finalen Generation dieses Algorithmus nicht nur näher an der bekannten wahren Paretofront liegen sondern auch gleichmäßiger entlang dieser verteilt waren. Diese Vorteile bestätigen den Vergleich von Deb et al. (2002) und machen NSGA2 geeigneter für das hier vorgestellte Beispiel als den anderen implementierten Mehrziel-Algorithmus PAES. Die maximale Anzahl an Generationen wurde auf 75 gesetzt. Bei 20 Individuen pro Generation plus 20 Individuen der Initialisierung ergeben sich somit 1520 Individuen, die jeweils einmal mit der Zielfunktion für den Neuzustand (im Folgenden "neu" genannt) und einmal mit simuliertem Verschleiß ("alt") berechnet werden müssen.

6.3.2 Ergebnisse der Optimumssuche

Für die Suche nach der Paretofront wurden insgesamt 2134 CFD-Simulationen durchgeführt. Dies entspricht 1067 Individuen, die direkt berechnet werden mussten. 301 Individuen verletzten mindestens eine Beschränkung (Länge oder Höhe) und wurden mit einem hohen arbiträren Strafwert von $C_D = 10^{20}$ bewertet. Weitere 152 Individuen waren gegen Ende der Suche bereits im Archiv vorhanden und mussten somit nicht mehrfach berechnet werden. Für die gesamte Optimierung – vielmehr die CFD-Simulationen, die mit deutlich über 99 Prozent den größten Teil der Zeit einnahmen – wurden zehn Tage und 17 Stunden auf zehn Prozessoren eines AMD-Opteron-6272-Prozessors benötigt (verfügbare Gleitkommaoperationen pro Sekunde ca. $8{,}4 \cdot 10^{10}$ (84 GFLOPS)). Jede Simulation dauerte im Mittel etwas mehr als sieben Minuten. Durch parallele Berechnung mehrerer Individuen auf jeweils mehreren Prozessoren könnte die Zeit bei entsprechend stärkerem Rechnereinsatz deutlich reduziert werden.

Die Ergebnisse der Suche sind in Tabelle 6.6 aufgeführt. Beim Vergleich der Parameter fällt auf, dass sich das beste "Neu"-Labyrinth in der Geometrie kaum vom besten verschlissenen "Alt"-Labyrinth unterscheidet. Nur beim Spitzenneigungswinkel γ und bei der relativen Stufenverschiebung ST_S/t sind Unterschiede, die über ein Prozent hinausgehen, zu bemerken.

In der letzten Zeile der Tabelle sind die Durchflussbeiwerte der beiden besten Individuen aufgelistet. Die in 6.3.1 angesprochene Berechnungsweise für die C_D-Werte führt dazu, dass auch Durchflussbeiwerte von $C_D > 1$ auftreten können, wie in Abbildung 6.10 und 6.11 zu sehen ist.

In Abbildung 6.10 sind die C_D-Werte jeder Geometrie über den Verlauf der Optimumssuche jeweils für die verschlissene und neue Dichtung dargestellt. Es wird ersichtlich, dass die Geometrie im Neuzustand bereits nach etwa 800 Individuen ihren Bestwert annähernd erreicht hat. Durch weitere Geometrievariationen kann für den verschlissenen Zustand jedoch von ungefähr dem 1000. Individuum an noch eine Verbesserung erzielt werden, nachdem zuvor eine Stagnation des C_D-Wertes zu beobachten war.

Abbildung 6.11 zeigt den Durchflussbeiwert im verschlissenen Zustand $C_{D,alt}$ aufgetragen über den Durchflussbeiwert im Neuzustand $C_{D,neu}$. In Teil I des Schaubilds sind alle Individuen, die während der Suche berechnet wurden, dargestellt. Die Suche verläuft in Richtung des Ursprungs und mündet schließlich im linken Arm der Punktwolke, wie aus dem Grauwert der Individuen

Tab. 6.6: Optimale Geometrien des konvergenten Stufenlabyrinths

Parameter	Symbol	Ergebnis (neu)	Ergebnis (alt)
Spitzenzahl	n	8	8
Spitzenbreite	S_B	0,303 mm	0,302 mm
Spitzenhöhe	S_H	2,01 mm	2,01 mm
Teilung	t	5,52 mm	5,52 mm
Spitzenneigung	γ	109,02°	106,08°
Flankenwinkel	θ	10,68°	10,43°
Stufenhöhe	ST_H	$-1{,}52$ mm	$-1{,}52$ mm
rel. Stufenverschiebung	ST_S/t	0,41	0,4
Gesamtlänge	L	38,94 mm	38,94 mm
Gesamthöhe	H	14,67 mm	14,67 mm
Durchflussbeiwert	C_D	0,183	0,362

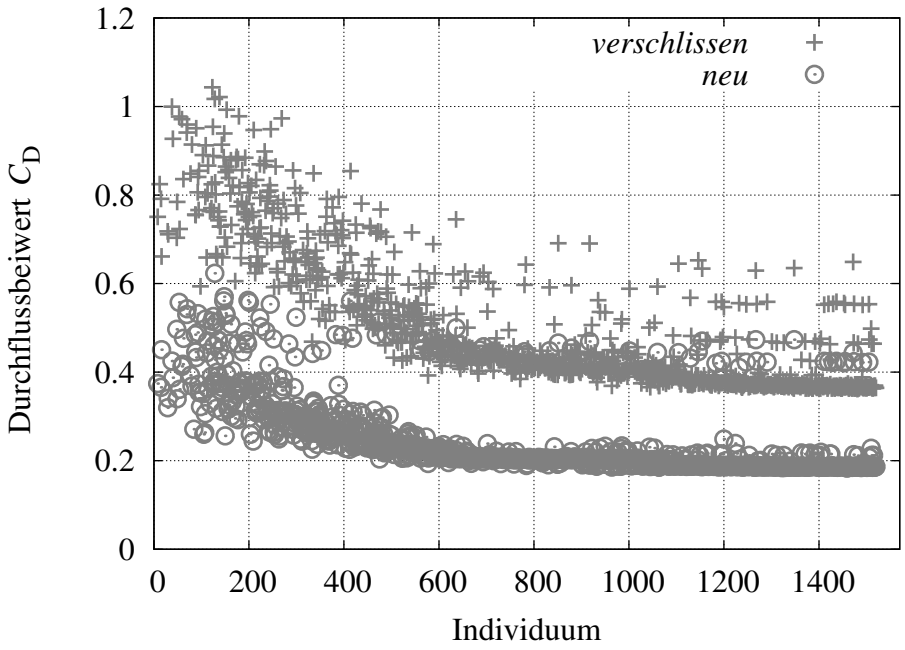

Abb. 6.10: C_D-Werte im Optimierungsverlauf (robustes Stufenlabyrinth)

nach dem „Geburts"-Zeitpunkt hervorgeht. In Teil II des Schaubilds ist die nur letzte Generation der pareto-optimalen Menge dargestellt. Fünf Individuen sind in dieser enthalten, wobei sich zwei in den C_D-Werten so ähnlich sind, dass sie fast übereinander liegen.

Aus der geringen Anzahl an Individuen in der resultierenden pareto-optimalen Menge und den sehr geringen Unterschieden zwischen den Geometrien lässt sich schließen, dass es bei dem hier betrachteten Fall gewissermaßen nur *ein* Optimum gibt. Durch die sehr kleinen Abstände zwischen den Punkten in Teil II des Schaubilds in Abbildung 6.11 lässt sich kaum von einer

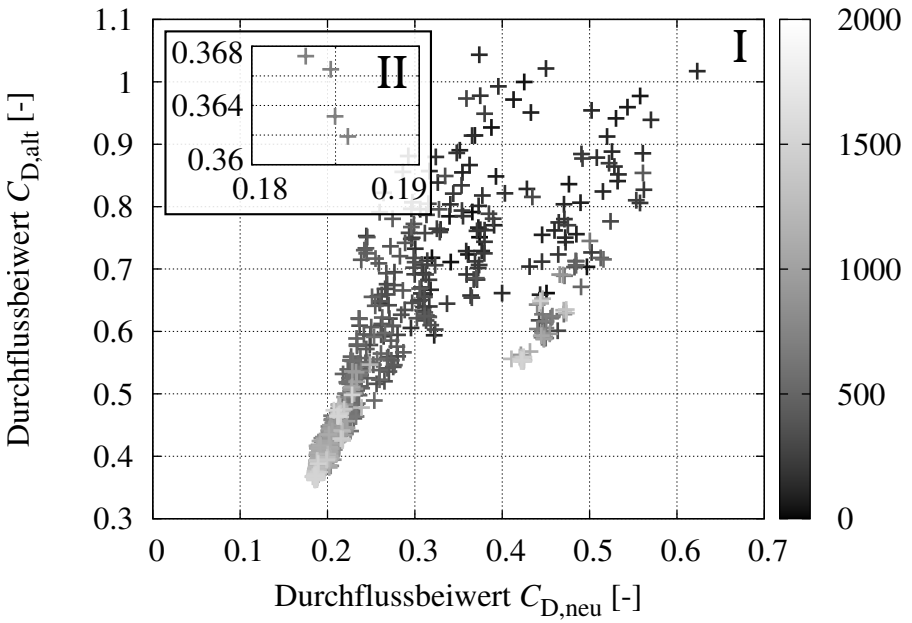

Abb. 6.11: C_D-Werte im direkten Vergleich (robustes Stufenlabyrinth)

Paretofront sprechen. Für die Konstruktion eines konvergenten Stufenlabyrinths innerhalb der vorgegebenen Grenzen ist dies sehr vorteilhaft: Bei der Auslegung muss kein Kompromiss zwischen Leckage im Neuzustand und verschlissenen Zustand gefunden werden. Die beste Neu-Geometrie ist gleichzeitig auch nahezu optimal im gealterten Zustand.

Bei den optimalen Parametern (siehe Tabelle 6.6) fällt auf, dass die zulässige Länge des Bauraums mit $L = 38{,}94\,mm$ nicht ausgenutzt wurde. Die Gesamthöhe liegt mit $H = 14{,}67\,mm$ nahe der oberen Grenze. Da die Spitzenhöhe mit nur $S_H = 2{,}01\,mm$ nahezu der minimal zulässigen entspricht, lässt sich schließen, dass eine maximale Spitzen*zahl* im Rahmen des Bauraums optimal für geringste Leckagewerte ist. Eine Erhöhung der Spitzenzahl wäre durch eine Verringerung der Stufenhöhe möglich gewesen, brachte aber keine Verbesserung.

Abbildung 6.12 stellt den Verlauf der Optimierungsparameter dar. Bei der Teilung t ist schon nach weniger als 400 Individuen klar, dass ein geringer Wert dieses Parameters bevorzugt wird. Es stellt sich letztendlich der minimal erlaubte ein. Auch die Spitzenhöhe S_H konvergiert schnell und erreicht bereits nach ca. 500 Individuen einen Wert von kleiner als $2{,}5\,mm$. Erst nach etwas mehr als 1200 Individuen wird jedoch der endgültige Wert fixiert. Dies hängt mit dem Sprung der Spitzenanzahl von $n = 7$ auf $n = 8$ zusammen. Bei der Stufenhöhe kristallisiert sich auch bereits nach etwa 500 Individuen ein Wert um $ST_H = -1{,}5\,mm$ heraus, der sich im weiteren Verlauf der Optimierung nur noch minimal ändert. Spitzenneigungswinkel γ, Flankenwinkel θ und die Spitzenbreite S_B erreichen zwischen 600 und 700 berechneten Individuen einen Wert, der nahe ihrem endgültigen liegt und nur noch kleine Variationen erfährt. Auch Individuen mit hoher Spitzenanzahl n werden früh bevorzugt. Die Variation von sieben auf acht Spitzen nach etwa 1000 Individuen ist dafür verantwortlich, dass der C_D-Wert noch einmal eine merkliche Verbesserung erfährt, die auch in Abbildung 6.10 zu erkennen ist. Die relative Stufenverschiebung ST_S/t ist der Parameter, für den lange keine endgültige Entscheidung getroffen werden kann. Erst nach ca. 1200 Individuen wird schließlich der kleinste zulässige Wert favorisiert.

