Deutsches Zentrum
für Luft- und Raumfahrt e.V.

Dissertation

Multifidelity-Optimierungsverfahren für Turbomaschinen

Andreas Schmitz

Institut für Antriebstechnik
Köln

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Multifidelity-Optimierungsverfahren für Turbomaschinen

Andreas Schmitz

Institut für Antriebstechnik
Köln

207 Seiten
37 Bilder
6 Tabellen
197 Literaturstellen

Multifidelity-Optimierungsverfahren für Turbomaschinen

Deutsches Zentrum
für Luft- und Raumfahrt e.V.

Sicher:

Noch einbaue n: [Le Gratiet, 2013][Box & Muller, 1958 Gill, 2007][J. Forrester et al., 2006][Jones et al., 1998] [Jones, 2001](10dimensions Samples minimum) [Jin et al., 2001](3dimensions Samples minimum in vielen Fällen)

Kurzfassung

Abstract

Vorwort

Ich möchten den Bäumen und der Natur danken für die verrückten Probleme dieser Welt. Ich befürchte nur, dass die Bäume mir nicht danken werden, da einige von ihnen für meine Arbeit und die nötige Literatur draufgegangen sind. Zusätzlich meinen Kopfhörern, die mir in nervigen Zeiten durch eintönige Techno Musik etwas „ruheartiges“ verschafft haben. Ein großes Lob gilt auch den Buchstaben, ohne diese könnte ich die Arbeit hier gar nicht schreiben. Ganz zu schweigen von PDF Dateien, eine wirklich tolle Sache. Auch die Heizung in meinem Büro hat mir wärmende Linderung in den kalten Wintertagen verschafft. Manchmal war mir aber auch zu heiß, dafür möchte ich ihr natürlich nicht danken. Eigentlich ist mir immer zu warm, also brauche ich die Heizung strenggenommen nicht. Ich nehme den Dank hiermit zurück, blöde Heizung.

Nicht danken möchte ich allen tollen Filmen und Serien, die haben mich in motivationslosen Stunden eigentlich nur abgelenkt und meine Arbeit ernsthaft gefährdet.

Achja, ich möchte noch allen Menschen danken, die nicht an meinem Thema geforscht haben, ohne euch hätte das hier gar keinen Sinn gehabt und allen die auch an dem Thema forschen: Ich war schneller :P Und wenn nicht, will ich es nicht wissen....

Der deutschen Sprache kann ich nur so neutral danken, irgendwie beherrsche ich diese ja, aber irgendwie dann auch wieder nicht. Ich möchte auch meinem Gehirn danken, für diese merkwürdigen Einfälle, die es mir erlauben so einen Schwachsinn wie dieses Vorwort hier zu schreiben. Ich glaube es ist irgendwie merkwürdig seinem Gehirn zu danken, steckt da nicht mein ich drin? Naja ich bin ja eigentlich Buddhist und glaube nicht wirklich an ein ich, also existiert demzufolge mein Gehirn auch nicht und damit scheint mir (das Wort „mir“ ist ja dann auch irgendwie sinnfrei in diesem Kontext) dieser Dank auch wieder sinnlos. Alles äußerst merkwürdig und mysteriös. Mysteriös ... Übrigens habe ich während ich an dieser Arbeit gearbeitet habe, AkteX komplett geschaut. Also wirklich, die ersten Staffeln fand ich ja noch etwas zu retro, aber es wurde besser. Nach ca. 300 Folgen kommt man ganz gut rein.

Hmm mir fällt jetzt auf, dass ich manche Ausdrucke für meine Arbeit nur einseitig bedruckt habe, das werden mir die Bäume die ich anfangs erwähnt habe bestimmt übel nehmen.

Nomenklatur

Abkürzungen

AVX Advanced Vector Extensions, eine Befehlssatzerweiterung für Prozessoren. Nachfolger der SSE Architektur.

CSM Computational Structure Mechanics

EGO Efficient Global Optimization. Ein Optimierungsverfahren, welches die Unsicherheiten in einer Ersatzmodellvorhersage miteinbezieht

EVG Erwarteter Volumenzugewinn oder auch Expected Volume Gain

Freie Variable Parameter, welche der Optimierungsalgorithmus variieren darf. Meistens Geometrische Parameter wie z.B. die Länge oder Dicke einer Schaufel.

GPGPU General Purpose Computation on Graphics Processing Unit. Beschreibt die Verwendung von GPUs über den ursprünglichen Einsatz hinaus. Meist für die Verwendung von massiv parallelen rechenintensiven Algorithmen.

GPU Abkürzung für Graphics Processing Unit. Grafikbeschleuniger in der Regel als Zusatzkarte in einem Rechner

HPC High-Performance-Computing

Member Ein Satz freier Variablen mit dazugehörigen Funktionswerten. Beispielsweise ein Satz geometrischer Parameter mit berechneten mechanischen und aerodynamischen Größen.

MO Multi-Objective

ROI Region of Interest. Ein für die Zielfunktionen festgelegtes Gültigkeitsintervall.

SIMD Single Instruction Multiple Data, eine Architektur welche es erlaubt dieselbe Operation parallel auf einen sich verändernden Datenstrom anzuwenden.

SSE Streaming SIMD Extensions, eine Befehlssatzerweiterung für Prozessoren.

Formelzeichen

δ	Kronecker Delta
κ	Konditionszahl
λ	Diagonalaufschlag
$\mathbf{Cov}$	Kovarianzmatrix
$\mathbf{R}$	Korrelationsmatrix
ω	Druckverlustbeiwert
σ^2	Varianz
$\mathrm{cov}(\vec{x}, \vec{y})$	Ortsabhängige Kovarianzfunktion zwischen den Ortsvektoren x und y
$\mathrm{cov}(X, Y)$	Kovarianzfunktion zwischen den Zufallsvariablen X und Y
$\mathrm{var}(X)$	Varianz der Zufallsvariable X
$\vec{\beta}$	Beta Vektor, beinhaltet alle Erwartungswerte des Modells.
$\vec{\theta}$	Hyperparameter einer Korrelationsfunktion, ohne Varianzen oder Regularisierungsterme
$\vec{c}$	Kovarianzvektor
$\vec{r}$	Korrelationsvektor
$\vec{w}$	Gewichte eines Kriging Modells
$\vec{x}$	Ortsvektor
$\vec{y}_s$	Vektor, welcher alle bekannten Stützstellen enthält
Ξ	Eigenwert einer Matrix
$c(X, Y)$	Korrelationsfunktion zwischen den Zufallsvariablen X und Y
$E[X]$	Erwartungswert der Zufallsvariable X
$F(\vec{x})$	Fehlerfunktion an der Stelle $\vec{x}$
$f_{dec}(...)$	Entscheidungsfunktion
m_f	Anzahl der verwendenten Fidelities bei Multifidelity Verfahren
$y(\vec{x})$	Bekannter Funktionswert oder Stützstelle an der Stelle $\vec{x}$

$y^*(\vec{x})$	Geschätzter Funktionswert an der Stelle $\vec{x}$
a	Skalierungsfaktor für das CO-Kriging Kovarianzmodel
c	Anzahl der Nebenbedingungen
h	Allgemeiner Hyperparameter
I	Informationsgehalt
k	Anzahl der freien Variablen eines Members
L	Likelihood Funktion
m	Anzahl der gegebenen partiellen Ableitungen
N	Multivariate Normalverteilung
n	Anzahl der beprobten Stützstellen, auch Member oder Samples genannt
o	Anzahl der Hyperparameter
P	Wahrscheinlichkeit
p	Druck
q	Anzahl der verwendeten Ersatzmodelle innerhalb einer Optimierung
s	Anzahl der verschiedenen Gütestufen bei einem Mutlifidelty Modell
t	Zeit
z	Anzahl der Zielfunktionen einer Optimierung

Inhaltsverzeichnis

Kurzfassung **4**

Abstract **5**

1 Einleitung **14**

1.1 Motivation 14

1.2 Stand der Forschung 15

1.3 Anforderungen und Zielsetzung 22

2 Optimierungsstrategie Turbomaschine **26**

2.1 Grundlagen Turbomaschinenoptimierung 26

2.1.1 Ingenieurwissenschaftliche Disziplinen in der Turbomaschinenoptimierung 26

2.1.2 Allgemeine Begrifflichkeiten zu Optimierungen 28

2.2 Herausforderungen der Turbomaschinenoptimierung 30

2.3 Automatisierte Optimierung im DLR 34

2.3.1 Grundlegender Optimierungsprozess 34

2.3.2 Ersatzmodellbeschleunigung 37

3 Multifidelity Optimierungsstrategie Turbomaschine **43**

3.1 Gütestufen in der Turbomaschinenauslegung 43

3.1.1 Beispiel für verschiedene Gütestufen 45

3.1.2 Verschiedene Simulationsverfahren 47

3.2 Multifidelity-Optimierungsstrategie in AutoOpti 48

3.2.1 Änderungen des Optimierungsprozesses 49

3.2.2 Entscheidungsfunktion 51

3.2.2.1 Benutzergesteuerte Entscheidungsfunktionen 53
3.2.2.2 Ersatzmodellgesteuerte Entscheidungsfunktionen . . . 53

4 Die Kriging Verfahren 63
4.1 Grundlagen Kriging 64
4.2 Co-Kriging 65
4.2.1 Allgemeine Kriging Vorhersage 71
4.2.1.1 Erwartungswert 72
4.2.1.2 Varianz des Schätzfehlers direkte Herleitung 72
4.2.1.3 Kovarianz zwischen zwei Vorhersagen 73
4.2.1.4 Zusammenhang zwischen Vorhersage-Kovarianz und Fehler Varianz 74
4.2.2 Kovarianzmodell 75
4.3 Andere Kriging Verfahren 78
4.3.1 Ordinary Kriging 78
4.3.2 Gradient Enhanced Kriging 79
4.4 Kovarianzmodelle 81
4.5 Maximum Likelihood für alle Kriging Verfahren 85
4.5.1 Zusammenhang zwischen Bedingter Normalverteilung und Kriging 89
4.6 Regularisierungsterm und Nugget für alle Verfahren 89
4.7 Vergleiche zu anderen CO-Kriging Modellen 89
4.8 Vorhersagen Beispiele 92