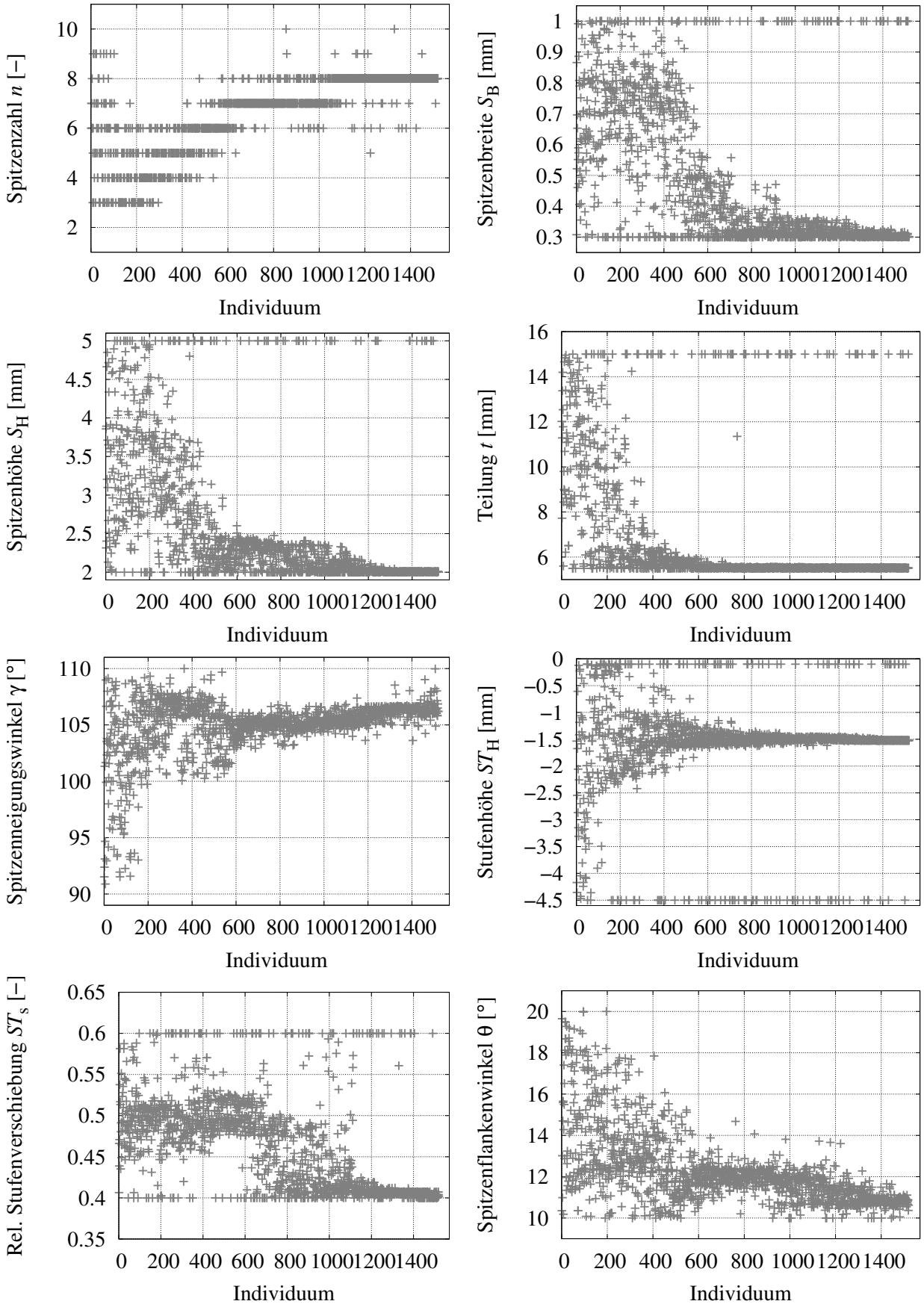

Abb. 6.12: Variation der Parameter über Optimierungsverlauf (robustes Stufenlabyrinth)

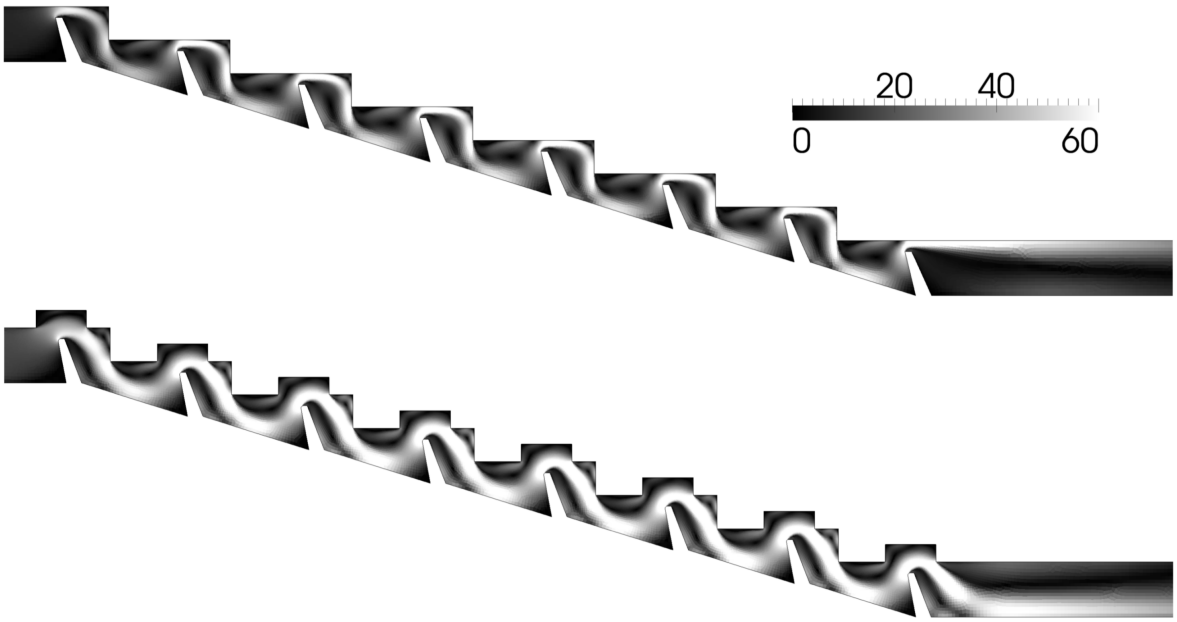

Abb. 6.13: Geschwindigkeitsfeld der besten Geometrien (robustes Stufenlabyrinth) $[m/s]$

Abbildung 6.13 zeigt das Geschwindigkeitsfeld des Fluids, wie es sich bei der Durchströmung der beiden Optimalgeometrien (jeweils Neuzustand und verschlissen) ausbildet. In beiden Fällen entstehen in den Labyrinthkammern zwei etwa gleich große Wirbel. Einer liegt im Ablösegebiet an der stromab liegenden Spitzenflanke und der zweite gegensinnig drehende befindet sich nach der Stufe vor der stromauf liegenden Flanke. Zusätzlich entwickeln sich zwei kleinere Wirbel, in der Ecke vor der Stufe und am stromauf liegenden Spitzenfuß, die sich jeweils mit entgegengesetzter Richtung zum benachbarten Hauptwirbel drehen. Bei der verschlissenen Geometrie fallen die Hauptwirbel etwas kleiner aus, da sich der Strahl verbreitert. Zusätzlich entstehen weitere Eckenwirbel in den Einlaufnuten. Die Wirbel, die sich in der Kammer nach der ersten Spitze ausbilden, sind in Abbildung 6.14 dargestellt. Die Wirbelstärke ist aus Abbildung 6.15 ersichtlich.

Deutlich sichtbar wird in den beiden Grafiken in Abbildung 6.13, dass sich die Strahleinschnürung zwischen der ersten und allen weiteren Spitzen stark unterscheiden. Die unterschiedliche Anströmung des Spaltes, die ab der zweiten Spitze in radialer Richtung nach außen entlang der Spitzenflanke anstatt in axialer Richtung parallel zum Stator erfolgt, bewirkt eine sichtbar stärkere Ablösung. Die bei der Suche gefundene Geometrie optimiert die Anströmung der Spitze auf eine solche maximale Strömungsablösung. Dieses Phänomen ist dasselbe im Neu- wie verschlissenen Zustand, wobei die Ablösung über der Spitze mit Einlaufnut sogar größer ausfällt. Die Leckage ist dennoch höher, da die tatsächliche Spaltweite durch die Nut erheblich größer ist.

6.3.3 Fazit

In diesem Beispiel konnte erfolgreich demonstriert werden, dass die mit der hier gewählte Vorgehensweise die Möglichkeit besteht, eine komplexe Optimierungsaufgabe mit mehreren Zielfunk-

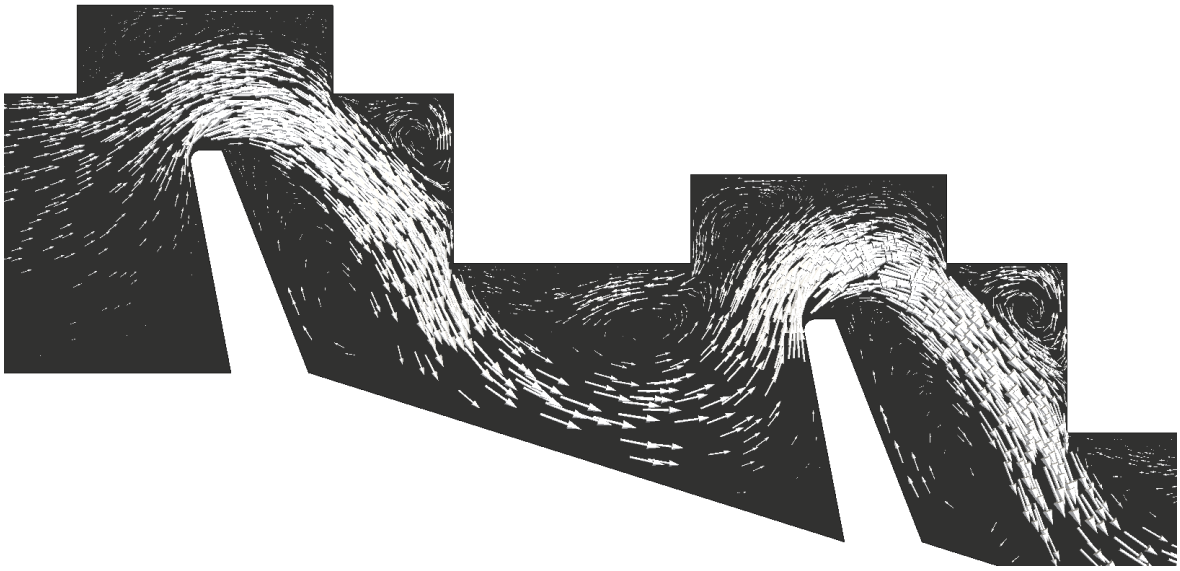

Abb. 6.14: Strömungsverlauf in der ersten Kammer

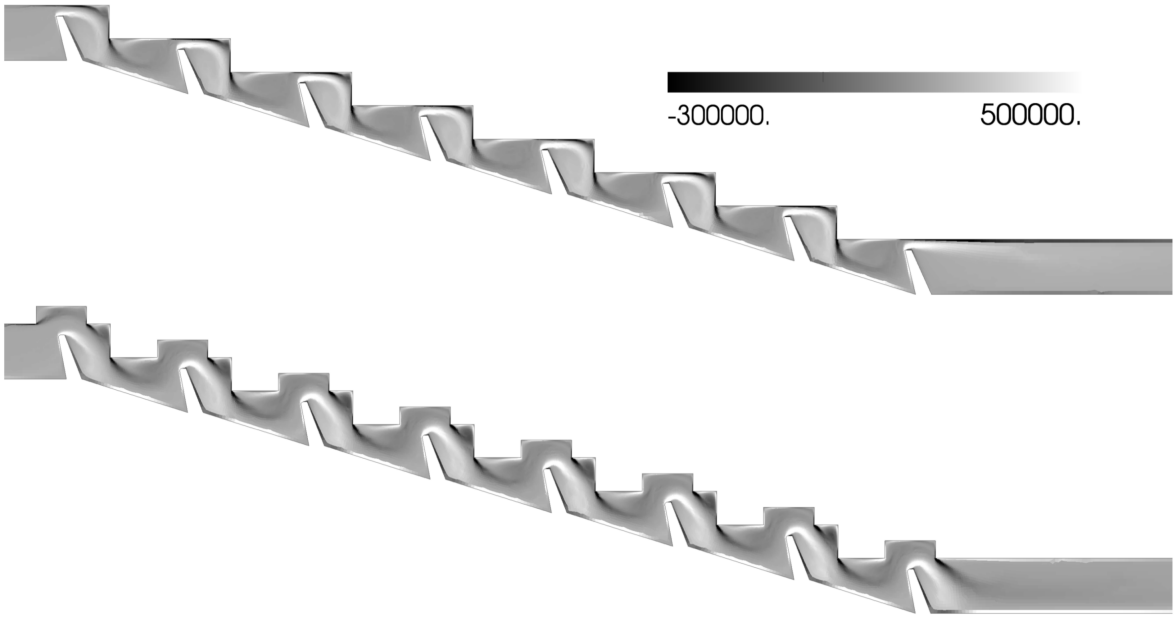

Abb. 6.15: Wirbelstärke senkrecht zur Ansichtsebene (beste Geometrien des robusten Stufenlabyrinths) $[1/s]$

tionen zu lösen. Die Tatsache, dass die Suche statt zu einer klassischen Paretofront mit mehreren Geometrien in *einer* Geometrie gemündet ist, vereinfacht das Problem des optimalen, robusten Stufenlabyrinths erheblich. Für dieses spezielle Beispiel ist es offenbar nicht nötig, nach einem Kompromiss zu suchen, der die beste Kombination geringer Leckage im Neu- und Altzustand darstellt. Die beste Geometrie im Neuzustand ist gleichzeitig auch die beste im verschlissenen Zustand.

Neben dieser Aussage kann als Ergebnis folgender Sachverhalt festgehalten werden. Bei ähnlichen

Größenverhältnissen bietet die gefundene Geometrie eine geringere Leckage als eine vergleichbare konvergente Stufenlabyrinthdichtung. Die Vergleichsgeometrie von Denecke (2007) besitzt eine Gesamtlänge $L = 24{,}4\,mm$ und eine Höhe von $H = 13{,}3\,mm$ bei vier Spitzen. Bei einem Druckverhältnis von $\pi = 1{,}1$ stellte sich ein experimentell bestimmter Durchflussbeiwert von $C_D = 0{,}281$ ein. Bei den Messungen wurde der Neuzustand der Dichtung hergestellt. Der C_D-Wert der Vergleichsdichtung liegt im Vergleich zur unverschlissenen Dichtung dieses Beispiels um fast 54 Prozent darüber. Dabei ist die Gesamtlänge der optimierten Dichtung um 37,3 Prozent und die Höhe um 9,3 Prozent höher. Die drastisch verminderte Leckage lässt die etwas größeren Anforderungen an den Bauraum und den höheren Fertigungsaufwand jedoch akzeptabel erscheinen.