5 Implementierung 93
5.1 Softwaredesign 94
5.1.1 Offenes Softwaredesign 94
5.1.2 Klasse zur Berechnung von Matrix Operationen 94
5.1.3 Korrelationsfunktionen 97
5.1.4 Algorithmus zur Bildung der Korrelationsmatrix für alle Kriging Verfahren 101
5.1.5 Bestimmung der Korrelationen bei Verwendung der Kovarianzmatrix und Co-Kriging 110
5.1.6 Bestimmung der Hyperparameter durch die Maximum Likelihood Methode 110

5.1.7 Likelihood 111
5.2 Minimierungsverfahren/Training 115
5.2.1 Vermeidung negativer Hyperparameter 116
5.2.2 Konvergenzkriterium für das Kriging Training 117
5.3 Analytische Bestimmung der Kriging Varianz beim Ordinary Kriging . . 117
5.3.1 Regularisierungsterm und Nugget für alle Kriging Modelle 118
5.3.1.1 Likelihood Probleme durch multiplikativen Diagonalaufschlag 122
5.3.2 Initialisierung der Hyperparameter für alle Kriging Modelle . . . 122
5.3.3 Minimierungsverfahren 129
5.3.4 Softwaretechnische Umsetzung 135
5.3.5 Renormalisierung der Hyperparameter 141
5.3.6 Konvergenz von globalen und lokalen Hyperparametern im Bezug auf die Entscheidungsfunktionen 143
5.4 Algorithmische Effizienz steigern 144
5.4.1 Filtern von unwichtigen Samples 144
5.4.2 Inverse durch Gleichungssysteme ersetzen 145
5.4.3 Vollständiger Verzicht auf die Inverse durch Likelihood Partielle Ableitungen durch Approximation der Spur 146
5.4.4 Vollständiger Verzicht auf die Inverse durch Rückwärtsdifferentiation der Cholesky Zerlegung 147
5.4.5 Vergleich zwischen der Invertierung und der Rückwärtsdifferenziereu 153
5.5 Verwendung von GPGPU 155
5.5.1 Adjoint Matrix 157
5.6 Resourcenverteilung bei parallelen Trainings 162
5.7 Analysesoftware von Krigingmodellen während der Laufzeit 164
5.7.1 Hyperparameter aus zusammengesetzten Kovarianzfunktionen plotten 164
5.8 Verteiltes Rechnen 164
5.9 Umsetzung der Entscheidungsfunktion 165

6 Benchmarks **166**

6.1 GPU Benchmarks 166

6.2 Analytische Tests Ersatzmodelle 166
6.2.1 Initialisierungsverfahren im Vergleich 166
6.3 Analytische Tests Optimierung 168
6.4 2 Mises Optimierung 168
6.5 Trace Erich 169

7 Reale Turbomaschinen Optimierung 170
7.1 un 1 175
7.2 Run 2 175

8 Fazit und Ausblick 176
8.0.1 Gradient Enhanced CO-Kriging 177

A Anhang 178
A.1 Aerodynamische Größen 178
A.2 Varianz der Fehlerfunktion 178
A.3 Kovarianz zwischen Vorhersagen 180
A.4 Varianz des Schätzfehlers 182
A.5 Zusammenhang Kovarianz und Fehler Varianz 185
A.6 Likelihood Schätzer Varianzen im CO-Kriging 186
A.7 Korrelationsmatrix oder Kovarianzmatrix 187
A.8 Vereinfachung der Vorhersagegleichung 189
A.9 Beschleunigter Rückwärtsdifferenzierter-Cholesky-Algorithmus nach Smith 190
A.10 Alternative Herleitung der symbolischen Differenzierung 194
A.11 Wann würde eine globales Varianzkriterium zu einer anderen Entscheidung führen als das lokale? 195

1 Einleitung

1.1 Motivation

Der Luftverkehr im Fracht- und Passagierbereich ist in den letzten Jahren sehr stark angestiegen. Gleichzeitig steigt auch der weltweite Energieverbrauch [p.l.c., 2016]. In beiden Bereichen stellen Turbomaschinen eine sehr wichtige technische Komponente dar. Das Thema Klimaschutz ist zunehmend in den Vordergrund gerückt. Die Politik stellt hier sehr hohe Anforderungen, welche bei der Auslegung von Turbomaschinen mit berücksichtigt werden müssen. Außerdem herrscht ein sehr hoher Konkurrenz- und Preisdruck unter den Triebwerks- und den Energiemaschinenherstellern. Diese konkurrierenden Anforderungen machen die Auslegung von Turbomaschinen zu einer sehr großen Herausforderung für Forschung und Industrie und erfordern bereits während des Auslegungsprozesses ein besonderes Augenmerk auf Emissionen, Effizienz, Betriebs- und Wartungskosten.

Um diesen Problemen zu begegnen, sind moderne CFD- und Optimierungsverfahren ein sehr wichtiges Werkzeug. Sie erlauben es, die komplexen Strömungsvorgänge in Turbomaschinen am Computer zu simulieren und unter Berücksichtigung multipler Ziele und Nebenbedingungen zu optimieren. Diese noch relativ neuen Werkzeuge führen zu einer Verlagerung von sehr kostspieligen experimentellen Untersuchungen hin zu Simulationen am Computer durch CFD-Verfahren. Heutzutage werden experimentelle Untersuchungen vermehrt zur Validierung und Kalibrierung von CFD-Verfahren verwendet und finden zudem Anwendung im Bereich der Grundlagenforschung von Strömungsphänomenen.
Durch die neuen numerischen Verfahren wurden die Auslegungszyklen im Turbomaschinenbereich in den letzten Jahren massiv beschleunigt. Allerdings sind die genutzten Verfahren mit einem so hohen numerischen Aufwand verbunden, dass vereinfachte CFD-Modelle verwendet werden. Durch diese Vereinfachungen können diese Modelle allerdings nicht alle physikalischen Phänomene auflösen und sind numerisch immer noch sehr aufwendig.

Zur Bewältigung dieses Problems, werden zunehmend sehr große HPC-Cluster eingesetzt um die benötigte Rechenleistung zur Verfügung zu stellen. Solch ein Cluster

ist allerdings sehr kostspielig und die Optimierungs- und CFD-Verfahren müssen auf die sehr spezielle Hardware von solchen Clustern angepasst werden. Diese Problemstellung ist so komplex, dass sich daraus ein eigener Forschungszweig entwickelt hat. Dennoch ist man trotz der enormen Rechenleistung, die moderne HPC-Cluster bieten, immer noch auf stark vereinfachte CFD-Modelle und hocheffiziente Optimierungsverfahren angewiesen. Beispielsweise ist man von dem Ziel, ein vollständiges Flugzeug inklusive Antrieb am Rechner zeitlich aufgelöst zu simulieren und zu optimieren noch sehr weit entfernt. Selbst die Optimierung eines vollständigen Triebwerks ist mit den momentan zur Verfügung stehenden Ressourcen nicht durchführbar.

In der beschriebenen Problemstellung liegt auch die Motivation dieser Arbeit. Es soll ein industriell nutzbares Optimierungsverfahren vorgestellt werden, welches es ermöglichen soll, höherwertigere CFD-Verfahren innerhalb der Auslegung von Gasturbinen zu verwenden. Hierfür wird ein effizientes Optimierungsverfahren entwickelt und getestet, wobei ein besonderes Augenmerk auf die industrielle HPC-Hardwareumgebung und eine hohe numerische Effizienz gelegt wird. Durch dieses Verfahren soll es in Zukunft ermöglicht werden, deutlich komplexere numerische Simulationen innerhalb von automatisierten Optimierungen zu verwenden.

1.2 Stand der Forschung

Industriell genutzte Optimierungsverfahren für den Turbomaschinenbereich sind schon seit längerer Zeit ein umfangreiches Forschungsgebiet und Thema zahlreicher Arbeiten und Forschungsbeiträge. Eine vollständige Übersicht der vorhandenen Literatur ist daher nahezu unmöglich. Dennoch soll dem Leser ein kurzer Überblick über einige aktuelle Forschungsarbeiten im Bereich der Turbomaschinenoptimierung geboten werden.

Bei Optimierungen im Turbomaschinenbereich handelt es sich meist um nichtlineare mehrdimensionale multidisziplinäre Mehrziel-Optimierungsprobleme mit Nebenbedingungen. Es werden oftmals Geometrien für Turbomaschinenkomponenten gesucht, welche einen besonders hohen Wirkungsgrad aufweisen. In der Regel unter Einhaltung aerodynamischer, mechanischer, akustischer oder anderer Restriktionen. Aufgrund der Komplexität der hier beschriebenen Optimierungsprobleme sind in den letzten Jahren zahlreiche Optimierungsverfahren für die Turbomaschinenauslegung entwickelt worden. Innerhalb dieses Abschnitts sollen die in der Turbomaschinenforschung aktuell verwendeten Verfahren kurz beschrieben und mit Literatur belegt werden. Zu diesem Zweck wird zu jedem Verfahren zuerst allgemeine Übersichtsliteratur vorgestellt und anschließend folgen einige Veröffentlichungen von industrienahen Anwen-

dungen dieser Verfahren aus den letzten sieben Jahren.

Optimierungen auf Basis der Evolutionsstrategie: Als erstes sollen einige allgemeine Beiträge aus dem Bereich der evolutionären Optimierungsalgorithmen vorgestellt werden. Bei evolutionären Optimierungsalgorithmen handelt es sich um stochastische Optimierungsverfahren, welche der Funktionsweise der natürlichen Evolution von Lebewesen nachempfunden wird. Im Wesentlichen werden bei einem evolutionären Algorithmus aus einem vorhandenen Pool bekannter Daten die Besten selektiert, dann rekombiniert und/oder mutiert und wieder evaluiert. Dieser Prozess wird dann solange durchlaufen, bis ein zufriedenstellendes Ergebnis erreicht wird. Die Vorteile dieses Verfahrens liegen in der globalen Suche nach einem Optimum und darin, dass so gut wie keine Anforderungen an die Zielfunktion und Nebenbedingungen gestellt werden. Die Zielfunktion muss weder differenzierbar noch an jeder Stelle auswertbar sein. Dies macht das Verfahren sehr robust. Der größte Nachteil bei dieser Art von Optimierung liegt in der Konvergenzgeschwindigkeit: Es sind meist sehr viele Tastschritte nötig, um ein zufriedenstellendes Optimum zu finden.

Grundlegende Literatur über evolutionäre Algorithmen bieten zum einen die beiden Arbeiten von Holland [Holland, 1975] und Rechenberg [Rechenberg, 1973], welche den Grundstein der heutigen modernen Algorithmen darstellen. Zum anderen bieten die Bücher von Gen et al. [Gen & Cheng, 2000] und Weicker [Weicker, 2015] einen umfassenden und aktuellen Überblick über die gesamte Thematik.

Für den Einsatz von evolutionären Algorithmen im Bereich der Turbomaschinenoptimierung gibt es zahlreiche Anwendungsbeispiele. Die Arbeit von Sozio et al. [Sozio et al., 2013] stellt ein aktuelles und industrienahes Beispiel dar und soll daher kurz beschrieben werden. Innerhalb dieser Arbeit wird ein Auslegungsprozess für gegenläufige Axialturbinen auf Basis eines evolutionären Algorithmus vorgestellt. Der beschriebene Auslegungsprozess soll ausgehend von eindimensionalen Kenngrößen eine dreidimensionale aerodynamisch optimierte Geometrie erzeugen. Um dies zu erreichen, verwendet der beschriebene Auslegungsprozess mehrere nacheinander durchgeführte evolutionäre Optimierungen. Sozio testete den entwickelten Auslegungsprozess an einer gegenläufigen Axialturbine und konnte damit eine aerodynamisch sinnvolle dreidimensionale Geometrie einer mehrstufigen Axialturbine erzeugen.

Es gibt noch eine Vielzahl von weiteren interessanten Anwendungsbeispielen. Da sich diese Arbeit allerdings mit dem Thema der ersatzmodellgestützen Optimierung befasst, soll der Fokus auf diese Thematik gelegt werden.

Beschleunigung von Optimierungen mit Ersatzmodellen: Bei Ersatzmodellen handelt es sich um Interpolations- oder Approximationsverfahren, welche während ei-

ner Optimierung verwendet werden, um Vorhersagen über unbekannte Parametersätze zu machen und damit besonders vielversprechende Parametersätze zu finden und somit die Optimierung zu beschleunigen. Da sich eine Vielzahl an aktuellen Arbeiten mit diesem sehr komplexen Themengebiet befassen, soll es dem Leser zuerst ermöglicht werden einen Überblick über dieses Themengebiet zu bekommen. Hierfür werden folgend einige interessante Übersichtsbeiträge für diesen speziellen Bereich der Turbomaschinenoptimierung genannt. Als erstes soll ein sehr umfassender Übersichtsbeitrag von Queipo et al. [Queipo et al., 2005] erwähnt werden. Dieser befasst sich unter anderem mit initialen Sampling-Prozessen, der Auswahl richtiger Ersatzmodelle für verschiedene Anwendungen und deren Verwendung innerhalb von multidisziplinären Optimierungen. Eine weitere sehr interessante Übersichtsarbeit bietet Simpson et al. [Simpson et al., 2008]. In diesem Beitrag wird die historische Entwicklung von ersatzmodellgestützten multidisziplinären Optimierungen innerhalb der letzten 20 Jahre beschrieben. Simpson kommt zu dem Schluss, dass die Herausforderungen wie der „Fluch der Dimensionen“ oder die Komplexität der numerischen Verfahren immer noch dieselben sind wie vor 20 Jahren. Er hebt allerdings hervor, dass die berechenbare Größe dieser Herausforderungen deutlich gestiegen ist (vgl. [Simpson et al., 2008], Seite 14).

Der Beitrag von Forrester et al. [Forrester & Keane, 2009] bietet einen technisch fundierten Einblick in diesen Bereich und beschreibt mehrere Ersatzmodellverfahren und deren Anwendung im Luft- und Raumfahrtbereich. Es wird ebenfalls ein kurzer Einblick in die Nutzung von Gradienteninformation und mehreren Gütestufen gegeben.

Im Folgenden sollen einige aktuelle Anwendungen von ersatzmodellbeschleunigten Optimierungen aus dem Turbomaschinenbereich genannt werden. Verstraete et al. [Verstraete et al., 2014] zeigt die erfolgreiche Anwendung von Neuronalen Netzwerken bei der Optimierung eines Diffusors einer Niederdruckdampfturbine. Die durchgeführte Optimierung sollte zum einen den Wirkungsgrad erhöhen und zum anderen den Betriebsbereich der Maschine erweitern. Weiterhin soll die Frage beantwortet werden, inwiefern die Verwendung von mehreren Betriebspunkten eine solche Optimierung beeinflusst. Das verwendete Optimierungsverfahren wird im Von-Karman-Institut in Belgien entwickelt und in den Arbeiten von Pierret [Pierret & Van den Braembussche, 1998, Pierret, 1999] und Verstraete et al. [Verstraete, 2008] beschrieben. Verstraete kommt in seiner Arbeit zu dem Schluss, dass eine substantielle Verbesserung (vgl. [Verstraete et al., 2014], Seite 9) erreicht werden konnte und die Berücksichtigung mehrerer Betriebspunkte in einer verbesserten Charakteristik des Betriebsverhaltens (vgl. [Verstraete et al., 2014], Seite 9) resultiert.

Eine weitere interessante Optimierung einer hochbelasteten Fan-Stufe bietet die Arbeit von Aulich et al. [Aulich & Siller, 2011]. Innerhalb dieser Arbeit wird die Opti-

mierung einer Fan-Stufe mit über 230 Design-Parametern und zahlreichen Nebenbedingungen beschrieben. Das dort angewendete Verfahren ist ein durch Ordinary-Kriging beschleunigter evolutionärer Algorithmus, welcher im Deutschen Zentrum für Luft- und Raumfahrt entwickelt wird. Beim Ordinary-Kriging handelt es sich um ein robustes statistisches Interpolationsverfahren. Ein statistisches Interpolationsverfahren sagt nicht nur den reinen Funktionswert voraus, sondern eine statistische Verteilung dessen. Praktisch bedeutet dies, dass eine Abschätzung über den Fehler der Vorhersage gemacht werden kann. Diese Information kann in einem Optimierungsverfahren gewinnbringend eingesetzt werden und wird oft mit EGO für Efficient Global Optimization abgekürzt. Wichtige theoretische Beiträge zu diesem Verfahren liefern die Arbeiten von Jones et al. [Jones et al., 1998], Keane [Keane, 2006] und Aulich [Aulich & Siller, 2011]. Eine Anwendung einer solchen EGO-Optimierung findet sich in der Arbeit von Voß et al. [Voß et al., 2014]. In diesem Beitrag wird eine aeromechanische Optimierung eines bereits ausgelegten transsonischen Radialverdichters beschrieben. Die durchgeführte Optimierung übertrifft den Wirkungsgrad der ursprünglichen Geometrie um ca. 2% und auch der Sperrmassenstrom ist um ca. 7% gesteigert. Die darauf aufbauende Arbeit von Elfert et al. [Elfert et al., 2016] bestätigt diese Ergebnisse experimentell.

Eine weitere Anwendung des EGO-Algorithmus wird in der Arbeit von Lepot et al. [Lepot et al., 2011] dargestellt. Es wird die aeromechanische Optimierung eines gegenläufigen offenen Rotors unter Berücksichtigung von akustischen Kriterien beschrieben.

Die Anwendung einer ersatzmodellgestützten Optimierung zum allgemeinen Vergleich einer gegenläufigen und einer konventionellen Fanstufe wird in der Dissertation von Lengyel [Lengyel-Kampmann, 2015] dargestellt. Dabei sollte der Wirkungsgrad der Stufe optimiert werden, bei einer Gleichverteilung von Machzahl und Druckverhältnis. Durch diese Technik lässt sich eine optimale Wirkungsgradfläche über Machzahl und Druckverhältnis auffinden. So konnten innerhalb dieser Arbeit allgemeingültige Vergleiche über die Effizienz einer solchen Fanstufe angestellt werden. In der Arbeit von Baert et al. [Baert et al., 2015] wird ein Verfahren zur Optimierung von kontinuierlichen und nicht kontinuierlichen Designparametern auf Basis von Ersatzmodellen vorgestellt und an einem Bypass Kanal getestet. Wobei mit nicht kontinuierliche Designparameter, z.B. ganzzahlige Parameter wie Schaufelzahlen, gemeint sind. Baert kommt zu dem Schluss, dass diese Art von Ansatz wichtige Design-Entscheidungen in die Optimierung integriert (vgl. [Baert et al., 2015], Seite 11-12) und belegt dies mit der erfolgreichen Optimierung eines Bypass-Kanal.

Eine andere Sicht auf ersatzmodellgestützte Optimierungen bietet der Beitrag von Chahine et al. [Chahine et al., 2012]. Innerhalb des Beitrags wird der Nutzen von Neuronalen-Netzwerken zur Beschleunigung von evolutionsbasierten Optimierungen

in Frage gestellt. Chahine belegt diese Aussage mit der Vergleichsoptimierung eines Radialverdichters. Dieses Ergebnis ist allerdings verwunderlich, da die Verwendung von Ersatzmodellen zur Beschleunigung von evolutionären Optimierungsverfahren als besonders effizientes Verfahren gilt. Es gilt aber zu beachten, dass die Effizienz Neuronaler Netzwerke äußerst abhängig von deren Aufbau und Struktur ist. Dies ist auch Gegenstand zahlreicher aktueller Forschungsarbeiten im Bereich des maschinellen Lernens, bspw. die Arbeit von Silver et al. [Silver et al., 2016]. Verfahren wie das Kriging sind in dieser Hinsicht robuster. Dennoch zeigt das Ergebnis sehr deutlich, dass die Effizienz von Optimierungen mit Ersatzmodellen sehr unterschiedlich ausfallen kann und stark von den verwendeten Optimierungsverfahren, Ersatzmodellen und auch der Anwendung abhängt. Ersatzmodelle können eine Optimierung stark beschleunigen, allerdings wird auch die Komplexität des Verfahrens durch den Einsatz von Ersatzmodellen deutlich erhöht. Dies erfordert in der Regel einen erfahrenen Anwender zur Überwachung solcher Optimierungen.