7 Zusammenfassung

Die wachsenden Anforderungen an einen geringeren Verbrauch fossiler Brennstoffe betreffen in besonderem Maße Gasturbinen. Die Steigerung des Wirkungsgrades dieser Maschinen ist daher eine der wichtigsten Forschungsaufgaben im ingenieurtechnischen Umfeld. Der sparsame Umgang mit Kühlluft, die zur Bauteilkühlung erforderlich ist, dem Verdichter jedoch vor der Brennkammer entnommen wird und somit dem Kreisprozess nur noch partiell zur Verfügung steht, ist dabei von essentieller Bedeutung. Labyrinthdichtungen sind neben Bürstendichtungen und anderen adaptiven Dichtungsformen derzeit die einzigen Dichtungen, die in der heißen Umgebung unter hohen Relativgeschwindigkeiten über die gesamte Lebensdauer einer Gasturbine eingesetzt werden können.

Eine Methodik zu schaffen, um Labyrinthdichtungen für einen beliebigen Anwendungsfall über den gesamten Lebenszyklus optimal auszulegen, und somit die bestehende Lücke im Entwicklungsprozess für Turbomaschinen zu schließen, war das Hauptziel dieser Arbeit. Für diesen Zweck wurde ein in großen Teilen automatisiertes rechnergestütztes Formoptimierungswerkzeug entwickelt. Zunächst war dazu eine Bewertung und Wahl geeigneter Methoden zur Leckagevorhersage erforderlich. Mit einer Kombination aus datenbasierten Modellen wie Künstlichen Neuronalen Netzwerken und Gaußschen Prozessen mit numerischer Strömungssimulation konnte ein sehr guter Kompromiss aus effizienter Vorhersage und der Eignung für neuartige, bisher nicht untersuchte Geometrien bei sehr hoher Vorhersagegenauigkeit gefunden werden. Die datenbasierten Methoden wurden dabei mit vorher ermittelten Ergebnissen numerischer Strömunssimulationen trainiert. Der Einsatz von Messdaten aus experimentellen Untersuchungen für das Training ist ebenfalls möglich, jedoch erschweren unvollständige Angaben in der Literatur zu Geometrie und Randbedingungen die exakte Definition der Messpunkte. Durch den Einsatz der automatischen CFD-Methode ist es jederzeit möglich, den eingeschränkten bekannten Parameterraum, der den Modellen zugrundeliegt, flexibel zu erweitern. Dadurch können die grundsätzlichen Nachteile, die die Inter- und Extrapolation innerhalb der Modelle mit sich bringen, umgangen werden.

Für die Optimumssuche wurden mehrere Klassen von Optimierungsalgorithmen auf ihre Eignung zur Formoptimierung von Labyrinthdichtungen hin bewertet und einzelne Kandidaten ausgewählt. Für die monokriterielle Optimierung auf *ein* Ziel hin, wurden Differential Evolution, Downhill-Simplex, ein Hybridalgorithmus aus beiden, sowie der Particle Swarm Optimizer und Simulated Annealing gewählt. Gradientenbasierte Algorithmen wurden bewusst ausgeschlossen, um eine größtmögliche Flexibilität hinsichtlich der a priori unbekannten Zielfunktionen sicherzustellen. Als multikriterielle Optimierer wurden die Algorithmen Non-Dominated Sorting Genetic Algorithm II und Pareto Archive Evolution Strategy gewählt. Diese ermöglichen es nun, eine Anzahl möglicher Kandidaten (Labyrinthgeometrien) ohne vorherige Gewichtung zur ermitteln, aus denen ein geschulter Entwickler den besten Kompromiss für den jeweiligen Einsatzfall wählen kann.

Anhand von drei Beispielen wurden schließlich die Fähigkeiten der Optimierungsmethodik demonstriert. Mit steigendem Schwierigkeitsgrad wurde zunächst ein optimales Durchblickla-

byrinth gesucht. Dieses sowie das im zweiten Beispiel gesuchte ideale Stufenlabyrinth wurden nur im Neuzustand betrachtet, um die Eignung der datenbasierten Modelle im Zusammenspiel mit der numerischen Strömungssimulation zu zeigen. Im letzten Beispiel wurden dem Optimierungsalgorithmus möglichst wenige Vorgaben gemacht, um ein gegen Verschleiß robustes konvergentes Stufenlabyrinth zu finden.

Die Auswertung der letzten Untersuchung zeigte, dass sich die Geometrien des optimalen Neu- und des optimalen verschlissenen Labyrinths für die gewählten Rahmenbedingungen kaum unterscheiden. Somit wird in diesem speziellen Fall dem Konstrukteur die Entscheidung für einen Kompromiss aus beiden Zielen abgenommen. Diese Sicherheit bei der Entscheidung wäre ohne die Betrachtung beider Extrema (Neuzustand und verschlissen) nicht möglich gewesen. Das gesteckte Ziel, diese Lücke in der Auslegungskette für Turbomaschinen zu schließen, konnte somit erreicht werden.

In dieser Arbeit wurde nunmehr erfolgreich demonstriert, dass die gewählten Verfahren den Entwicklungsprozess im Bereich der Labyrinthdichtungen entscheidend unterstützen können. Die angesprochene Problematik der Bauteiloptimierung auf einen einzelnen Betriebspunkt oder einen einzelnen Punkt im Lebenszyklus hin, besteht jedoch auch in anderen Bereichen. Eine Erweiterung des in dieser Arbeit vorgeschlagenen Ansatzes stellt somit eine sinnvolle Fortsetzung dar und kann dazu beitragen, den Wirkungsgrad, die Leistung und die Lebensdauer von thermischen Turbomaschinen zu verbessern, sowie die korrespondierenden Emmissionen zu verringern.

Literaturverzeichnis

Asok, SP, Sankaranarayanasamy, K, Sundararajan, T, Rajesh, K und Sankar Ganeshan, G (2007): *Neural network and CFD-based optimisation of square cavity and curved cavity static labyrinth seals*. Tribology international, Bd. 40, S. 1204–1216.

BDL (2013): *BDL Report Energieeffizienz und Klimaschutz 2013*. Techn. ber., Bundesverband der Deutschen Luftverkehrswirtschaft.

Benckert, H. (1980): *Strömungsbedingte Federkennwerte in Labyrinthdichtungen*. Dissertation, Universität Stuttgart.

Bianchini, Cosimo, Micio, Mirko, Maiuolo, Francesco und Facchini, Bruno (2013): *Numerical investigation to support the design of a flat plate honeycomb seal test rig*. In: *ASME Turbo Expo 2013: Turbine Technical Conference and Exposition*, S. V03AT15A017–V03AT15A017. American Society of Mechanical Engineers.

Binghui, Jia und Xiaodong, Zhang (2011): *An optical fiber Blade Tip Clearance Sensor for Active Clearance Control Applications*. Procedia Engineering, Bd. 15, S. 984–988.

Büche, Dirk, Schraudolph, Nicol N und Koumoutsakos, Petros (2005): *Accelerating evolutionary algorithms with gaussian process fitness function models*. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, Bd. 35, S. 183–194.

Burgreen, Greg W und Baysal, Oktay (1994): *Aerodynamic shape optimization using preconditioned conjugate gradient methods*. AIAA journal, Bd. 32, S. 2145–2152.

Burgreen, Greg W und Baysal, Oktay (1996): *Three-dimensional aerodynamic shape optimization using discrete sensitivity analysis*. AIAA journal, Bd. 34, S. 1761–1770.

Campbell, D.A. (1978): *Gas Turbine Disc Sealing System Design*. AGARD-CP-273.

Chougule, Hasham H, Ramerth, Douglas, Ramchandran, Dhinagaran und Kandala, Ramnath (2006): *Numerical Investigation of Worn Labyrinth Seals*. In: *ASME Turbo Expo 2006: Power for Land, Sea, and Air*, S. 1483–1493. American Society of Mechanical Engineers.

Coello Coello, Carlos A und Lechuga, Maximino Salazar (2002): *MOPSO: A proposal for multiple objective particle swarm optimization*. In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, Bd. 2, S. 1051–1056. IEEE.

Coppinger, M., Cox, G., Hannis, J. und Cox, N. (2002): *Cycle Optimization using an Advanced Real Engine Performance Prediction Model*. ASME Paper GT-2002-30515.

Darwin, Charles (1859): *On the origins of species by means of natural selection*. London: Murray.

Deb, K., Agrawal, S., Pratap, A. und Meyarivan, T. (2002): *A fast and elitist multi-objective genetic algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation.

Denecke, Jens (2007): *Rotierende Labyrinthdichtungen mit Honigwabenanstreifbelägen-Untersuchung der Wechselwirkung von Durchflussverhalten, Drallverlauf und Totaltemperaturänderung*. Logos-Verlag.

Denecke, J., Dullenkopf, K. und Bauer, H.J. (2004a): *Vergleichende Bestimmung des Drallverlaufs in schnell rotierenden Labyrinthdichtungen mittels LDA und CFD*. 12. GALA Fachtagung, Lasermethoden in der Strömungstechnik, Karlsruhe.

Denecke, J., Dullenkopf, K. und Wittig, S. (2004b): *Influence of Preswirl and Rotation on Labyrinth Seal Leakage*. ISROMAC10-2004-105: Proceedings of the 10th International Symposiumon Transport Phenomena and Dynamics of Rotating Machinery ISROMAC,Hawaii.

Denecke, J., Dullenkopf, K., Wittig, S. und Bauer, H.-J. (2005a): *Experimental Investigation of the Total Temperature Increase and Swirl Development in Rotating Labyrinth Seals*. ASME-Paper GT-2005-68677.

Denecke, J., Färber, J., Dullenkopf, K. und Bauer, H.-J. (2005b): *Dimensional Analysis of Rotating Seals*. ASME-Paper GT-2005-68676.

Dörr, L. (1985): *Modellmessungen und Berechnungen zum Durchflussverhalten von Durchblicklabyrinthen unter Berücksichtigung der Übertragbarkeit*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Dréo, Johann (2006): *Metaheuristics for hard optimization: methods and case studies*. Springer.

El-Beltagy, Mohammed A und Keane, Andy J (2001): *Evolutionary optimization for computationally expensive problems using gaussian processes*. In: *Proc. Int. Conf. on Artificial Intelligence (IC-AI'2001), CSREA Press, Las Vegas*, S. 708–714. Citeseer.

Friedrich, F. (1933): *Untersuchungen über Das Verhalten der Schaufelspaltdichtungen in Gegenlaufdampfturbinen*. Dissertation, Technische Hochschule Karlsruhe.

Fröhlig, F. (2004): *Entwicklung und Implementierung eines hybriden Optimierungsverfahrens basierend auf einer Simplex-Methode und Evolutionsstrategie*. Diplomarbeit, Aerodynamisches Institut der RWTH Aachen.

Geuzaine, C. und Remacle, J.-F. (2009): *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*. International Journal for Numerical Methods in Engineering, Bd. 79, S. 1309–1331.

Halila, EE, Lenahan, DT und Thomas, TT (1982): *Energy efficient engine high pressure turbine test hardware detailed design report*.

He, Kun, Li, Jun, Yan, Xin und Feng, Zhenping (2012): *Investigations of the conjugate heat transfer and windage effect in stepped labyrinth seals*. International Journal of Heat and Mass Transfer, Bd. 55, S. 4536–4547.

Hendricks, R.C., Steinetz, B.M. und Braun, M.J. (2004a): *Turbomachinery Sealing and Secondary Flows: Part I - Review of Sealing Performance, Customer, Engine Designer, and Research Issues*. NASA TM-2004-211991/Part 1.

Hendricks, R.C., Tam, L.T: und Muszynska, A. (2004b): *Turbomachinery Sealing and Secondary Flows: Part II - Review of Rotordynamics Issues in Inhernetly Unsteady Flow Systems with Small Clearances*. NASA TM-2004-211991/Part 2.

Herrmann, Nico (2013): *Untersuchung und Vergleich von konkurrierenden adaptiven Dichtsystemen in Turbomaschinen hinsichtlich Effektivität und dynamischem Verhalten*. Techn. Ber. Förderkennzeichen BMWi 0327717 G. - Verbundvorhaben-Nr.: 01063652, Karlsruher Institut für Technologie.

Herrmann, Nico, Dullenkopf, Klaus und Bauer, Hans-Jörg (2013): *Flexible Seal Strip Design for Advanced Labyrinth Seals in Turbines*. ASME Paper GT2013-95424.

Hestenes, Magnus Rudolph (1980): *Conjugate direction methods in optimization*. Springer.

Ingber, Lester (1996): *Adaptive simulated annealing (ASA): Lessons learned*. Control and Cybernetics, Bd. 25, S. 33–54.

Jacobsen, Knut (1987): *Experimentelle Untersuchungen zum Durchfluß und Wärmeübergang in Durchblick- und Stufenlabyrinthdichtungen*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Jefferson-Loveday, Richard J, Nagabhushana Rao, V, Tyacke, James C und Tucker, Paul G (2013): *High-order detached eddy simulation, zonal LES and URANS of cavity and labyrinth seal flows*. International Journal for Numerical Methods in Fluids, Bd. 73, S. 830–846.

Jong, Kenneth Alan De (1975): *Analysis of the behavior of a class of genetic adaptive systems*.

Keller, C. (1934): *Strömungsversuche an Labyrinthdichtungen für Dampfturbinen*. Escher Wyss. Mitteilungen, Bd. 7, S. 9–13.

Kennedy, J. und Eberhart, R. (1995): *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks, Bd. IV, S. 1942–1948.

Kirkpatrick, Scott, Vecchi, MP et al. (1983): *Optimization by simulated annealing*. Science, Bd. 220, S. 671–680.

Knowles, Joshua und Corne, David (1999): *The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation*. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, Bd. 1. IEEE.