Gradienteninformation in Optimierungen: Weiterhin beschäftigt sich ein ein großer Teil der aktuellen Forschungsbeiträge mit der Verwendung von Gradienteninformation in Optimierungen. Moderne Strömungslöser und die dazugehörige Software, wie z.B. Rechennetzgeneratoren, bieten immer häufiger die Berechnung partieller Ableitungen von Funktionalen, wie z.B. Wirkungsgrad nach geometrischer Deformation an. Die numerischen Kosten für die Bestimmung mehrerer partieller Ableitungen befinden sich hierbei in einer ähnlichen Größenordnung wie die Strömungslösung selbst. Die Nutzung dieser Gradienteninformation kann innerhalb einer Optimierung direkt verwendet werden. Dies wird erreicht, indem von einem beliebigen Startpunkt in Gradientenrichtung abgestiegen wird und so sehr schnell ein Minimum gefunden werden kann. Eine andere Möglichkeit ist die Verwendung von Ersatzmodellen. Einige Ersatzmodelle können die vorhandene Gradienteninformation zur Verbesserung der Vorhersagen heranziehen und bieten den Vorteil, dass keine vollständige Gradienteninformation vorhanden sein muss. Dies ist wichtig, da in den meisten Turbomaschinenoptimierungen für einige Funktionale keine Gradienteninformation verfügbar ist.

Ein aktuelles Beispiel in diesem Bereich bietet die Arbeit von Willeke et al. [Willeke & Verstraete, 2015]. Es wird eine Optimierung beschrieben, welche den Druckverlust eines Kühlkanals mithilfe eines Gradientenabstiegverfahrens optimiert und mit einem rein evolutionären Algorithmus vergleicht. Bei dem angestellten Vergleich konnte sich das Gradientenverfahren als das schnellere Verfahren behaupten. Weiterhin schreibt Willeke, dass die Sorge in ein lokales Minimum zu geraten in dem analysierten Testfall nicht relevant ist, da der verwendete Ansatz in sehr kurzer Zeit eine ausreichende Verbesserung erzielen konnte (vgl. [Willeke & Verstraete, 2015], Seite 11-12).

Ein weiterer Vergleich wird in dem Beitrag von Tang [Tang et al., 2016] angestellt. Verglichen wird eine ersatzmodellgestützte Optimierung mit einem Gradientenabstiegsverfahren. Das verwendete Ersatzmodell (Radiale-Basisfunktionen) die vorhandene Gradienteninformation verarbeiten kann. Tang kommt in seiner Arbeit zu dem Schluss, dass das reine Abstiegsverfahren sehr schnell ein lokales Minimum findet und sich nicht sehr weit von der initialen Geometrie wegbewegt. Das ersatzmodellgestützte Verfahren konnte in derselben Zeit die Region des globalen Optimums auffinden (vgl [Tang et al., 2016], Seite 12).

Erwähnenswert sind innerhalb der Verfahren der gradientenbasierten Turbomaschinenoptimierung auch die sogenannten One-Shot-Verfahren. Im Gegensatz zu den klassischen Optimierungsverfahren wird hier innerhalb des Iterationsverfahrens einer Strömungssimulation eine optimale Geometrie gesucht. Hierfür sind innerhalb der Lösung ebenfalls Gradienteninformationen notwendig und das verwendete Rechennetz muss deformiert werden. Diese Verfahren sind in der Regel um ein Vielfaches schneller als klassische Optimierungsansätze. Allerdings sind multidisziplinäre Problemstellungen, instationäre Strömungslösungen und auch Berechnungen mit verschiedenen gekoppelten Simulationsverfahren eine sehr große Herausforderung. Daher bieten diese Verfahren oftmals nicht die benötigte Flexibilität einer „konventionellen“ Optimierung. Für eine aktuelle und umfassende Übersicht in dem Bereich der One-Shot-Verfahren sei hier auf die Disseration von Ozkaya [Özkaya & Gauger, 2014] verwiesen.

Multifidelity Verfahren: Innerhalb dieser Arbeit soll ebenfalls ein ersatzmodellgestütztes Multifidelity-Verfahren entwickelt werden. Dieses Verfahren soll in der Lage sein, während einer Optimierung verschiedene Gütestufen von Daten mit Hilfe von Ersatzmodellen zu verarbeiten. Eine mögliche Gütestufen ist z.B. die örtliche Diskretisierung eines zu simulierenden Strömungskanals in der Form eines Rechengitters. Je nach Anzahl der Zellen dieses Netzes werden einige Strömungsphänome besser aufgelöst, was allerdings mit einer erheblich gesteigerten Rechenzeit bezahlt werden muss. Der Anwender muss also vor einer Optimierung immer einen Kompromiss, zwischen Genauigkeit und Geschwindigkeit treffen.

In der Regel sind die Gütestufen stark korreliert, oftmals aber verzerrt, verschoben oder skaliert. Es ist also ein Ersatzmodell notwendig, welches diese verschiedenen Gütestufen verarbeiten kann und die Transferfunktion zwischen den Gütestufen automatisch auffindet. Die einfachste Möglichkeit eines solchen Ersatzmodells stellen sogenannte Brückenfunktionen dar. Eine solche Brückenfunktion wird aus zwei Ersatzmodellen gebildet, wobei das eine nur die niedrigere Stufe darstellt und das andere die Differenz zwischen hoher und niedriger Stufe wiedergibt. Addiert man die Vorhersagen beider Modelle, so bekommt man eine Vorhersage für die höchste Gütestufe unter Berücksichtigung der Niedrigeren. Je nach Komplexität der Differenzfunktion können

bereits mit sehr wenigen Daten der höherwertigen Funktion gute Vorhersagen aus den Daten niedriger Güte gewonnen werden. Beispielsweise verwendet die bekannte Open-Source-Optimierungssoftware Dakota solche additiven Brückenfunktionen in der aktuellen Version 6.5 (vgl. [Adams et al., 2016]).

Ein statistisches Verfahren, welches auch komplexere Zusammenhänge beschreiben kann, ist das CO-Kriging-Ersatzmodell. In der Arbeit von Han et al. [Z.-H. Han, R. Zimmermann, 2010] wird ein Vergleich zwischen einem CO-Kriging-Modell und einer additiven Brückenfunktion angestellt, wobei sich das CO-Kriging-Modell als das leistungsfähigere herausgestellt hat. Wie das Ordinary-Kriging, sagt auch das CO-Kriging-Modell eine Verteilung voraus und lässt sich damit in dem „Efficient Global Optimization“ Verfahren verwenden. Es gibt zahlreiche Umsetzungen des CO-Kriging Verfahrens, wobei die Meisten auf der Arbeit von Kennedy und O'Hagan [Kennedy & O'Hagan, 2000] beruhen. Für das CO-Kriging Modell von Kennedy und O'Hagan muss allerdings an jeden bekannten Ort der höchsten Gütestufe auch die niedrigere Gütestufe berechnet werden. Innerhalb dieser Arbeit soll eine CO-Kriging Variation vorgestellt werden, die dies nicht benötigt. Weitere interessante CO-Kriging Ansätze bietet zum einen die Arbeit von Le Gratiet [Le Gratiet, 2013], welche eine sehr effiziente Umsetzung der Methode von Kennedy darstellt und zum anderen die Arbeiten von Han [Han et al., 2012, Z.-H. Han, R. Zimmermann, 2010].

Neben der technischen Umsetzung des CO-Kriging-Verfahrens ist auch die Verwendung eines solchen Modells innerhalb einer Multifidelity-Optimierung ein sehr wichtiger Punkt. Dabei sind die Entscheidung wann welche Gütestufe berechnet wird oder die Daten- und Prozesskettenverwaltung die wesentlichen Herausforderungen.

Leider existieren zu dieser Thematik nur sehr wenige Forschungsbeiträge. Eine akademische Anwendung bietet hier die Arbeit von Brooks [Brooks et al., 2011] dieser stellt einen Vergleich zwischen einer Multifidelity-Optimierung und einer konventionellen Ordinary-Kriging-Optimierung an. Brooks optimierte für diesen Zweck den bekannten NASA-Rotor 37 [Reid & Moore, 1978] mit dem Ziel den Wirkungsgrad zu verbessern, wobei das originale Druckverhältnis und der Massenstrom beibehalten werden sollte. Um eine Vergleichbarkeit gewährleisten zu können, stellte Brooks für beide Optimierungen dasselbe Rechenbudget zur Verfügung. Weiterhin wurde innerhalb der Optimierung ein industriell genutztes 3D-CFD-Verfahren verwendet [Lapworth & Shahpar, 2004]. Brooks kommt in seiner Arbeit zu dem Ergebnis, dass die Multifidelity-Optimierung bei gleicher Rechenzeit eine höhere Effizienzsteigerung erreicht. In der durchgeführten Multifidelity-Optimierung konnte eine Steigerung des Wirkungsgrad von 2.34% erreicht werden wobei die Referenzoptimierung eine Steigerung von 1.79% erzielte. Dieses Ergebnis lässt darauf hoffen, dass bei komplexeren industriellen Optimierungen ein deutlicher Geschwindigkeitszuwachs zu erwarten ist.

1.3 Anforderungen und Zielsetzung

Im Rahmen dieser Arbeit soll ein industriell einsetzbares und erweiterbares Multifidelity-Optimierungsverfahren entwickelt und getestet werden. Das Verfahren soll in die bereits bestehende Optimierungssoftware AutoOpti integriert werden und dabei die speziellen Anforderungen im Bereich der Turbomaschinenentwicklung berücksichtigen. In diesem Abschnitt sollen zuerst die Anforderungen an das entwickelte Verfahren herausgearbeitet und daraus folgend eine konkrete Zielsetzung bestimmt werden.