Komotori, K. (1957): *Flow observations in the Labyrinth packing*. Proceedings of the Fujihara Memorial Faculty of Engineering,K, Bd. 9, S. 1–9.

Kukkonen, Saku und Lampinen, Jouni (2005): *GDE3: The third evolution step of generalized differential evolution*. In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, Bd. 1, S. 443–450. IEEE.

Kwanka, Klaus (2001): *Der Einfluss von Labyrinthdichtungen auf die Dynamik von Turbomaschinen*. Nr. 415 In Reihe 7 Strömungstechnik. Fortschritt-Berichte VDI, Habilitation, Technische Universität München.

Lattime, S.B. und Steinetz, B.M. (2002): *Turbine Engine Clearance Control Systems: Current Practices and Furture Directions*. AIAA Paper: AIAA 2002-3790 and NASA TM-2002-211794.

Ludwig, L.P. und Bill, R.C. (1980): *Gas Path Sealing in Turbine Engines*. ASLE Transactions, Bd. 23, S. 1–22.

MacKay, David JC (1997): *Gaussian processes - a replacement for supervised neural networks?*.

MacKay, David JC (2003): *Information theory, inference and learning algorithms*. Cambridge University Press.

Martin, P.M. (1967): *Beitrag zur Durchflussberechnung von Spaltdichtungen*. Dissertation, Universität Karlsruhe.

Menter, F.R. (1993): *Zonal two-equation k-omega turbulence models for aerodynamic flows*. AIAA Paper 93-2906.

Menter, FR, Kuntz, M und Langtry, R (2003): *Ten years of industrial experience with the SST turbulence model*. Turbulence, heat and mass transfer, Bd. 4, S. 625–632.

Nelder, John A und Mead, Roger (1965): *A simplex method for function minimization*. Computer journal, Bd. 7, S. 308–313.

Nissen, Steffen et al. (2014): *FANN: Fast Artificial Neural Network Library*. http://leenissen.dk/fann/wp/.

Noll, B. (1986): *Numerische Berechnung brennkammertypischer Ein- und Zweiphasenströmungen*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Noll, B. (1992): *Möglichkeiten und Grenzen der numerischen Beschreibung von Strömungen in hochbelasteten Brennräumen*. Habilitationsschrift, Institut für Thermische Strömungsmaschinen, Fakultät für Maschinenbau der Universität Karlsruhe (TH).

OpenCFD (2012): *OpenFOAM User Guide*.

Oza, Nikunj C (2009): *Ensemble Data Mining Methods*.

Parsopoulos, Konstantinos E und Vrahatis, Michael N (2002): *Particle swarm optimization method in multiobjective problems*. In: *Proceedings of the 2002 ACM symposium on Applied computing*, S. 603–607. ACM.

Patankar, Suhas V und Spalding, D Brian (1972): *A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows*. International Journal of Heat and Mass Transfer, Bd. 15, S. 1787–1806.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. und Duchesnay, E. (2011): *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, Bd. 12, S. 2825–2830.

Pychynski, T., Dullenkopf, K., Bauer, H.-J. und Mikut, R. (2010): *Modelling the Labyrinth Seal Discharge Coefficient Using Data Mining Methods*. ASME Paper GT2010-22661.

Pychynski, Tim, Dullenkopf, Klaus und Bauer, Hans-Jörg (2013): *Theoretical Study on the Origin of Radial Cracks in Labyrinth Seal Fins due to Rubbing*. In: *ASME Turbo Expo 2013: Turbine Technical Conference and Exposition*, S. V07AT27A006–V07AT27A006. American Society of Mechanical Engineers.

Rasmussen, Carl Edward (2006): *Gaussian processes for machine learning*.

Remacle, J-F, Henrotte, F, Carrier-Baudouin, T, Béchet, Eric, Marchandise, E, Geuzaine, Christophe und Mouton, Thibaud (2013): *A frontal Delaunay quad mesh generator using the L norm*. International Journal for Numerical Methods in Engineering.

Rhode, D.L. und Allen, B.F. (1998): *Visualization and Measurements of Rub-Groove Leakage, Effects on Straight-Through Labyrinth Seals*. ASME Paper 98-GT-506.

Rhode, D.L. und Allen, B.F. (1999): *Measurement and Visualization of Leakage Effects of Rounded Teeth Tips and Rub-Grooves on Stepped Labyrinths*. ASME Paper 99-GT-377.

Schelling, U. (1988): *Numerischen Berechnung kompressibler Strömungen mit Wärmeübergang in Labyrinthdichtungen*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Scherer, Thomas (1994): *Grundlagen und Voraussetzungen der numerischen Beschreibung von Durchfluß und Wärmeuebergang in rotierenden Labyrinthdichtungen*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Schramm, V. (2010): *Labyrinthdichtungen maximaler Dichtwirkung: Ein Ansatz zur rechnerbasierten Formoptimierung*. Dissertation, Institut für Thermische Strömungsmaschinen, Karlsruher Institut für Technologie.

Shimoyama, Y. und Yamada, Y. (1957): *Experiments on the Labyrinth Packing (1st Report)*. Nihon-Kikai-Gakkai-ronbun-shun, Bd. 125, S. 44–49.

Spalart, PHILLIPE R und Allmaras, Steven R (1992): *A one equation turbulence model for aerodinamic flows.*. AIAA journal, Bd. 94.

Stewart, P.A.E. und Brasnett, K.A. (1978): *The Contribution of X-Ray to Gas Turbine Air Sealing, Technology*. AGARD-CP-237, S. 10.1–10.13.

Storn, R. und Price, K. (1995): *Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*.

Sutherland, William (1895): *XXXVII. The viscosity of mixed gases*. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Bd. 40, S. 421–431.

Ueda, T. und Kubo, T. (1967): *The Leakage of Air Through Radial Labyrinth Glands*. JSME International Journal, Bd. 10, S. 298–307.

Voorneveld, M. (2003): *Characterization of Pareto dominance*. Operations Research Letters, Bd. 31, S. 7–11.

Waschka, Walter (1991): *Zum Einfluß der Rotation auf das Durchfluß- und Warmeübergangsverhalten in Labyrinthdichtungen und Wellendurchführungen*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Weinberger, Tina (2014): *Einfluss geometrischer Labyrinth- und Honigwabenparameter auf das Durchfluss- und Wärmeübergangsverhalten von Labyrinthdichtungen. Experiment, Numerik und Data Mining*. Logos-Verlag.

Weise, Thomas (2009): *Global optimization algorithms-theory and application*. Citeseer.

Willenborg, Klaus (2007): *Ölfeuer in Flugtriebwerken: Selbstentzündung und Flammenausbreitung*. Dissertation, Institut für Thermische Strömungsmaschinen, Universität Karlsruhe (TH).

Winston, Patrick Henry (1987): *Künstliche Intelligenz*. Reihe künstliche Intelligenz. Addison-Wesley, Bonn [u.a.]. ISBN 3-925118-60-8.

Wittig, S., Dörr, L. und Kim, S. (1986): *Untersuchungen zum Einfluß der Kantenradien auf das Durchflußverhalten in Labyrinthdichtugen*. Techn. ber.

Xu, J., Ambrosia, M.S. und Rhode, D.L. (2004): *Effect of Rub-Groove Shape on the Leakage of Abradable Stepped LabyrinthSeals*. AIAA Paper 2004-3718.

Yegnanarayana, B (2009): *Artificial neural networks*. PHI Learning Pvt. Ltd.

Zimmermann, H. und Wolff, K.H. (1998): *Air System Correlations : Part 1 - Labyrinth Seals*. ASME-Paper 98-GT-206.

Zimmermann, H., Kammerer, A. und Wolff, K.H. (1994): *Performance of Worn Labyrinth Seals*. ASME-Paper 94-GT-131.

Anhang

A.1 Optimierungsumgebung TkinterOpti.py

```
#This file is part of Funky Optimizer.

#Funky Optimizer is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.

#Funky Optimizer is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#GNU General Public License for more details.

#You should have received a copy of the GNU General Public License
#along with Funky Optimizer.  If not, see <http://www.gnu.org/licenses/>.



from Tkinter import *
import re, sys
import optimizers
import targetfunctions
import time
import tkMessageBox
import tkFileDialog
from ScrolledText import ScrolledText
from datetime import datetime
import threading

class InputParamsMismatchError(Exception): pass
class InputParamsWrongTypeError(Exception): pass
class InputNoParNumberError(Exception): pass
class InputFileNotFoundError(Exception): pass
class InputFileNotReadableError(Exception): pass

class VerticalScrolledLabelFrame(LabelFrame):
    """A pure Tkinter scrollable frame with label that actually works!
    """
    def __init__(self, parent, *args, **kw):
        LabelFrame.__init__(self, parent, *args, **kw)

        # create a canvas object and a vertical scrollbar for scrolling it
        vscrollbar = Scrollbar(self, orient=VERTICAL)
        vscrollbar.pack(fill=Y, side=RIGHT, expand=FALSE)
        canvas = Canvas(self, bd=0, highlightthickness=0, yscrollcommand=vscrollbar.set)
        canvas.pack(side=LEFT, fill=BOTH, expand=TRUE)
        vscrollbar.config(command=canvas.yview)

        # reset the view
        canvas.xview_moveto(0)
        canvas.yview_moveto(0)

        # create a frame inside the canvas which will be scrolled with it
        self.interior = interior = LabelFrame(canvas)
        interior_id = canvas.create_window(0, 0, window=interior, anchor=NW)

```

```
        # track changes to the canvas and frame width and sync them,
59      # also updating the scrollbar
        def _configure_interior(event):
61          # update the scrollbars to match the size of the inner frame
            size = (interior.winfo_reqwidth(), interior.winfo_reqheight())
63          canvas.config(scrollregion="0 0 %s %s" % size)
            if interior.winfo_reqwidth() != canvas.winfo_width():
65              # update the canvas's width to fit the inner frame
                canvas.config(width=interior.winfo_reqwidth())
67      interior.bind('<Configure>', _configure_interior)

69      def _configure_canvas(event):
            if interior.winfo_reqwidth() != canvas.winfo_width():
71              # update the inner frame's width to fill the canvas
                canvas.itemconfigure(interior_id, width=canvas.winfo_width())
73      canvas.bind('<Configure>', _configure_canvas)

75      return


77
class Parameter(LabelFrame):
79      """Defines parameter input

81      """
        def __init__(self, master, parname="", lbound=0., ubound=0., resolution=1):
83          LabelFrame.__init__(self, master, class_="Parameter")
            self.grid()
85
            self.res = DoubleVar()      # resolution of parameter range
87          self.res.set(resolution)
            self.lower = DoubleVar()    # lower limit of parameter value
89          self.lower.set(lbound)
            self.upper = DoubleVar()    # upper limit of parameter value
91          self.upper.set(ubound)
            self.name = StringVar()     # name of parameter
93          self.name.set(parname)

95          self.createWidgets()

97      def createWidgets(self):

99          self.nameInput = Entry (self, textvariable=self.name)
            self.nameLabel = Label(self, text="name")
101         self.lowerInput = Spinbox (self, textvariable=self.lower, from_=-1E10, to=1E10,
        increment=0.00001, width=10)
            self.lowerLabel = Label(self, text="min")
103         self.upperInput = Spinbox (self, textvariable=self.upper, from_=-1E10, to=1E10,
        increment=0.00001, width=10)
            self.upperLabel = Label(self, text="max")
105         self.resInput = Spinbox (self, textvariable=self.res, from_=0, to=1E10, increment
        =0.00001, width=10)
            self.resLabel = Label(self, text="res")
107         self.delButton = Button(self, command=self.delParam, text="Delete")

109         self.nameLabel.grid(row=0, column=0, sticky=E)
            self.nameInput.grid(row=0, column=1)
111         self.upperLabel.grid(row=1, column=0, sticky=E)
            self.upperInput.grid(row=1, column=1)
113         self.lowerLabel.grid(row=2, column=0, sticky=E)
            self.lowerInput.grid(row=2, column=1)
115         self.resLabel.grid(row=3, column=0, sticky=E)
            self.resInput.grid(row=3, column=1)
117         self.delButton.grid(row=3, column=2)

119     def delParam(self):
```

```
        self.destroy()


class Constraint(LabelFrame):
    """Defines constraint

    """
    def __init__(self, master, conname="", definition="", lbound=0, ubound=0):
        LabelFrame.__init__(self, master, class_="Constraint")
        self.grid()

        self.lower = DoubleVar()  # lower limit of parameter value
        self.lower.set(lbound)
        self.upper = DoubleVar()  # upper limit of parameter value
        self.upper.set(ubound)
        self.name = StringVar()   # name of parameter
        self.name.set(conname)
        self.definition = StringVar()
        self.definition.set(definition)

        self.createWidgets()

    def createWidgets(self):

        self.nameInput = Entry (self, textvariable=self.name)
        self.nameLabel = Label(self, text="name")
        self.defInput = Entry (self, textvariable=self.definition)
        self.defLabel = Label(self, text="definition")
        self.lowerInput = Spinbox (self, textvariable=self.lower, from_=-1E10, to=1E10,
    increment=0.0001, width=10)
        self.lowerLabel = Label(self, text="min")
        self.upperInput = Spinbox (self, textvariable=self.upper, from_=-1E10, to=1E10,
    increment=0.0001, width=10)
        self.upperLabel = Label(self, text="max")
        self.delButton = Button(self, command=self.delCon, text="Delete")

        self.nameLabel.grid(row=0, column=0, sticky=E)
        self.nameInput.grid(row=0, column=1)
        self.defLabel.grid(row=1, column=0, sticky=E)
        self.defInput.grid(row=1, column=1)
        self.upperLabel.grid(row=2, column=0, sticky=E)
        self.upperInput.grid(row=2, column=1)
        self.lowerLabel.grid(row=3, column=0, sticky=E)
        self.lowerInput.grid(row=3, column=1)
        self.delButton.grid(row=3, column=2)

    def delCon(self):
        self.destroy()


class TFMenu(OptionMenu):
    """Defines target function input

    """
    def __init__(self, master, funcmodule, *fmlist):
        OptionMenu.__init__(self, master, funcmodule, *fmlist)
        self.grid()

        self.fm = funcmodule
        self.fmlist = fmlist


class Application(Frame):
```

```
183     def __init__(self, master=None):
            Frame.__init__(self, master)
185         self.grid(sticky = N+S+E+W)
            self.createWidgets()
187         sys.stdout = self

189         # build list of optimization algorithms and target functions from imported modules
            self.optialgolist = [elem for elem in dir(optimizers) if not "__" in elem and not "sys
        " in elem]
191         self.imported_tfs = [elem for elem in targetfunctions.gettflist()]

193         self.nparams = IntVar()
            self.nTF = IntVar()
195         self.optialgo = StringVar()
            self.ncons = IntVar()
197         self.tv = DoubleVar()
            self.logfile = "optilog_" + datetime.now().isoformat()[:-7]
199
        def write(self, txt):
201         """Redirection of stdout to Text widget and log file"""

203         # Follow new text only if scroll position is at the end
            scrollPos = self.textout.vbar.get()
205         self.textout.insert(END, txt)
            if scrollPos[1] == 1.0 or scrollPos[0] == scrollPos[1]:
207             self.textout.see("end")
            self.update()
209
            out = open(self.logfile, "a")
211         #out.write("# " + str(time.asctime(time.localtime())) + "\n")
            out.write(txt)
213         out.close


215
        def createWidgets(self):
217         """Creates widgets of main window.

219         """

221         self.top=self.winfo_toplevel()
            self.top.rowconfigure(0, weight=1)
223         self.top.columnconfigure(0, weight=1)

225         self.rowconfigure(0, weight=1)
            self.columnconfigure(0, weight=1)
227         self.textout = ScrolledText(self, wrap=WORD, width=80, bd=2, relief=SUNKEN)
            self.textout.grid(row=0, column=0, sticky = N+S+E+W)
229
            self.menu = Menu(self)
231         self.top["menu"] = self.menu
            self.menu.setmenu = Menu(self.menu, tearoff=0)
233         self.menu.helpmenu = Menu(self.menu, tearoff=0)
            self.menu.add_command(command=self.onExit, label="Quit")
235         self.menu.add_command(command=self.runrun, label="Run", state=DISABLED)
            self.menu.add_cascade(label="Settings", menu=self.menu.setmenu)
237         self.menu.setmenu.add_command(command=self.settings, label="General")
            self.menu.add_cascade(label="Help", menu=self.menu.helpmenu)
239         self.menu.helpmenu.add_command(command=self.showsettingshelp, label="Settings")
            self.menu.helpmenu.add_command(command=self.showtfhelp, label="Target Functions")
241         self.menu.helpmenu.add_command(command=self.about, label="About")
            self.menu.helpmenu.add_command(command=self.showlicense, label="License")
243
        def onExit(self):
245         #print "Cya!"
            quit()
```

```
247
249     def settings(self):
            """Opens a window in which to change the boundaries of all parameters
251
            """
253
            # find out if Settings-window is already there
255         toplevellist = [elem for elem in self.winfo_children() if elem.winfo_class()=="
        Toplevel" and elem.title() == "Settings"]
257         if (toplevellist):
                self.setWindow.deiconify()
259         else:
                # Disable Settings-menu in main window
261             self.menu.setmenu.entryconfigure(0, state=DISABLED)
263             # Read params.par and assign variables
                try:
265                 optimodule, funcmodules_fromfile, self.numparams, parnames, lbounds, ubounds,
        resolutions = readInput("params.par", "params")
                except:
267                 optimodule, funcmodules_fromfile, self.numparams, parnames, lbounds, ubounds,
        resolutions = loadDefaults("params")
269
                numTF = len(funcmodules_fromfile)
271
                self.nparams.set(self.numparams)
273             self.nTF.set(numTF)
275             if optimodule in self.optialgolist:
                    self.optialgo.set(optimodule)
277             else:
                    self.optialgo.set(self.optialgolist[0])
279
                self.funcmodules = [StringVar() for i in range(numTF)]
281             for i in range(numTF):
                    if funcmodules_fromfile[i] in self.imported_tfs:
283                     self.funcmodules[i].set(funcmodules_fromfile[i])
                    else:
285                     self.funcmodules[i].set(self.imported_tfs[i])
287             try:
                    connames, condefs, conlbounds, conubounds = readInput("params.par", "
        constraints")
289             except:
                    connames = []
291                 condefs = []
                    conlbounds = []
293                 conubounds = [] #  = loadDefaults("constraints")
295             self.numcons = len(connames)
                self.ncons.set(self.numcons)
297
                try:
299                 targetvalue = readInput("params.par", "target")
                except:
301                 targetvalue = loadDefaults(42)
303             self.tv.set(targetvalue)
305             # Create window
                self.setWindow = Toplevel (self, padx=10, pady=10)
307             self.setWindow.title("Settings")
```

```
            self.setWindow.transient(self)
309
            # Set up widgets
311             # First create inputs for each parameter in a frame.
            self.setWindow.paramFrame = VerticalScrolledLabelFrame(self.setWindow, text="
    Parameters")
313         self.params = [Parameter(self.setWindow.paramFrame.interior, parname=parnames[i],
    lbound=lbounds[i], ubound=ubounds[i], resolution=resolutions[i]) for i in range(self.
    numparams)]


315
                # Then build a frame for constraints.
317         self.setWindow.conFrame = VerticalScrolledLabelFrame(self.setWindow, text="
    Constraints")
            self.constraints = [Constraint(self.setWindow.conFrame.interior, conname=connames[
    i], definition = condefs[i], lbound=conlbounds[i], ubound=conubounds[i]) for i in range(
    self.numcons)]
319
                # Then build a frame for all the rest.
321         self.setWindow.otherFrame = LabelFrame(self.setWindow, text="Other Settings")

323             # Number of parameters can be set here...
            self.setWindow.otherFrame.numparamslabel = LabelFrame(self.setWindow.otherFrame,
    text="No. of params")
325         self.setWindow.otherFrame.numparamslabel.numps = Spinbox (self.setWindow.
    otherFrame.numparamslabel, textvariable=self.nparams, from_=1, to=100, increment=1, width
    =10)
            self.setWindow.otherFrame.numparamslabel.setNPBtn = Button (self.setWindow.
    otherFrame.numparamslabel, command = self.updateParams, text="Set")
327
                # ...as well as the number of constraints...
329         self.setWindow.otherFrame.numconslabel = LabelFrame(self.setWindow.otherFrame,
    text="No. of constraints")
            self.setWindow.otherFrame.numconslabel.ncons = Spinbox (self.setWindow.otherFrame.
    numconslabel, textvariable=self.ncons, from_=0, to=100, increment=1, width=10)
331         self.setWindow.otherFrame.numconslabel.setNconsBtn = Button (self.setWindow.
    otherFrame.numconslabel, command = self.updateCons, text="Set")

333             # ...and the number of target functions.
            self.setWindow.otherFrame.nTFlabel = LabelFrame(self.setWindow.otherFrame, text="
    No. of target functions")
335         self.setWindow.otherFrame.nTFlabel.nTFs = Spinbox (self.setWindow.otherFrame.
    nTFlabel, textvariable=self.nTF, from_=1, to=100, increment=1, width=10)
            self.setWindow.otherFrame.nTFlabel.setNTFBtn = Button (self.setWindow.otherFrame.
    nTFlabel, command = self.updateTFs, text="Set")
337
                # Chooser for optimizer (width will be set by the longest name of TF or
    optimizer)
339         self.setWindow.otherFrame.optilabel = LabelFrame(self.setWindow.otherFrame, text="
    Optimizer")
            self.setWindow.otherFrame.optilabel.menu = OptionMenu (self.setWindow.otherFrame.
    optilabel, self.optialgo, *self.optialgolist)
341         width = max(max([len(e) for e in self.imported_tfs]), max([len(e) for e in self.
    optialgolist]))
            self.setWindow.otherFrame.optilabel.menu["width"] = width
343
                # Chooser(s) for target function modules (width will be set as before)
345         self.setWindow.otherFrame.tflabel = LabelFrame(self.setWindow.otherFrame, text="
    Target functions")
            self.tfmenus = [(TFMenu (self.setWindow.otherFrame.tflabel, self.funcmodules[i], *
    self.imported_tfs)) for i in range(numTF)]
347         for e in self.tfmenus:
                e["width"] = width
349
                # Optional target value
```

```
351             self.setWindow.otherFrame.tvlabel = LabelFrame(self.setWindow.otherFrame, text="
    Targetted fitness value")
                self.setWindow.otherFrame.tvlabel.tv = Entry (self.setWindow.otherFrame.tvlabel,
    textvariable=self.tv, width=10)
353
                    # Finally add some buttons to save or cancel
355             self.setWindow.otherFrame.buttonFrame = Frame (self.setWindow.otherFrame)
                self.setWindow.otherFrame.buttonFrame.loadButton = Button (self.setWindow.
    otherFrame.buttonFrame, command=self.loadSettings, text="Load")
357             self.setWindow.otherFrame.buttonFrame.saveAsButton = Button (self.setWindow.
    otherFrame.buttonFrame, command=self.saveSettingsAs, text="Save as...")
                self.setWindow.otherFrame.buttonFrame.cancelButton = Button (self.setWindow.
    otherFrame.buttonFrame, command=self.setWindow.destroy, text="Cancel")
359             self.setWindow.otherFrame.buttonFrame.saveButton = Button (self.setWindow.
    otherFrame.buttonFrame, command=self.saveSettings, text="Save & Close")
                #self.setWindow.otherFrame.buttonFrame.testButton = Button (self.setWindow.
    otherFrame.buttonFrame, command=self.test, text="Test")
361

363             # Show widgets
                self.setWindow.paramFrame.grid(row=0, column=0, sticky=N+S) # parameters go left
365             self.setWindow.conFrame.grid(row=0, column=1, sticky=N+S) # parameters go left
                self.setWindow.otherFrame.grid(row=0, column=2, sticky=N+S) # the rest goes to the
     right
367             self.setWindow.otherFrame.numparamslabel.grid(column=0, row=0, sticky=E+W)
                self.setWindow.otherFrame.numparamslabel.numps.grid()
369             self.setWindow.otherFrame.numparamslabel.setNPBtn.grid(column=1, row=0)
                self.setWindow.otherFrame.numconslabel.grid(column=0, row=1, sticky=E+W)
371             self.setWindow.otherFrame.numconslabel.ncons.grid()
                self.setWindow.otherFrame.numconslabel.setNconsBtn.grid(column=1, row=0)
373             self.setWindow.otherFrame.nTFlabel.grid(column=0, row=2, sticky=E+W)
                self.setWindow.otherFrame.nTFlabel.nTFs.grid()
375             self.setWindow.otherFrame.nTFlabel.setNTFBtn.grid(column=1, row=0)
                self.setWindow.otherFrame.optilabel.grid(column=0, row=3, sticky=E+W)
377             self.setWindow.otherFrame.optilabel.menu.grid()
                self.setWindow.otherFrame.tflabel.grid(column=0, row=4, sticky=E+W)
379             self.setWindow.otherFrame.tvlabel.grid(column=0, row=5, sticky=E+W)
                self.setWindow.otherFrame.tvlabel.tv.grid()
381             self.setWindow.otherFrame.buttonFrame.grid(sticky=E+W)
                self.setWindow.otherFrame.buttonFrame.loadButton.grid(column=0, row=0)
383             self.setWindow.otherFrame.buttonFrame.saveAsButton.grid(column=1, row=0, sticky=E+
    W)
                self.setWindow.otherFrame.buttonFrame.cancelButton.grid(column=2, row=0)
385             self.setWindow.otherFrame.buttonFrame.saveButton.grid(column=1, row=1)
                #self.setWindow.otherFrame.buttonFrame.testButton.grid(column=2, row=1)
387
                # Event bindings
389             self.setWindow.bind("<Destroy>", self.__setWindowCloseHandler)

391             self.updateParams()
                self.updateCons()
393

395     #def test(self):
            #print self.params
397         #print [p.name.get() for p in self.params]
            #print self.constraints
399         #print [c.name.get() for c in self.constraints]


401
        def updateParams(self):
403         """Creates or deletes parameters when the number of parameters is changed."""

405         difference = self.nparams.get() - self.numparams
```

```
407         if difference > 0:
                self.params.extend([Parameter(self.setWindow.paramFrame.interior, parname="NEW",
    lbound=0, ubound=0) for i in range(difference)])
409         elif difference < 0:
                for i in range(difference, 0):
411                 self.params[i].destroy()            # remove widgets
                    #self.params.remove(self.params[i])  # remove parameter itself
413         self.numparams = self.nparams.get()         # update variable
            for param in self.params:
415             param.bind("<Destroy>", self.__paramDestroyHandler)


417
        def updateCons(self):
419         """Creates or deletes parameters when the number of parameters is changed."""

421         difference = self.ncons.get() - self.numcons

423         if difference > 0:
                self.constraints.extend([Constraint(self.setWindow.conFrame.interior, conname="NEW
    ", definition="42", lbound=0, ubound=0) for i in range(difference)])
425         elif difference < 0:
                for i in range(difference, 0):
427                 self.constraints[i].destroy()           # remove widgets
                    #self.constraints.remove(self.constraints[i])  # remove parameter itself
429         self.numcons = self.ncons.get()         # update variable
            for con in self.constraints:
431             con.bind("<Destroy>", self.__conDestroyHandler)


433
        def updateTFs(self):
435         """Creates or deletes target functions when the number of TFs is changed."""

437         difference = self.nTF.get() - len(self.tfmenus)

439         if difference > 0:
                self.funcmodules.extend([StringVar() for i in range(difference)])
441             self.tfmenus.extend([(TFMenu (self.setWindow.otherFrame.tflabel, self.funcmodules[
    len(self.tfmenus)+i], *self.imported_tfs)) for i in range(difference)])
            elif difference < 0:
443             for i in range(difference, 0):
                    self.tfmenus[i].destroy()                       # remove widgets