Die Anforderungen lassen sich hierbei in zwei Kategorien unterteilen. Die eine Kategorie umfasst die Anforderungen durch die spezielle Problematik in der Turbomaschinenauslegung. Besonders kritische Punkte sind hierbei die sehr aufwendigen Prozessketten, die hochdimensionalen Suchräume und die Vielzahl an auftretenden physikalischen Disziplinen. Die andere Kategorie betrifft die computertechnischen Anforderungen, wie die Unterstützung einer HPC-Umgebung oder eine sinnvolle Integration in die bereits vorhandene Software AutoOpti.

Anforderungen Turbomaschinenoptimierung

Die speziellen Anforderungen von Turbomaschinenoptimierungen können zusammengefasst werden zu folgenden Punkten:

- Mehrere Zielfunktionen
- Viele Restriktionen
- Unterschiedliche physikalische Disziplinen
- Hochdimensionale Suchräume
- Numerisch sehr aufwendige Prozessketten

Grundsätzlich werden alle Punkte bereits durch die Optimierungssoftware AutoOpti unterstützt. Da die Anforderungen an die Turbomaschinenindustrie und damit auch an die entsprechenden Optimierungen stetig steigen, werden zukünftig leistungsfähigere Optimierungsverfahren benötigt. Durch das Multifidelity-Verfahren sollen insbesondere die hochdimensionalen Suchräume effizienter abgetastet werden können und dadurch auch der Aufwand durch die numerisch aufwendigen Prozessketten reduziert werden.

Software- und hardwaretechnische Anforderungen

Die Entwicklung eines solchen Verfahrens bringt auch zahlreiche softwaretechnische Anforderungen mit sich. Im folgenden sind die wichtigsten Punkte aufgelistet:

- Integration in die bestehende Optimierungssoftware AutoOpti
- Unterstützung einer HPC-Rechnerarchitektur und GPUs
- Offene objektorientierte Softwarestruktur
- Hohe Effizienz des Optimierungsverfahrens, insbesondere des Ersatzmodells

Die Software AutoOpti ist vollständig in der Programmiersprache C99 geschrieben. Zusammen mit der Forderung nach einer objektorientierten Softwarelösung wurde für die Entwicklung des neuen Verfahrens die Programmiersprache C++ inklusive der Verwendung der umfangreichen Boost Bibliothek gewählt. Der Vorteil liegt darin, denselben Compiler verwenden zu können und den AutoOpti-Entwicklern eine verwandte Sprachumgebung anzubieten.

Weiterhin werden Optimierungen im Turbomaschinendesign aufgrund der komplexen numerischen Prozesskette, meist auf HPC-Großrechnern ausgeführt. Die innerhalb dieser Arbeit entwickelte Software muss in einer solchen Umgebung lauffähig sein. In den letzten Jahren ist besonders im HPC-Bereich die Verwendung von GPUs für komplexe Rechenaufgaben ein sehr wichtiges Thema geworden. Diese Zusatzkarten bieten eine im Vergleich zu herkömmlichen CPUs eine größere Rechenleistung bei vergleichsweise niedrigem Energieverbrauch. Allerdings benötigen diese GPUs stark parallelisierte SIMD-Algorithmen, was den Entwicklungsaufwand erhöht. Da das in dieser Arbeit entwickelte Optimierungsverfahren selbst sehr aufwendig ist und eventuelle Rechenzeiten den Optimierungsverlauf empfindlich stören könnten, soll die in dieser Arbeit entwickelte Software durch aktuelle GPUs optional beschleunigt werden. Zudem sollen aktuelle GPUs mit diesem Verfahren getestet und auf Eignung überprüft werden.

Ein weiterer wichtiger Punkt ist die Forderung einer offenen objektorientierten Softwarestruktur, um eine spätere Erweiterbarkeit zu gewährleisten.

Zielsetzung

Die aus den Anforderungen resultierende Zielsetzung soll folgend beschrieben werden. Hierfür werden in der folgenden Auflistung die wichtigsten Ziele dieser Arbeit zusammengefasst und im darauffolgenden Text genauer beschrieben.

1. Entwicklung eines CO-Kriging-Ersatzmodells
   (a) Objektorientierte Sprache
   (b) Ersatzmodellsoftware muss erweiterbar aufgebaut sein
   (c) Prozessinterne synchrone Parallelisierung
   (d) Prozessweite asynchrone Parallelisierung: Client/Server System

   (e) GPU-Unterstützung

2. Erweiterung des bisherigen Optimierungsverfahrens

   (a) Offene Schnittstellen zwischen Optimierer und Ersatzmodellen

   (b) Automatisierte Entscheidung zur Laufzeit über die verwendete Gütestufe

   (c) Erweiterung der Optimierer-Datenbank auf verschiedene Gütestufen

   (d) Erweiterung des Optimierers auf verschiedene Prozessketten für die jeweiligen Gütestufen

3. Testen des entwickelten Verfahrens an analytischen und auch realitätsnahen Beispielen

4. Praktische Anwendbarkeit an einer industriellen Turbomaschinenoptimierung zeigen

Der erste Punkt umfasst die Entwicklung und Integration eines CO-Kriging-Ersatzmodells für die AutoOpti-Umgebung. Das Ersatzmodell soll in einer objektorientierten Sprache programmiert werden, wie bereits erwähnt wurde die Sprache C++ gewählt. Die Erweiterbarkeit des Verfahrens spielt ebenfalls eine sehr große Rolle, zum einen sollen hier alle Kriging Verfahren (Ordinary-, Gradient-Enhanced-,CO-Kriging) in einer Softwarestruktur untergebracht werden und zum anderen sollen auch Klassifikatoren wie z.B. Supporting-Vector-Machines in die Software implementiert werden. Weitergehend soll die Software über eine prozessinterne synchrone Parallelisierung durch Threads verfügen und auch moderne Befehlssatzerweiterungen wie SSE unterstützen. Eine weitere Möglichkeit der Beschleunigung ist die prozessweite Parallelisierung, damit sollen die notwendigen Berechnungen auch über Rechnergrenzen hinaus (bspw. auf einem Rechencluster) parallelisierbar sein. Dies soll durch ein eigen entwickeltes Client/Server System erfolgen. Zudem soll die praktische Anwendbarkeit und der Nutzen eines solchen Verfahrens überprüft werden. Weiterhin soll die Verwendung von GPUs möglich sein, womit die notwendigen Berechnungen für das Ersatzmodell beschleunigt werden sollen.

Der zweite Punkt umfasst die Erweiterung der bestehen Optimierungssoftware AutoOpti. Grundsätzlich müssen allgemeine Schnittstellen zwischen Optimierer und Ersatzmodellen definiert und umgesetzt werden. Zudem muss eine automatisierte Strategie entwickelt werden, um während einer laufenden Optimierung zu bestimmen, welche Güte in der nächsten Evaluierung verwendet werden soll. Die Optimierungssoftware muss außerdem die Daten verschiedener Güte speichern und auswerten können und zudem die Verwaltung von unterschiedlichen Prozessketten für die jeweiligen Gütestufen übernehmen.

Der dritte Punkt umfasst erste analytische Tests, in denen einzelne Komponenten des neuen Verfahrens auf ihre Effizienz und Fehlerfreiheit getestet werden sollen.

Der vierte und letzte Punkt soll die Anwendung des gesamten Verfahrens in einer umfangreichen industriellen Turbomaschinenoptimierung darstellen. Wobei eine Anwendung mit sehr hohen Prozesskettenlaufzeiten und großen Parameterraum gewählt werden soll. Die Komplexität soll möglichst hoch sein, sodass man diese Optimierung in dieser Form nicht mit einer reinen Single-Fidelity-Optimierung durchführen würde.

2 Optimierungsstrategie Turbomaschine

In diesem Kapitel erfolgt eine erste Einführung in die Turbomaschinenoptimierung. Hierfür werden im ersten Teil allgemeine Grundlagen und Begrifflichkeiten erklärt, wobei ein Grundwissen über Optimierung vorausgesetzt wird. Im zweiten Teil sollen die Besonderheiten und damit verbundenen Schwierigkeiten der Optimierung von Turbomaschinen erläutert werden. Darauf aufbauend wird im dritten Teil dieses Kapitels die bisher im DLR umgesetzte Optimierungsstrategie beschrieben.

2.1 Grundlagen Turbomaschinenoptimierung

Die Auslegung und Optimierung von Turbomaschinen ist eine multidisziplinäre Aufgabe, für die verschiedenste Anforderungen aus einer Vielzahl von physikalischen Disziplinen gegeneinander abgewogen und bewertet werden müssen. Diese Disziplinen sind für einen sicheren, effizienten und umweltverträglichen Betrieb verantwortlich. Zum besseren Verständnis der Komplexität dieser Aufgabenstellung sollen nachfolgend die wesentlichen Fachdisziplinen aufgelistet und mit Beispielen hinterlegt werden.

2.1.1 Ingenieurwissenschaftliche Disziplinen in der Turbomaschinenoptimierung

Aerodynamik Die Aerodynamik von Turbomaschinen kann sich auf verschiedene Baugruppen einer Turbomaschine beziehen, bspw. den Fan, Verdichter, Brennkammer oder die Turbine. Ein häufiges Ziel ist z.B. das Auffinden einer Geometrie für eine oder mehrere Schaufelreihen, welche den Wirkungsgrad (Verhältnis von genutzter Energie zu zugeführter Energie) und gleichzeitig auch das Druckverhältnis (Verhältnis vom Druck am Austritt der Komponente zum Druck am Eintritt der Komponente) maximiert. In der Regel werden zusätzlich Nebenbedingungen definiert, bspw. ein bestimmter Ab-

strömwinkel in Umfangsrichtung und/oder ein einzuhaltender Betriebsbereich. Diese Ziele und Nebenbedingungen müssen für mehrere Betriebspunkte der Komponente definiert und eingehalten werden, wodurch sehr viele Nebenbedingungen und komplexe Zielfunktionen entstehen können. Um diese Kennzahlen zu erhalten, wird aus einem vorher definierten Parametersatz eine Geometrie erzeugt und diese dann mittels eines CFD-Verfahrens simuliert. Dieser Prozess ist numerisch extrem aufwendig und benötigt pro Geometrie auf einem modernen Rechencluster mehrere Stunden. Bei komplexen Simulationen sind auch Rechenzeiten von mehreren Tagen möglich.