445                 self.tfmenus.remove(self.tfmenus[i])            # remove from tfmenu
                    self.funcmodules.remove(self.funcmodules[i])    # remove from list
447


449
        def __setWindowCloseHandler(self, event):
451         """Event handler for closing setWindow"""

453         # Reenable menu entry when window closes
            self.menu.setmenu.entryconfigure(0, state=NORMAL)
455


457
        def __paramDestroyHandler(self, event):
459         """Event handler for deletion of parameter (set and delete buttons)"""

461         #print event.widget.name.get()
            pnames = [p.name.get() for p in self.params]
463         i = pnames.index(event.widget.name.get())
            self.params.remove(self.params[i])
465         self.numparams = len(self.params)
            self.nparams.set(self.numparams)
467
```

```
469
    def __conDestroyHandler(self, event):
471     """Event handler for deletion of constraint (set and delete buttons)"""

473     #print event.widget.name.get()
        cnames = [c.name.get() for c in self.constraints]
475     i = cnames.index(event.widget.name.get())
        self.constraints.remove(self.constraints[i])
477     self.numcons = len(self.constraints)
        self.ncons.set(self.numcons)
479

481 def saveSettings(self):
        """Saves parameter settings to file and closes window."""
483
        # Enable 'run'-menu
485     self.menu.entryconfigure(2, state=NORMAL)

487     # write settings file
        self.writeParams("params.par")
489
        print "Settings saved."
491
        # close window
493     self.setWindow.destroy()

495
    def loadSettings(self):
497     """Read parameter settings from file and update Settings window accordingly.

499     """

501     filename = tkFileDialog.askopenfilename()

503     if not filename:
            filename = "params.par"
505
        # First parameters
507     try:
            optimodule, funcmodules_fromfile, np, parnames, lbounds, ubounds, res = readInput(
    filename, "params")
509     except:
            optimodule, funcmodules_fromfile, np, parnames, lbounds, ubounds, res =
    loadDefaults("params")
511
        self.nparams.set(np)          # update number of parameters
513     self.updateParams()

515     for i in range(self.numparams):
            self.params[i].name.set(parnames[i])
517         self.params[i].lower.set(lbounds[i])
            self.params[i].upper.set(ubounds[i])
519         self.params[i].res.set(res[i])

521     self.nTF.set(len(funcmodules_fromfile))
        self.updateTFs()
523
        for i in range(len(funcmodules_fromfile)):
525         if funcmodules_fromfile[i] in self.imported_tfs:
                self.funcmodules[i].set(funcmodules_fromfile[i])
527
        # Then constraints
529     try:
            connames, condefs, conlbounds, conubounds = readInput(filename, "constraints")
531     except:
```

```
            connames, condefs, conlbounds, conubounds = [], [], [], [] #loadDefaults("
    constraints")
533
        #self.numcons = len(connames)
535         self.ncons.set(len(connames))
        self.updateCons()
537
        for i in range(self.numcons):
539             self.constraints[i].name.set(connames[i])
            self.constraints[i].definition.set(condefs[i])
541             self.constraints[i].lower.set(conlbounds[i])
            self.constraints[i].upper.set(conubounds[i])
543
        try:
545             target_value = readInput(filename, "target")
        except:
547             target_value = -1e20

549         self.tv.set(target_value)

551     def saveSettingsAs(self):
        """Save parameter settings to a new file using Tk file dialog.
553
        """
555         filename = tkFileDialog.asksaveasfilename()
        if (filename):
557             self.writeParams(filename)
        else:
559             pass


561


563     def run(self):
        """Starts the optimization.
565
        """
567
        # Disable Run-button
569         self.menu.entryconfigure(2, state=DISABLED)

571         self.logfile = "optilog_" + datetime.now().isoformat()[:-7] # Create new logfile

573         optimodule = self.optialgo.get()
        funcmodules = [e.get() for e in self.funcmodules]
575
        # x variables are not used
577         x1, x2, x3, x4, lbounds, ubounds, x5 = readInput("params.par", "params")

579         bounds = {"lower":lbounds, "upper":ubounds}

581         settings = parse_config("optimizers/optisettings")

583         x5, x6, x7, x8 = readInput("params.par", "constraints")

585         print "\n----------------------------------------"
        print str(time.asctime(time.localtime()))
587         print "----------------------------------------"
        print "# Optimizer"
589         print optimodule
        print "# TF module"
591         print ' '.join([e for e in funcmodules])
        print "# No. of params"
593         print x3
        print "# Target value"
595         print self.tv.get()
```

```
        print "# Parameter names"
597     print ' '.join(x4)
        print "# Lower limits"
599     print ' '.join([str(e) for e in lbounds])
        print "# Upper limits"
601     print ' '.join([str(e) for e in ubounds])
        print "# Constraints"
603     print ' '.join([e for e in x5])
        print ' '.join([e for e in x6])
605     print ' '.join([str(e) for e in x7])
        print ' '.join([str(e) for e in x8])
607     print "-----------------------------------------"
        print "Running", optimodule, "..."
609     print "-----------------------------------------"
        #self.textout.update()
611


613


615     optimizer = getattr(getattr(optimizers, optimodule), optimodule)
        optimizer(funcmodules, bounds, settings)
617
        # Re-enable Run-button
619     self.menu.entryconfigure(2, state=NORMAL)


621
    def runrun(self):
623     """Helper function which runs run(self) as thread to keep GUI responsive

625     """

627     thrd = threading.Thread(target=self.run)
        thrd.start()
629


631 def showsettingshelp(self):
        """Opens help window showing info on the settings.
633
        """
635     # find out if window is already there
        toplevellist = [elem for elem in self.winfo_children() if elem.winfo_class()=="
    Toplevel" and elem.title() == "Help - Settings"]
637
        if (toplevellist):
639         self.settingshelpWindow.deiconify()
        else:
641         text = "Parameters:\n\n" \
            + "\tParameter Name\n" \
643         + "\tUpper Limit\n" \
            + "\tLower Limit\n" \
645         + "\tResolution (search width)\n\n" \
            + "There is no check if parameter names, limits and number of parameters make
    sense when used " \
647         + "with a certain target function. Best set up an example file for each target " \
            + "function.\n" \
649         + "\n" \
            + "Constraints:\n\n" \
651         + "\tConstraint Name\n" \
            + "\tConstraint Definition\n" \
653         + "\tUpper Limit\n" \
            + "\tLower Limit\n\n" \
655         + "Constraints are defined as equations in which the variables can be any
    parameter " \
            + "defined under 'Parameters'. \n" \
657         + "\n" \
```

```
            + "Targetted fitness value: \n\n" \
659         + "If you want to optimize towards a certain value instead of the lowest one. Not
    for MOOP! To disable, set to -1e20."

661
            self.settingshelpWindow = Toplevel (self)
663         self.settingshelpWindow.title("Help - Settings")
            self.settingshelpWindow.rowconfigure(0, weight=1)
665         self.settingshelpWindow.columnconfigure(0, weight=1)

667         self.settingshelpWindow.settingshelpLabel = Label(self.settingshelpWindow, text =
    text, justify=LEFT, padx=10, wraplength=250, relief=SUNKEN)
            self.settingshelpWindow.closeButton = Button(self.settingshelpWindow, text="Close"
    , command=self.settingshelpWindow.destroy)
669
            self.settingshelpWindow.settingshelpLabel.grid(sticky=N+S+E+W)
671         self.settingshelpWindow.closeButton.grid(sticky=S)

673


675     def showtfhelp(self):
            """Opens help window showing info on imported target functions.
677
            """
679
            # find out if window is already there
681         toplevellist = [elem for elem in self.winfo_children() if elem.winfo_class()=="
    Toplevel" and elem.title() == "Target Functions"]

683         if (toplevellist):
                self.tfhelpWindow.deiconify()
685         else:
                tfdocstrings = [getattr(targetfunctions, e).__doc__ for e in self.imported_tfs]
687             text = "Name: Description\n\n"
                for i in range(len(tfdocstrings)):
689                 if tfdocstrings[i] == None:
                        text += self.imported_tfs[i] + ":    " + "no doc string found" + "\n"
691                 else:
                        text += self.imported_tfs[i] + ":    " + tfdocstrings[i] + "\n"
693
                self.tfhelpWindow = Toplevel (self)
695             self.tfhelpWindow.title("Target Functions")
                self.tfhelpWindow.rowconfigure(0, weight=1)
697             self.tfhelpWindow.columnconfigure(0, weight=1)

699             self.tfhelpWindow.tfhelpLabel = Label(self.tfhelpWindow, text = text, justify=LEFT
    , padx=10, wraplength=1000, relief=SUNKEN)
                self.tfhelpWindow.closeButton = Button(self.tfhelpWindow, text="Close", command=
    self.tfhelpWindow.destroy)
701
                self.tfhelpWindow.tfhelpLabel.grid(sticky=N+S+E+W)
703             self.tfhelpWindow.closeButton.grid(sticky=S)

705
    def about(self):
707     """Opens 'about' window."""

709     text = "Funky Optimizer\nWritten 2011-2013 by Erik Braun <braun@kit.edu>"
        tkMessageBox.showinfo("About this program", text)
711


713 def showlicense(self):
        """Opens a window showing GPL info."""
715
        # find out if window is already there
```

```
717         toplevellist = [elem for elem in self.winfo_children() if elem.winfo_class()=="
    Toplevel" and elem.title() == "License"]

719         if (toplevellist):
                self.licWindow.deiconify()
721         else:
                text = "        'Funky Optimizer' optimizin ur stuff!\n\
723             Written 2011-2014 by Erik Braun\n\
                \n\
725             This program is free software: you can redistribute it and/or modify\n\
                it under the terms of the GNU General Public License as published by\n\
727             the Free Software Foundation, either version 3 of the License, or\n\
                (at your option) any later version.\n\
729             \n\
                This program is distributed in the hope that it will be useful,\n\
731             but WITHOUT ANY WARRANTY; without even the implied warranty of\n\
                MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n\
733             GNU General Public License for more details.\n\
                \n\
735             You should have received a copy of the GNU General Public License\n\
                along with this program.  If not, see <http://www.gnu.org/licenses/>.\n\
737             \n\
                Author contact: Erik Braun <braun@kit.edu>"
739
                self.licWindow = Toplevel (self)
741             self.licWindow.title("License")

743             self.licWindow.licLabel = Label(self.licWindow, text = text, justify=CENTER, padx
    =10, relief=SUNKEN)
                self.licWindow.closeButton = Button(self.licWindow, text="Close", command=self.
    licWindow.destroy)
745
                self.licWindow.licLabel.grid()
747             self.licWindow.closeButton.grid()


749
    def writeParams(self, filename):
751
        # write settings file
753     out = open(filename, "w")

755     out.write("# Parameter file for optimization script\n")
        out.write("# " + str(time.asctime(time.localtime())) + "\n")
757     out.write("\n")
        out.write("# optimizer module \n")
759     out.write(self.optialgo.get() + "\n")
        out.write("\n")
761     out.write("# target function modules \n")
        for i in range(self.nTF.get()):
763         out.write(self.funcmodules[i].get() + " "),
        out.write("\n\n")
765     out.write("# number of parameters\n")
        out.write(str(self.nparams.get()) + "\n")
767     out.write("\n")
        out.write("# targetted fitness value\n")
769     out.write(str(self.tv.get()) + "\n")
        out.write("\n")
771     out.write("# parameters\n")
        for elem in self.params:
773         out.write(elem.name.get() + "\t")
        out.write("\n")
775     for elem in self.params:
            out.write(str(elem.lower.get()) + "\t")
777     out.write("\n")
        for elem in self.params:
```

```
779             out.write(str(elem.upper.get()) + "\t")
        out.write("\n")
781         for elem in self.params:
            out.write(str(elem.res.get()) + "\t")
783         out.write("\n\n")
        out.write("# constraints\n")
785         for elem in self.constraints:
            out.write(elem.name.get() + "\t")
787         out.write("\n")
        for elem in self.constraints:
789             out.write(elem.definition.get() + "\t")
        out.write("\n")
791         for elem in self.constraints:
            out.write(str(elem.lower.get()) + "\t")
793         out.write("\n")
        for elem in self.constraints:
795             out.write(str(elem.upper.get()) + "\t")
        out.write("\n\n")
797


799         out.close

801
def stringToNumber(string):
803     """Returns int or float if possible."""
    try:
805         return int(string)
    except ValueError:
807         try:
            return float(string)
809         except ValueError:
            return string
811


813 def loadDefaults(what):
    """Returns standard values for parameters and constraints if needed"""
815
    if (what == "params"):
817         #optimodule = "de"
        #funcmodules_fromfile = ["ann"]
819         #self.numparams = 1
        #parnames = ["param0"]
821         #lbounds = [-1]
        #ubounds = [1]
823         #res = [1]
        return "de", ["ann"], 1, ["param0"], [-1], [1], [1]
825
    elif (what == "constraints"):
827         #connames = ["constraint0"]
        #condefs = ["param0"]
829         #conlbounds = [-1]
        #conubounds = [1]
831         return ["constraint0"], ["param0"], [-1], [1]
        #return [], [], [], []
833     else:
        return -1.0e20
835


837 def parse_config(filename):
    """parses optimizer settings file
839
    syntax: key = value
841     comments (#) allowed
    """
843     options = {}
```

```
    try:
845         f = open(filename, "r")
    except IOError, e:
847         if e.errno == 13:
                raise InputFileNotReadableError, "File '%s' not readable." % filename
849         elif e.errno == 2:
                raise InputFileNotFoundError, "File '%s' not found." % filename
851
    for line in f:
853         # First, remove comments:
            if '#' in line:
855             # split on comment char, keep only the part before
                line, comment = line.split('#', 1)
857         # Second, find lines with an option=value:
            if '=' in line:
859             # split on option char:
                option, value = line.split('=', 1)
861             # strip spaces:
                option = option.strip()
863             value = stringToNumber(value.strip())
                # store in dictionary:
865             options[option] = value
    f.close()
867 return options


869
def readInput(file, what="params"):
871     """Reads and parses input file.

873     Comments allowed (#), no inline comments
        Empty lines allowed
875     Spaces allowed

877     File structure:

879     1 Optimizer module name (string)
        2 Target function module names (strings, seperated by blanks)
881     3 Number of parameters (integer)
        4 Target value (float)
883     5 Parameter names (strings)
        6 Lower boundaries of parameters (floats or integers, seperated by blanks)
885     7 Upper boundaries of parameters (floats or integers, seperated by blanks)
        8 Resolution (steps) of parameters (integers, seperated by blanks)
887     9 Constraint names (strings)
        10 Definition of constraints using parameters (strings)
889     11 Lower boundaries of constraints (floats or integers, seperated by blanks)
        12 Upper boundaries of constraints (floats or integers, seperated by blanks)
891     """

893     # open input file
        try:
895         input = open(file, "r")
    except IOError, e:
897         if e.errno == 13:
                raise InputFileNotReadableError, "File '%s' not readable." % file
899         elif e.errno == 2:
                raise InputFileNotFoundError, "File '%s' not found." % file
901     inputs = []
        i = 0
903
        # parse input file
905     for line in input:
            if not line.strip():
907             continue
            if re.compile('^#').search(line) is not None:
```

```
            continue
        else:
            inputs[i:1] = [line.strip()]
            i+=1
        if (i in [1,2,5,9]) and line.strip().isdigit():                    # Error if numbers
 on lines 1, 2, 4 or 7,8 TODO: Auskommentierte Zeilen mitzaehlen.
            raise InputParamsWrongTypeError, "Wrong data type in input file, line %i." % i
        elif i == 3 and not line.strip().isdigit():                         # Error if
strings on line 3 or 11
            raise InputParamsWrongTypeError, "Wrong data type in input file, line %i." % i
        elif (i in [4,6,7,8,11,12]):                                          # Check param
values for chars
            line = line.strip().split()                                     # split line in
values
            for elem in line:
                if elem.isalpha():
                    raise InputParamsWrongTypeError, "Wrong data type in input file, line %i."
 % i

    # assign variables
    inputs = [elem for elem in inputs]
    optimodule = inputs[0].split()[0]
    funcmodules = inputs[1].split()
    numparams = int(inputs[2])
    try:
        targetvalue = inputs[3]
    except IndexError:
        targetvalue = -1.0e20
    parnames = inputs[4].split()
    lbounds = [stringToNumber(elem) for elem in inputs[5].split()]
    ubounds = [stringToNumber(elem) for elem in inputs[6].split()]
    resolutions = [stringToNumber(elem) for elem in inputs[7].split()]

    if (len(inputs) > 8):
        connames = inputs[8].split()
        condefs = inputs[9].split()
        conlbounds = [stringToNumber(elem) for elem in inputs[10].split()]
        conubounds = [stringToNumber(elem) for elem in inputs[11].split()]



    # some further checks
    if len(lbounds) != numparams:
        raise InputParamsMismatchError, "No. of lower boundaries does not match no. of params.
"
    if len(ubounds) != numparams:
        raise InputParamsMismatchError, "No. of upper boundaries does not match no. of params.
"
    if len(resolutions) != numparams:
        raise InputParamsMismatchError, "No. of resolutions does not match no. of params."

    if (what == "params"):
        return optimodule, funcmodules, numparams, parnames, lbounds, ubounds, resolutions
    elif (what == "constraints" and len(inputs)>8):
        return connames, condefs, conlbounds, conubounds
    elif (what == "target"):
        return float(targetvalue)
    elif (what == "resolutions"):
        return resolutions
    else:
        return 42
```

```
967
if __name__ == '__main__':
969     from os import uname
        app = Application()
971     title = "Funky Optimizer (" + uname()[1] + ")"
        app.master.title(title)
973     app.mainloop()
```

Listing A.1: Quelltext der Optimierungsumgebung: *TkInterOpti.py*

A.2 Zielfunktionen targetfunctions.py

```
1

3 """Target functions for optimization"""

5     #This file is part of Funky Optimizer.

7     #Funky Optimizer is free software: you can redistribute it and/or modify
      #it under the terms of the GNU General Public License as published by
9     #the Free Software Foundation, either version 3 of the License, or
      #(at your option) any later version.
11
      #Funky Optimizer is distributed in the hope that it will be useful,
13    #but WITHOUT ANY WARRANTY; without even the implied warranty of
      #MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
15    #GNU General Public License for more details.

17    #You should have received a copy of the GNU General Public License
      #along with Funky Optimizer.  If not, see <http://www.gnu.org/licenses/>.
19


21

23 import sys
   from parse_constraint import *
25 from ann_validity_check import *
   from TkinterOpti import readInput
27 sys.path.append("targetfunctions")

29 from cfd2 import target_function as cfd2
   from time import time
31
   TIMING = True
33
   # ------------------------------------------------------------------------
35 # Helper functions
   # ------------------------------------------------------------------------
37
   try:
39     target_value = readInput("params.par", "target")
   except:
41     target_value = -1e20

43 try:
       resolutions = readInput("params.par", "resolutions")
45 except:
       resolutions = [2e-13]*1000
47
   for i in range(len(resolutions)):
49     if resolutions[i] == 0.:
```

```
        resolutions[i] = 2e-13

def distance2target(trial_value, target_value):
    """Calculates distance between trial value and target value"""

    if target_value != -1e20:
        return abs(trial_value - target_value)
    else:
        return trial_value


def gettflist():

    tflist = []
    tflist.append("rosenbrock")
    tflist.append("de_jong")
    tflist.append("rastrigin")
    tflist.append("easom")
    tflist.append("zdt1_1")
    tflist.append("zdt1_2")
    tflist.append("zdt2_2")
    tflist.append("zdt3_2")
    tflist.append("doerr_durchblick_1985")
    tflist.append("martin_1967")
    tflist.append("verschleiss")
    tflist.append("cfd_all")
    tflist.append("cfd_unworn")
    tflist.append("cfd_worn_NwFix")
    tflist.append("cfd_worn_NwVar")
    tflist.append("ann_DB")
    tflist.append("ann")
    tflist.append("ann_stepped")
    tflist.append("GP_DB")
    return tflist

def stringToNumber(string):
    """Returns int or float if possible."""
    try:
        return int(string)
    except ValueError:
        try:
            return float(string)
        except ValueError:
            return string


def searcharchive(filename, calc, m):
    """Returns trial_value from archive or calculated by calc"""

    if filename:                                        # archive is used
        archive = open(filename, "a+")
        firstline = archive.readline().split()

        if (len(firstline)==len(m)):                    # check if archive is valid for this
    number of parameters
            match = False
            ind = [line.split() for line in archive]
            individuals = [[stringToNumber(f) for f in e] for e in ind] # build list of
    individuals

            for individual in individuals:
                n = 0
                if individual:
                    for i in range(len(m)):             # look for match
```

```
                        if (m[i] < (individual[i] + resolutions[i]/2.) and m[i] > (individual[
    i] - resolutions[i]/2.)):
113                         n += 1
                    if n == (len(m)):                   # match found
115                     trial_value = individual[-1]    # use old value
                        match = True
117                     print "Using archived value."
                        archive.close()
119                     break

121             if not match:
                    trial_value = calc(m)               # calculate new value and write to
    archive
123                 if trial_value != 1.e20:
                        for e in m:
125                         archive.write(str(e) + "\t")
                        archive.write(str(trial_value) + "\n")
127                     archive.close()

129         else:
                archive.close()
131             trial_value = calc(m)       # calculate new value if number of parameters do not
    match

133     else:
            trial_value = calc(m)           # calculate new value if archive is not used
135
        return trial_value
137


139 # ------------------------------------------------------------------------
    # Test functions
141 # ------------------------------------------------------------------------

143 def rosenbrock(m, *args):
        """Rosenbrock valley function (2D only)"""
145     trial_value = (1. - m[0])**2 + 100.*(m[1] - m[0]**2)**2
        return distance2target(trial_value, target_value)
147


149 def de_jong(m, *args):
        """Multidimensional parabola"""
151
        def calc(m):
153         t = 0.
            for x in m:
155             t = t + x**2
            return t
157
        trial_value = searcharchive("archivtest", calc, m)
159
        return distance2target(trial_value, target_value)
161


163 def rastrigin(m, *args):
        """Rastrigin's multimodal test function"""
165     import numpy as np
        pi = 2.*np.arcsin(1.0)
167     trial_value = 10.*len(m)
        for i in m:
169         trial_value = trial_value + (i**2 - 10*np.cos(2*pi*i))
        #return trial_value
171     return distance2target(trial_value, target_value)


173
```

```
def easom(m, *args):
    """Flat plane with hole"""
    import numpy as np
    pi = 2.*np.arcsin(1.0)
    trial_value = -np.cos(m[0])*np.cos(m[1])*np.exp(-(m[0] - pi)**2 - (m[1] - pi)**2)
    #return trial_value
    return distance2target(trial_value, target_value)


def zdt1_1(m, *args):
    """f1 of ZDT1, ZDT2 and ZDT3 problem"""
    return m[0]


def zdt1_2(m, *args):
    """f2 of ZDT1 problem, second argument must provide return value of zdt1_1"""
    import numpy as np

    n = len(m)
    g = 1 + 9/(n-1) * sum(m[1:])
    h = 1 - np.sqrt(args[0]/g)

    return g*h


def zdt2_2(m, *args):
    """f2 of ZDT2 problem, second argument must provide return value of zdt1_1"""

    n = len(m)
    g = 1 + 9/(n-1) * sum(m[1:])
    h = 1 - (args[0]/g)**2

    return g*h


def zdt3_2(m, *args):
    """f2 of ZDT3 problem, second argument must provide return value of zdt1_1"""
    import numpy as np
    pi = 2.*np.arcsin(1.0)

    n = len(m)
    g = 1 + 9/(n-1) * sum(m[1:])
    h = 1 - np.sqrt(args[0]/g) - (args[0]/g)*np.sin(10*pi*args[0])

    return g*h









# ------------------------------------------------------------------------
# Useful target functions for labyrinth seals
# ------------------------------------------------------------------------



def ann_DB(m, *args):
    """ANN function, whole parameter set, straight-through seals

    Reverts to CFD method if individual is not in known range of ANN.
    """
```

```
    # m[0]: Sh
    # m[1]: Sb
    # m[2]: s
    # m[3]: t
    # m[4]: gamma
    # m[5]: Pi

    if TIMING: t1 = time()

    if (not constraint_check(m)):
        params = []
        params.append(int(m[0]))# n
        params.append(m[1])     # Ri
        params.append(m[2])     # s
        params.append(m[3])     # Sb
        params.append(m[4])     # Sh
        params.append(m[5])     # t
        params.append(m[6])     # gamma
        params.append(m[7])     # theta
        params.append(m[8])     # STh
        params.append(m[9])     # STs
        params.append(m[10])    # Nh
        params.append(m[11])    # Nw
        params.append(m[12])    # Ns
        params.append(m[13])    # Pi
        params.append(m[14])    # p_sz
        params.append(m[15])    # R (an der Spitze)
        params.append(m[16])    # u_w

        # try ANN
        sys.path.append("targetfunctions/ANN/")
        import ann_functions as ann_functions

        try:
            trial_value_ann = ann_functions.ann_test(parameter_list=[m[4],m[3],m[2],m[5],m[6],
    m[13]], path="targetfunctions/ANN/Netze", train_data_name="Lerndatensatz_Homogen",
    neurons_hidden_1=10, iteration=2000)

            if TIMING:
                t2 = time()
                print "Time (sec): ", (t2-t1)

            return distance2target(trial_value_ann, target_value)
        except Exception, e:
            print "Reverting to CFD method."
            trial_value_cfd = searcharchive("cfd-archiv", cfd2, params)
            return distance2target(trial_value_cfd, target_value)


def ann(m, *args):
    """ANN function, whole parameter set

    Reverts to CFD method if individual is not in known range of ANN.
    """

    if TIMING: t1 = time()

    if (not constraint_check(m)):
        params = []
        params.append(int(m[0]))# n
        params.append(m[1])     # Ri
        params.append(m[2])     # s
        params.append(m[3])     # Sb
        params.append(m[4])     # Sh
        params.append(m[5])     # t
```

```
        params.append(m[6])      # gamma
303     params.append(m[7])      # theta
        params.append(m[8])      # STh
305     params.append(m[9])      # STs
        params.append(m[10])     # Nh
307     params.append(m[11])     # Nw
        params.append(m[12])     # Ns
309     params.append(m[13])     # Pi
        params.append(m[14])     # p_sz
311     params.append(m[15])     # R (an der Spitze)
        params.append(m[16])     # f
313     params.append(m[17])     # HCd
        params.append(m[18])     # HCw
315     params.append(m[19])     # HCh
        params.append(m[20])     # T_t0
317

319     # try ANN
        sys.path.append("targetfunctions/ANN/")
321     import ann_functions as ann_functions

323     u_w = m[1]*3.14159*2*m[16]
        try:                                                    # f      Ri    u_w    STh   STs
      HCd   HCw   HCh   Nh    Nw    Ns
325         trial_value_ann = ann_functions.ann_test(parameter_list=[m[16],m[1],u_w   ,m[8],m
    [9],m[17],m[18],m[19],m[10],m[11],m[12],
                                                                # n     Sh    Sb    s     t
    R     theta gamma T_t0  p_sz  Pi
327                                                               m[0],m[4],m[3],m[2],m[5],
    m[15],m[7], m[6], m[20],m[14],m[13]],
                                                        path="targetfunctions/ANN/Netze",
329                                                     train_data_name="daten.fann",
                                                        neurons_hidden_1=42,
331                                                     iteration=3000)

333         if TIMING:
                t2 = time()
335             print "Time (sec): ", (t2-t1)

337         return distance2target(trial_value_ann, target_value)
        except Exception, e:
339         print "Reverting to CFD method.", e
            trial_value_cfd = searcharchive("cfd-archiv", cfd2, params)
341         return distance2target(trial_value_cfd, target_value)

343 def ann_stepped(m, *args):
    """ANN function, whole parameter set for stepped seals w/o HC or grooves
345
    Reverts to CFD method if individual is not in known range of ANN.
347 """

349 if TIMING: t1 = time()

351 if (not constraint_check(m)):
        params = []
353     params.append(int(m[0]))# n
        params.append(m[1])      # Ri
355     params.append(m[2])      # s
        params.append(m[3])      # Sb
357     params.append(m[4])      # Sh
        params.append(m[5])      # t
359     params.append(m[6])      # gamma
        params.append(m[7])      # theta
361     params.append(m[8])      # STh
        params.append(m[9])      # STs
```

```
363         params.append(m[10])    # Nh
            params.append(m[11])    # Nw
365         params.append(m[12])    # Ns
            params.append(m[13])    # Pi
367         params.append(m[14])    # p_sz