Strukturmechanik Die Strukturmechanik von Turbomaschinen beschäftigt sich mit der Bestimmung der mechanischen Belastung der verwendeten Bauteile. Hierzu gehören z.B. die Berechnung der auftretenden Spannungen in den Schaufelreihen durch die Fliehkraft- und Druckbelastung. Zudem müssen dynamische Problemstellungen beachtet werden. Beispielsweise müssen im gesamten Betriebsbereich der Turbomaschine resonante Schaufelschwingungen vermieden werden. Um dies zu realisieren werden für verschiedene Drehzahlen die Eigenfrequenzen und Eigenformen der Schaufelreihen mit modernen CSM-Verfahren berechnet.

Oftmals können resonante Bereiche aber nicht vollständig vermieden werden und müssen daher in der Triebwerksregelung Berücksichtigung finden. So können ggf. kritische Drehzahlbereiche schnell durchlaufen und dadurch Schaufelanregungen vermieden werden.

Aeroelastik Die Aeroelastik kann als eine Kopplung zwischen der Aerodynamik und Strukturmechanik angesehen werden. Ein sehr wichtiges Phänomen ist das Schaufelflattern. Hierbei handelt es sich um eine aeroelastische Instabilität, die durch die Wechselwirkung einer Schaufelschwingung mit den dadurch hervorgerufenen, instationären Druckverteilungen auf den Schaufeln verursacht wird. Je nach Phasenlage der instationären aerodynamischen Druckverteilungen können sie die auslösende Schwingung dämpfen oder anfachen. Im letzten Fall treten selbsterregte Schwingungen auf (Eigenschwingungen) und die Schaufel wird aerodynamisch instabil. Die hierfür notwendigen Berechnungen sind allerdings mit so hohem numerischem Aufwand verbunden, dass diese oftmals erst nach einer Optimierung für ausgewählte Geometrien durchgeführt werden. Einen umfassenden Überblick zu diesem Thema bietet das Buch *„Introduction to structural dynamics and aeroelasticity“* von Hodges [Hodges & Pierce, 2011].

Aeroakustik Die Aeroakustik beschäftigt sich mit der Entstehung und Ausbreitung aerodynamisch erzeugter Geräusche. Da es sich bei der Aeroakustik um ein rein instationäres Phänomen handelt, ist die Berechnung akustischer Kriterien für eine Optimie-

rung meist zu aufwendig. Daher wird oftmals auf vereinfachte Modelle zurückgegriffen oder versucht direkten Einfluss auf stationäre Strömungsphänomene zu nehmen, von denen man eine starke Lärmentstehung vermutet.

Die hier aufgelisteten Fachdisziplinen stellen nur die wesentlichen Disziplinen dar, die in einer Turbomaschinenoptimierung auftreten können. Dennoch wird ersichtlich, dass durch diese Vielzahl sehr hohe Anforderungen an das Optimierungsverfahren sowie den Anwender gestellt werden.

2.1.2 Allgemeine Begrifflichkeiten zu Optimierungen

Im Folgenden sollen einige Begrifflichkeiten bezüglich Optimierungsverfahren eingeführt werden. Zum besseren Verständnis werden auch einige Beispiele aus dem Bereich der Turbomaschinenauslegung genannt.

Zielfunktion Grundsätzlich ist es das Ziel einer Turbomaschinenoptimierung ein oder mehrere Zielfunktionen zu minimieren oder maximieren, wobei im folgenden grundsätzlich von einer Minimierung ausgegangen wird. Typische Zielfunktionen sind die Erhöhung des Wirkungsgrads, die Erhöhung des Betriebsbereichs, die Verminderung von Lärm etc.

Nebenbedingung Nebenbedingungen werden oftmals als Gültigkeitsintervalle festgelegt in denen die berechneten Größen liegen sollen. Beispielhafte Nebenbedingungen sind die maximale mechanische Spannung einer durch Fliehkraft belasteten Schaufel oder die Einhaltung eines gewissen Massenstrombereichs im aerodynamischen Designpunkt.

Region Of Interest (ROI) Ein für jede Zielfunktion festgelegtes Gültigkeitsintervall. Dies kann sinnvoll sein, wenn nur ein kleiner Bereich des Zielfunktionsraums von Interesse für den Anwender ist. Durch die Festlegung eines solchen Intervalls, kann das Optimierungsverfahren gezielter suchen.

Freie Variablen / freie Parameter Um die Zielfunktion(en) zu optimieren, kann der Optimierungsalgorithmus eine bestimmte Anzahl von freien Variablen oder auch Design Variablen innerhalb ihres Gültigkeitsintervalls verändern. Bei einem Verdichter wären das typischerweise Parameter, welche die Schaufelgeometrien beschreiben. Beispielsweise die Länge oder Dicke einer Schaufel für eine bestimmte radiale Höhe. Nach jetzigem Stand sind Optimierungen in der Größenordnung von bis zu 50-100 freien

Variablen üblich. In sehr komplexen Optimierungen werden bis zu 250 freie Variablen verwendet, siehe [Aulich & Siller, 2011].

Flowparameter Der Begriff „Flowparameter“ gehört grundsätzlich nicht zu den allgemeinen Optimierungsbegriffen, sondern wird speziell im DLR-Institut für Antriebstechnik verwendet. Da der Begriff dennoch eine gewisse Allgemeingültigkeit bekommen hat, wird er in diese Auflistung aufgenommen.

Wird mit einem Satz freier Parameter beispielsweise eine Verdichtergeometrie erzeugt und diese im Anschluss dann mit einem Strömungslöser berechnet, so entstehen durch das nachfolgende Post-Processing eine Vielzahl von Ergebnisgrößen. Oftmals werden hier hunderte bis tausende an Ergebnissgrößen pro Simulation erzeugt. Als Flowparameter bezeichnet man alle Ergebnisgrößen, welche von den genutzten numerischen Verfahren erzeugt und im Optimierungsprozess abgespeichert werden. Aus diesen Flowparametern werden dann die Zielfunktionen und Nebenbedingungen berechnet. Erfahrungsgemäß werden Optimierungsziele und auch Nebenbedingungen während einer laufenden Turbomaschinenoptimierung mehrfach geändert und angepasst. Aus diesem Grund ist es wichtig, möglichst alle wesentlichen Ergebnisgrößen zu speichern.

Member / Sample oder Stützstelle Ein Tupel freier Parameter $(x_1, ..., x_k)$ mit zugeordneten Zielfunktionswerten $(y_1, ..., y_z)$ und den Nebenbedingungen $(w_1, ..., w_c)$ eines Optimierungsproblems mit k-dimensionalen Parameterraum, z-dimensionalen Zielfunktionsraum und c-dimensionalen Nebenbedingungsraum bezeichnet man als Member oder Individuum.

Formulierung Optimierungsproblem Ein Optimierungsproblem lässt sich mit den vorhergehenden Bezeichnungen wie folgt definieren:

$$\min_{x_i \in [a_i, b_i] \forall i \in \{1,...,k\}} \{ \|\vec{y}(\vec{x})\| \,|\, w_j(\vec{x}) \in [c_j; d_j] \, \forall j \in \{1, ..., c\} \}$$

Aus der Formulierung wird ersichtlich, dass dieser Ansatz die Möglichkeit bietet mehrere Ziele $\vec{y}(\vec{x})$ gleichzeitig zu verfolgen. Es muss jedoch eine Norm $\|\vec{y}(\vec{x})\|$ definiert werden. Die Bestimmung einer geeigneten Norm $\|\vec{y}(\vec{x})\|$ ist allerdings sehr schwierig. Eine Möglichkeit stellt die (gewichtete) euklidische Norm dar. Diese hat aber den Nachteil, dass nur derjenige Punkt im Zielfunktionsraum mit der niedrigsten Norm das Optimum darstellt. Um dieses Problem zu umgehen wird in der Regel ein Paretorang-Gütekriterium verwendet, welches zwar nicht die Kriterien einer Norm erfüllt, dafür je-

doch die Möglichkeit bietet, mehrere Ziele gleichzeitig zu optimieren ohne diese gegeneinander zu gewichten. Der Paretorang soll im folgenden erläutert werden.

Paretorang / Paretofront Die Begriffe Dominanz, Paretorang und Paretofront seien im Folgenden kurz definiert, vgl. [Voß & Nicke, 2008]. Man sagt, dass ein bewerteter Member M_1 einen Member M_2 dominiert , falls bzgl. aller Zielfunktionen gilt:

$$M_1 \prec M_2 \Longleftrightarrow y_i\left(M_1\right) \leq y_i\left(M_2\right) \forall i \in \{1, ..., z\} \wedge \exists j \in \{1, ..., z\} : y_j\left(M_1\right) < y_j\left(M_2\right) \quad (2.1)$$

Ein dominanter Member M_1 muss also in jeder Zielfunktion kleiner oder gleich sein als der Member M_2 und zusätzlich in wenigstens einer Zielfunktion kleiner. Mit dem Begriff der Dominanz lässt sich der Paretorang P eines Members definieren:

$$P\left(M\right) \coloneqq \#\left\{ \hat{M} \in M_{all} \middle| \, \hat{M} \prec M \right\} + 1$$

Der Operator $\#$ gibt die Anzahl einer endlichen Menge an.

Die Paretofront PF bezeichnet die Teilmenge aus der Menge aller Member M_{all} mit Paretorang $P = 1$.

$$PF \coloneqq \left\{ \hat{M} \in M_{all} \middle| \, P\left(\hat{M}\right) = 1 \right\}$$

Abbildung 2.1 zeigt exemplarisch die verschiedenen Paretoränge einer fiktiven Optimierung mit zwei Zielfunktionen $fitness_1$ und $fitness_2$. Die Punkte markieren die bereits berechneten Member. An den zwei hervorgehobenen Membern werden die Paretoränge mithilfe von Bereichslinien erklärt. Der eingezeichnete Paretorang 1 Member wird von keinem anderen dominiert. Im Gegensatz dazu wird der gezeigte Paretorang 3 Member von 2 anderen dominiert, da diese in beiden Zielfunktionen besser sind.

Nach dem Abschluss der Optimierung muss der/die verantwortlichen Designer aus der Menge aller optimalen Lösungen (in diesem Fall der Paretofront) eine Konfiguration als Kompromiss zwischen den unterschiedlichen Zielen auswählen.