            params.append(m[15])    # R (an der Spitze)
369         params.append(m[16])    # f
            params.append(m[17])    # HCd
371         params.append(m[18])    # HCw
            params.append(m[19])    # HCh
373         params.append(m[20])    # T_t0


375
            # try ANN
377         sys.path.append("targetfunctions/ANN/")
            import ann_functions as ann_functions
379
            u_w = m[15]*3.14159*2*m[16]
381         try:                                                    # s    Sb
                trial_value_ann = ann_functions.ann_test(parameter_list=[m[2],m[3],
383                                                                 # Sh    t     gamma STh   STs
      Pi
                                                                    m[4],m[5],m[6], m[8],m
    [9], m[13]],
385                                                     path="targetfunctions/ANN/Netze",
                                                        train_data_name="konvergent_train",
387                                                     neurons_hidden_1=50,
                                                        iteration=2000)
389         except Exception, e:
                print "Reverting to CFD method.", e
391             trial_value_cfd = searcharchive("cfd-archiv", cfd2, params)
                return distance2target(trial_value_cfd, target_value)
393
            if not trial_value_ann:
395             print "Reverting to CFD method."
                trial_value_cfd = searcharchive("cfd-archiv", cfd2, params)
397             return distance2target(trial_value_cfd, target_value)
            else:
399
                if TIMING:
401                 t2 = time()
                    print "Time (sec): ", (t2-t1)
403
                return distance2target(trial_value_ann, target_value)
405


407 def doerr_durchblick_1985(m, *args):
    """Doerr's correlation for straight-through labyrinth seals"""
409     if TIMING: t1 = time()
        if (not constraint_check(m)):
411         import numpy as np
            st = m[2]/m[3]
413         Pi = m[0]
            n = m[1]
415         cd1 = 1-np.exp(-(-10*st+10.4)*(Pi-(-0.778*st+0.873)))
            cd2=0.179*Pi+(0.354*n**2-4.49*n+8.02)*st**2+(-0.118*n**2+1.622*n-2.876)*st+(0.01276*n
    **2-0.216*n+1.03)
417         cd3=1-np.exp(0.544*Pi-2.8)
            cd4=1+(-0.0975+4.907*st-0.00783*n)*((Pi-1)**(1.754*st+0.0167*n))*np.exp((-0.37-97.3*st
    +0.458*n)*(Pi-1))
419         trial_value = cd1*cd2*cd3*cd4
            #return trial_value
421         if TIMING:
                t2 = time()
423             print "Time (sec): ", (t2-t1)
```

```
            return distance2target(trial_value, target_value)
        else:
            return 1e20



def martin_1967(m, *args):
    """Martin's correlation for straight-through labyrinth seals"""
    if TIMING: t1 = time()
    if (not constraint_check(m)):
        import numpy as np
        sa = m[2]
        ta = m[3]
        Pi = m[0]
        z = m[1]
        p0 = Pi*1e5
        pz = 1e5
        T0 = 300.
        rho0 = p0/(287.*T0)
        area = 1.
        A = 0.0620
        B = 0.00678
        C = -0.000652
        kappa = 1.4

        mpunkt = np.sqrt(sa/ta)*area*np.sqrt((p0**2.-pz**2.)*rho0/p0)/(np.sqrt(z)*np.sqrt(A+np
    .sqrt(B*sa+C)))
        #print mpunkt
        Q_id = np.sqrt(2.*kappa/(287.*(kappa-1.))*(1.-(1./Pi)**((kappa-1.)/kappa)))*(1./Pi)
    **(1./kappa)
        mpunkt_id = p0*area*Q_id/np.sqrt(T0)
        #print mpunkt_id
        trial_value = mpunkt/mpunkt_id
        #return trial_value
        if TIMING:
            t2 = time()
            print "Time (sec): ", (t2-t1)
        return distance2target(trial_value, target_value)
    else:
        return 1e20


def cfd_all(m, *args):
    """CFD with FoamBatch - all parameters"""

    if TIMING: t1 = time()

    if (not constraint_check(m)):
        params = []
        params.append(int(m[0]))# n
        params.append(m[1])     # Ri
        params.append(m[2])     # s
        params.append(m[3])     # Sb
        params.append(m[4])     # Sh
        params.append(m[5])     # t
        params.append(m[6])     # gamma
        params.append(m[7])     # theta
        params.append(m[8])     # STh
        params.append(m[9])     # STs
        params.append(m[10])    # Nh
        params.append(m[11])    # Nw
        params.append(m[12])    # Ns
        params.append(m[13])    # Pi
        params.append(m[14])    # p_sz
        params.append(m[15])    # R (an der Spitze)
```

```
487             params.append(m[16])    # f
                params.append(m[17])    # HCh
489             params.append(m[18])    # HCd
                params.append(m[19])    # HCw
491             params.append(m[20])    # Tt0

493             trial_value = searcharchive("cfd-archiv", cfd2, params)

495             if TIMING:
                    t2 = time()
497                 print "Time (sec): ", (t2-t1)

499             return distance2target(trial_value, target_value)
        else:
501             return 1e20


503
    def cfd_unworn(m, *args):
505     """CFD with FoamBatch - unworn: no grooves, no tip radius"""

507     if TIMING: t1 = time()

509     if (not constraint_check(m)):
            params = []
511             params.append(int(m[0]))# n
                params.append(m[1])     # Ri
513             params.append(m[2])     # s
                params.append(m[3])     # Sb
515             params.append(m[4])     # Sh
                params.append(m[5])     # t
517             params.append(m[6])     # gamma
                params.append(m[7])     # theta
519             params.append(m[8])     # STh
                params.append(m[9])     # STs
521             params.append(0.0)      # Nh
                params.append(0.0)      # Nw
523             params.append(0.0)      # Ns
                params.append(m[13])    # Pi
525             params.append(m[14])    # p_sz
                params.append(0.0)      # R (an der Spitze)
527             params.append(m[16])    # f
                params.append(m[17])    # HCh
529             params.append(m[18])    # HCd
                params.append(m[19])    # HCw
531             params.append(m[20])    # Tt0

533             trial_value = searcharchive("cfd-archiv", cfd2, params)

535             if TIMING:
                    t2 = time()
537                 print "Time (sec): ", (t2-t1)

539             return distance2target(trial_value, target_value)
        else:
541             return 1e20


543

545 def cfd_worn_NwVar(m, *args):
    """
547     CFD with FoamBatch - worn tips: 0,1mm tip radius, groove width Sb+2*1mm, groove depth
        configurable
        Throat area calculated as gap width times depth (b*s) for better comparison
549
    """
```

```
551
    if TIMING: t1 = time()
553
    if (not constraint_check(m)):
555     params = []
        params.append(int(m[0]))# n
557     params.append(m[1])     # Ri
        params.append(m[2])     # s
559     params.append(m[3])     # Sb
        params.append(m[4])     # Sh
561     params.append(m[5])     # t
        params.append(m[6])     # gamma
563     params.append(m[7])     # theta
        params.append(m[8])     # STh
565     params.append(m[9])     # STs
        params.append(m[10])    # Nh
567     params.append(m[3]+2*0.001)    # Nw = Sb + 2*1 mm
        params.append(m[12])    # Ns
569     params.append(m[13])    # Pi
        params.append(m[14])    # p_sz
571     params.append(0.0001)   # R (an der Spitze)
        params.append(m[16])      # f
573     params.append(m[17])    # HCh
        params.append(m[18])    # HCd
575     params.append(m[19])    # HCw
        params.append(m[20])    # Tt0
577
        trial_value = searcharchive("cfd-archiv", cfd2, params)
579
        if TIMING:
581         t2 = time()
            print "Time (sec): ", (t2-t1)
583
        # correct cD calculated with Nh+s to cD calculated only with s
585     return distance2target(trial_value*(1+m[10]/m[2]), target_value)
    else:
587     return 1e20


589


591 def cfd_worn_NwFix(m, *args):
    """
593 CFD with FoamBatch - worn tips: 0,1mm tip radius, groove width and groove depth
    configurable
    Throat area calculated as gap width times depth (b*s) for better comparison
595
    """
597
    if TIMING: t1 = time()
599
    if (not constraint_check(m)):
601     params = []
        params.append(int(m[0]))# n
603     params.append(m[1])     # Ri
        params.append(m[2])     # s
605     params.append(m[3])     # Sb
        params.append(m[4])     # Sh
607     params.append(m[5])     # t
        params.append(m[6])     # gamma
609     params.append(m[7])     # theta
        params.append(m[8])     # STh
611     params.append(m[9])     # STs
        params.append(m[10])    # Nh
613     params.append(m[11])    # Nw
        params.append(m[12])    # Ns
```

```
615         params.append(m[13])    # Pi
            params.append(m[14])    # p_sz
617         params.append(0.0001)   # R (an der Spitze)
            params.append(0.0)      # u_w
619         params.append(m[17])    # HCh
            params.append(m[18])    # HCd
621         params.append(m[19])    # HCw
            params.append(m[20])    # Tto
623
            trial_value = searcharchive("cfd-archiv", cfd2, params)
625
            if TIMING:
627             t2 = time()
                print "Time (sec): ", (t2-t1)
629
            # correct cD calculated with Nh+s to cD calculated only with s
631         return distance2target(trial_value*(1+m[10]/m[2]), target_value)
        else:
633         return 1e20


635


637 def GP_DB(m, *args):
        """Gaussian process using Scikit package, trained on straight-through laby data from CFD
639
        Reverts to CFD method if Sigma is too large.
641     """

643     if TIMING: t1 = time()

645     sys.path.append("targetfunctions/GP/")
        from sklearn.gaussian_process import GaussianProcess
647     from sklearn.externals import joblib
        import numpy as np
649     sigma_max = 0.01    # maximum uncertainty of gaussian process

651     if (not constraint_check(m)):

653         e = 0
            if not int(m[0]) == 3:  # n
655             e += 1
            if not m[7] == 15:      # theta
657             e += 1
            if not m[8] == 0.:      # STh
659             e += 1
            if not m[10] == 0.:     # Nh
661             e += 1
            if not m[14] == 102000.:    # p_sz
663             e += 1
            if not m[15] == 0.:     # R (an der Spitze)
665             e += 1
            if not m[16] == 0.:     # f
667             e += 1
            if not m[17] == 0.:     # HCh
669             e += 1
            if not m[21] == 300.:   # T_t0
671             e += 1

673         if e > 0:
                print "Parameter(s) out of range. Reverting to CFD method.", e
675             trial_value_cfd = searcharchive("cfd-archiv", cfd2, m)
                return distance2target(trial_value_cfd, target_value)
677

679         # try GP
```

```
            # s      Sb     Sh     t      gamma  Pi
        ind = [m[2], m[3], m[4], m[5], m[6], m[13]]

        try:
            # Load existing GP
            gp = joblib.load('targetfunctions/GP/GP_DB-Labys.pkl')

            trial_value_gp, MSE = gp.predict(ind, eval_MSE=True)
            sigma = np.sqrt(MSE)
            print "Sigma:", sigma[0]

        except Exception, ex:
            print "Reverting to CFD method.", ex
            trial_value_cfd = searcharchive("cfd-archiv", cfd2, m)
            return distance2target(trial_value_cfd, target_value)

        if sigma <= sigma_max:

            if TIMING:
                t2 = time()
                print "Time (sec): ", (t2-t1)

            return distance2target(trial_value_gp, target_value)
        else:
            print "Sigma too large. Reverting to CFD method."
            trial_value_cfd = searcharchive("cfd-archiv", cfd2, m)
            return distance2target(trial_value_cfd, target_value)

    else:
        return 1e20
```

Listing A.2: Quelltext der Zielfunktionen: *targetfunctions.py*