2.2 Herausforderungen der Turbomaschinenoptimierung

Die Auslegung und damit auch Optimierung von Turbomaschinen stellt eine besondere Herausforderung dar, welche speziell angepasste Optimierungsverfahren notwendig

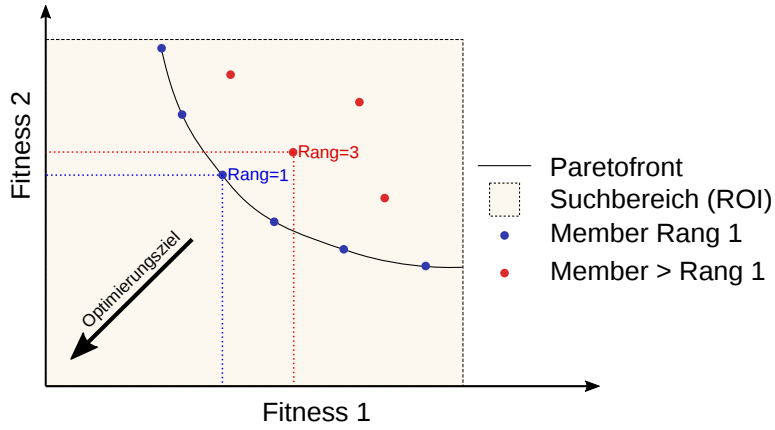


Abbildung 2.1: Paretorang und Paretofront einer Optimierung mit den zwei Zielfunktionen. Quelle: [Reimer, 2016]

macht. In diesem Abschnitt sollen diese Besonderheiten kurz aufgelistet und erläutert werden.

Mehrere Disziplinen Wie bereits im vorherigen Abschnitt gezeigt, ist die Anzahl und Komplexität der benötigten physikalischen Disziplinen sehr hoch. Jede Disziplin benötigt meist eigene Software/Simulationsverfahren und verlangt vom Anwender großes Fachwissen. Aus diesem Grund sind meist mehrere Fachabteilungen an einer Optimierung beteiligt.

Mehrere Zielfunktionen und Nebenbedingungen In der Regel gibt es mehrere Zielfunktionen welche wiederum aus mehreren physikalischen Größen zusammengesetzt werden. Ein typisches Beispiel wäre hier ein über mehrere Betriebspunkte gemittelter Wirkungsgrad. Die unterschiedlichen Zielfunktionen sind meist negativ korreliert. Hier muss also ein Kompromiss gefunden werden. Zusätzlich existiert eine große Anzahl relevanter Nebenbedingungen. Bei Start einer Optimierung werden diese Nebenbedingungen meistens von keinem Member vollständig erfüllt und oftmals ist es auch nicht sicher, ob die Nebenbedingungen überhaupt alle erfüllbar sind.

Diese Punkte führen dazu, dass eine Optimierung anfangs nur sehr grob definiert werden kann und die Ziele und Nebenbedingungen während der laufenden Optimierung umformuliert oder angepasst werden müssen. Das Optimierungsverfahren muss also auch Änderungen von Zielen und Nebenbedingungen während der laufenden Optimierung zulassen.

Gradienteninformation Ein weiterer wichtiger Punkt ist die Verwendung von Gradienteninformationen. Damit sind die partiellen Ableitungen der Zielfunktion $\frac{\partial y_i}{\partial x_j}; i \in$

$\{1, ..., z\}$ oder der Nebenbedingungen $\frac{\partial w_l}{\partial x_j}; l \in \{1, ..., c\}$ nach dem Parameter $x_j; j \in \{1, ..., k\}$ gemeint. Allerdings sind viele der verwendeten Programme und Simulationverfahren sehr komplex und oftmals ist kein Quellcode verfügbar. Dies bedeutet, dass man keine vollständige Gradienteninformationen ohne zusätzliche Simulationen erhalten kann. Einige Programme bieten allerdings die Berechnung der benötigten Gradienten. In diesem Fall sind also zumindest einige partielle Ableitungen verfügbar. Das verwendete Optimierungsverfahren muss also auch unvollständige Gradienteninformationen verarbeiten können.

Weiterhin sind einige der Funktionen oft nicht differenzierbar. Ein häufig verwendetes Beispiel ist die maximale mechanische Spannung einer Verdichterschaufel.

Konvergenz der verwendeten Programme Während einer Optimierung werden zahlreiche Geometrien/Rechennetze erzeugt und danach bspw. mit einem Strömungs- und/oder Strutkur-Löser bewertet. Bei dieser Vielzahl an Geometrien, Rechennetzen und auch Simulationsverfahren können unsinnige Geometrien, schlechte Rechennetze oder aerodynamisch instabile Konfigurationen entstehen. Teils treten auch numerische Probleme auf, die der Anwender nicht direkt nachvollziehen kann. In vielen Fällen benötigen diese Rechnungen dennoch sehr viel Zeit und liefern dann letztlich keine neue Information.

Lange Prozesskettenzeiten Dieser Punkt ist im Bereich der Turbomaschinenoptimierung sehr entscheidend, denn besonders im Bereich der Aerodynamik sind die Simulationsverfahren numerisch sehr aufwendig. Dies gilt besonders bei Optimierungen, da sehr viele dieser komplexen Berechnungen notwendig sind.

Trotz der steigenden Rechnerkapazität ist in diesem Punkt keine Besserung zu erwarten. Dies liegt daran, dass für die Berechnung von Strömungen vereinfachte Modelle verwendet werden und die steigenden Rechenkapazitäten eher für aufwendigere Verfahren eingesetzt werden, um komplexere Strömungsvorgänge simulieren zu können. Beispielsweise die zeitliche Auflösung von Strömungsvorgängen, bessere Modellierung von turbulenten Wirbeln oder größere Teile von Turbomaschinen auf einmal zu simulieren. Die Berechnungsverfahren bieten hier also noch erhebliches Potenzial, welches aufgrund mangelnder Rechnerkapazitäten nicht verwendet werden kann.

Ein Optimierungsverfahren sollte also mit so wenig Berechnungen wie möglich zu einem gewünschten Optimum finden. Es ist also sinnvoll sehr hochwertige Optimierungsverfahren anzuwenden, auch wenn diese selbst viel Zeit in Anspruch nehmen. Meist steht genügend Zeit für diese hochwertigen Verfahren zur Verfügung, da durch die aufwendigen Prozessketten viele Leerlaufphasen entstehen, welche genutzt werden können.

Hochdimensionaler Suchraum Die Problematik von hochdimensionalen Suchräumen wird auch als „Curse of Dimensions“ bezeichnet. Dieser bekannte Begriff entspringt einer Arbeit von Richard Bellman [Bellman, 1972]. Die Dimension des Suchraums ist im Bereich der Turbomaschinenoptimierung sehr hoch. Typische Parameterzahlen findet man bspw. in der Arbeit von Voss mit 50 Parametern [Voß et al., 2014] oder bis zu 230 Parametern in der Arbeit von Siller [Siller et al., 2009]. Berechnet man sich die Anzahl der Ecken eines solchen Hyperwürfels, so bekommt man für eine Dimension von 50 eine Anzahl von ca. 10^{15} Ecken, bei 230 Parametern liegt die Anzahl bei ca. 10^{69}. Einen Suchraum dieser Größe ausreichend zu samplen, ist bei den vorhandenen Prozesskettenzeiten nahezu unmöglich. Der Optimierungsprozess muss aus diesem Grund speziell für eine so große Parameteranzahl ausgelegt sein, um ausgehend von einer Initialisierung möglichst schnell Verbesserungen zu finden.

Behandlung von Nebenbedingungen In AutoOpti können Gültigkeitsintervalle für alle gespeicherten physikalischen Größen und Zielfunktionen angegeben werden. Liegt ein berechneter Member nicht innerhalb dieses Gültigkeitsbereich, so wird der Abstand $d_N(M)$ des Members M zum Gültigkeitsbereich als zusätzliches Gütekriterium herangezogen. Die Definition der Dominanz (siehe Gleichung 2.1) muss daher erweitert werden zu:

$$\begin{aligned} M_1 \prec M_2 \Longleftrightarrow & d_N(M_1) < d_N(M_2) \vee \\ & [d_N(M_1) \leq d_N(M_2) \wedge \\ & (y_i(M_1) \leq y_i(M_2) \, \forall i \in \{1,...,z\} \wedge \exists j \in \{1,...,z\} : y_j(M_1) < y_j(M_2))] \end{aligned}$$

Ein Member M_1 dominiert einen Member M_2, wenn der Abstand zum Gültigkeitsbereich verbessert wird oder der Abstand zum Gültigkeitsbereich gleich bleibt und gleichzeitig die ursprüngliche Definition der Dominanz (siehe Gleichung 2.1) erfüllt wird. Durch diese zusätzlichen Bedingungen ist die Erfüllung der Nebenbedingungen das wichtigste Kriterium für die Bestimmung des Paretoranges.

Diese hier aufgelisteten Probleme, machen die Optimierung von Turbomaschinen zu einer großen Herausforderung und schränken auch die Wahl der Optimierungsverfahren stark ein. Deterministische Verfahren wie z.B. Gradientenverfahren sind in den meisten Turbomaschinenoptimierungen nicht anwendbar, da keine oder nur ein Teil der benötigten Gradienteninformation zur Verfügung steht. Es gibt zwar auch gradientenfreie deterministische Verfahren, allerdings sind diese in hochdimensionalen Räumen sehr ineffizient. Zudem gestaltet sich die parallele Berechnung mehrerer Member bei

diesen Verfahren schwieriger. Im Gegensatz dazu stehen stochastische Verfahren wie z.B. evolutionäre Algorithmen. Diese benötigen nur die Zielfunktionswerte und bieten einen sehr hohen Parallelisierungsgrad.

Da in der vorliegenden Arbeit ein bestehendes Optimierungsverfahren erweitert wird, soll auf eine detaillierte Betrachtung der verschiedenen Verfahren verzichtet werden. Für weitere Informationen bieten folgende Literaturstellen einen umfassenden Vergleich verschiedenster Optimierungsverfahren im Bereich der Turbomaschinen: [Shyy et al., 2001, Müller-Töws, 2000, Shahpar, 2000]

2.3 Automatisierte Optimierung im DLR

In diesem Abschnitt soll die bisher im DLR verwendete Optimierungsstrategie erklärt werden. Der erste Teil erläutert den grundlegenden Ablauf einer Optimierung mit AutoOpti. Im darauffolgenden zweiten Teil wird dann die eingesetzte Ersatzmodellbeschleunigung beschrieben.

2.3.1 Grundlegender Optimierungsprozess

Eine automatisierte Optimierung, wie sie im DLR ausgeführt wird, basiert grundsätzlich auf einer Evolutionsstrategie [Rechenberg, 1973]. Die Evolutionsstrategie ist ein stochastisches Optimierungsverfahren welches kaum Voraussetzungen an die Zielfunktion stellt. Diese muss weder stetig noch differenzierbar sein. Die Abstiegsrichtung wird durch zufällige Tastschritte in der näheren Umgebung bekannter Member bestimmt.

Abbildung 2.2 zeigt den zugrundeliegenden Optimierungsablauf, bei diesem handelt es sich im Wesentlichen um eine Art von Kreisprozess. Innerhalb dieses Kreislaufes gibt es verschiedene Arten von Prozessen, welche über eine asynchrone Kommunikation verfügen.

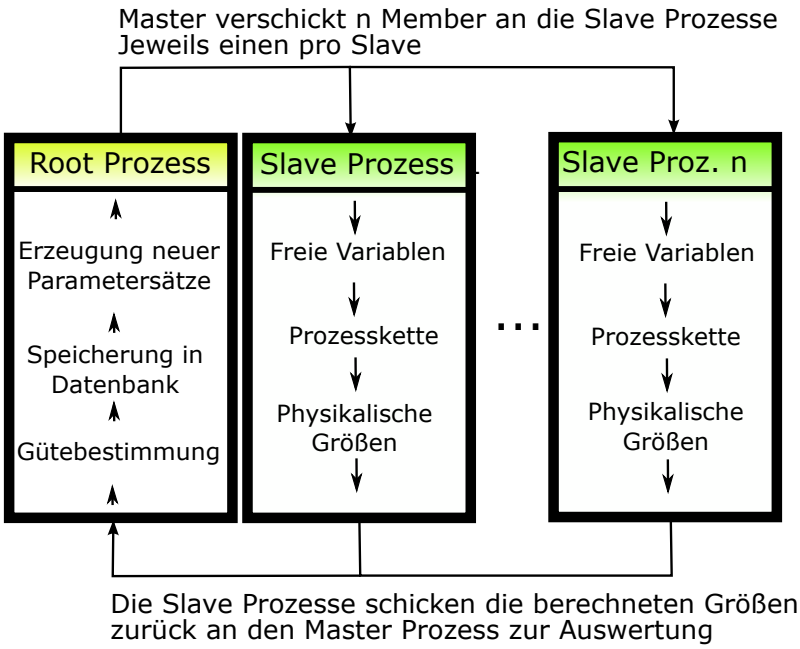


Abbildung 2.2: Prozesskette des genetischen Algorithmus in AutoOpti

Dies bedeutet, dass die Berechnung der Prozesskette und die Verwaltung der Datenbank sowie die Erzeugung neuer Member unabhängig voneinander agieren. Um solch ein asynchrones Verfahren umzusetzen bedient sich AutoOpti eines Master/Slave Ansatzes. Hierfür sind grundlegend zwei verschiedene Arten von Prozessen vorgesehen:

- Der Root-Prozess ist für die Verwaltung und Steuerung der Optimierung zuständig und läuft auf nur einem Rechner.
- Die Slave-Prozesse dienen zur Berechnung der Prozesskette und werden vom Root-Prozess gestartet und gesteuert. Meist wird innerhalb einer Prozesskette eine Geometrie erzeugt und diese dann mit den numerischen Simulationsverfahren bewertet. Damit übernehmen die Slave-Prozesse den numerisch aufwendigeren Teil der Optimierung.

Der in Abbildung 2.2 dargestellte Ablauf einer AutoOpti-Optimierung soll im folgenden genauer erläutert werden.

1. Initialisierungsphase Dieser hier beschrieben initiale Schritt ist in Abbildung 2.2 nicht dargestellt, da diese nur den zyklischen Teil des Optimierungsprozesses beschreibt. Während dieser Initialisierungsphase spielt die Minimierung der Zielfunktion noch keine Rolle. Ziel ist es vielmehr ein möglichst breites Spektrum an sinnvollen Membern zu erzeugen und damit einen möglichst großen Parameterbereich abzudecken. Um dies zu bewerkstelligen bietet AutoOpti diverse Verfahren an:

- Zufällige Variation der freien Variablen
- Latin Hypercube Sampling
- Mutation eines intialen Members

Das erste Verfahren erzeugt zufällige gleichverteilte Parametersätze, wobei die Parameter jeweils einzeln und vollkommen unabhängig voneinander bestimmt werden. Bei einfachen Optimierungen ist diese Art der initialien Erzeugung oftmals ausreichend.

Das Latin-Hypercube-Verfahren (siehe McKay et al. [McKay et al., 1979]) versucht den gegebenen Parameterraum möglichst raumfüllend abzutasten und ist auch in hochdimensionalen Räumen anwendbar.

Beide Verfahren erscheinen allerdings nur dann sinnvoll, wenn zumindest ein gewisser Prozentsatz der erzeugten Member die Prozesskette erfolgreich durchläuft. Bei den hier behandelten Turbomaschinenoptimierungen ist dies oft aber nicht der Fall.

Eine mögliche Lösung bietet das dritte Verfahren. Für dieses Verfahren muss ein initialer Member bekannt sein, welcher die Prozesskette erfolgreich durchläuft. Dieser initiale Parametersatz wird dann immer wieder mutiert und evaluiert und da sich diese Art der Suche nur lokal um den initialen Parametersatz bewegt, kann meist eine hohe Quote von erfolgreichen Prozesskettendurchläufen erreicht werden.

Die Auswahl der hier beschriebenen Verfahren und auch die Anzahl der zu erzeugenden Member, muss der jeweilige Anwender treffen. Erfahrungsgemäß ist es sinnvoll zuerst ein raumfüllendes Verfahren zu wählen und nur bei zu niedriger Erfolgsquote auf ein lokales Verfahren zu wechseln.

2. Erzeugung neuer Parametersätze Der eigentliche Optimierungsprozess startet nach der Initialisierungsphase mit der bereits erzeugten initialen Datenbank. Der Root Prozess erzeugt einen oder mehrere vielversprechende Parametersätze auf Basis der bereits vorhandenen Member. Für die Erzeugung dieser neuen Parametersätze gibt es zahlreiche Verfahren, welche die Geschwindigkeit und den Konvergenzverlauf der Optimierung maßgeblich beeinflussen. AutoOpti nutzt hauptsächlich eine ersatzmodelbeschleunigte-Evolutionsstrategie zur Erzeugung neuer Member. Diese Strategie zur Erzeugung vielversprechender Parametersätze hat sich in der Vergangenheit als sehr effizient und flexibel herausgestellt und wird in Kapitel 2.3.2 genauer erläutert.

3. Verschicken neuer Parametersätze an die Slave-Prozesse Nachdem der Root Prozess einen vielversprechenden Parametersatz generiert hat, muss dieser zur Berechnung an einen freien Slave-Prozess verschickt werden. Die Kommunikation findet

bei AutoOpti über ein gemeinsames Dateisystem statt und beinhaltet im Wesentlichen nur einen Satz freier Variablen pro Slave-Prozess.

4. Slave-Prozesse berechnen die Prozesskette Nachdem der Root-Prozess die neuen Parametersätze an die noch freien Slave-Prozesse verschickt hat, können diese die Prozesskette durchlaufen. Bei erfolgreicher Ausführung der Prozesskette müssen alle relevanten Ergebnisse in einer Datei bereit gestellt werden. Die gesammelten Ergebnissen werden vom Slave-Prozess eingelesen und als Flowparameter abgespeichert.

5. Slave-Prozesse senden die Ergebnisse an den Root-Prozess Die Slave-Prozesse schicken die berechneten Flowparameter zurück an den Root-Prozess. Die Kommunikation findet auch hier über ein gemeinsames Dateisystem statt.

6. Auswertung der Ergebnisse durch den Root-Prozess Aus den vom Slave bereitgestellten Flowparametern, berechnet der Root Prozess nun die Zielfunktionen, Nebenbedingungen und daraus auch das Gütekriterium. Die Definition der Zielfunktionen und Nebenbedingungen ist in einer Konfigurationsdatei festgelegt und kann vom Anwender jederzeit angepasst werden. Ein möglicher Gütewert wäre hier der Paretorang (siehe Kapitel 2.1.2). Nachdem diese Werte bestimmt worden sind, trägt der Root-Prozess den Member gemäß seiner Güte in die Datenbank ein. Danach springt der Root-Prozess wieder zu Punkt 2.

2.3.2 Ersatzmodellbeschleunigung

Im vorhergehenden Abschnitt wurde der grundlegende Ablauf einer Optimierung mit AutoOpti beschrieben. Der Punkt der Erzeugung neuer Parametersätze wurde allerdings nur kurz erwähnt und soll im Folgenden genauer erklärt werden. Eine weit verbreitete Möglichkeit der Erzeugung neuer Member ist die Verwendung einer Evolutionsstrategie. Wie bereits beschrieben, verwendet die Evolutionsstrategie zufallsbasierte Tastschritte in der unmittelbaren Nähe bekannter Member. Typische Operatoren um die zufallsbasierte Änderung von Parametern vorzunehmen sind z.B. Mutation oder Kreuzung von Membern. Dies führt jedoch dazu, dass die erreichten Verbesserungen eher klein sind. Um größere Fortschritte zu erreichen, sind dafür sehr viele Tastschritte und damit Funktionsauswertungen (z.B. Strömungslösungen) notwendig. In der Kombination mit den großen Parameterräumen und langen Prozesskettenzeiten wäre eine solche Optimierung nicht mehr rentabel. Um den Prozess zu beschleunigen, versucht man die Membererzeugung des Root Prozesses (siehe Abbildung 2.2) zu verbessern.

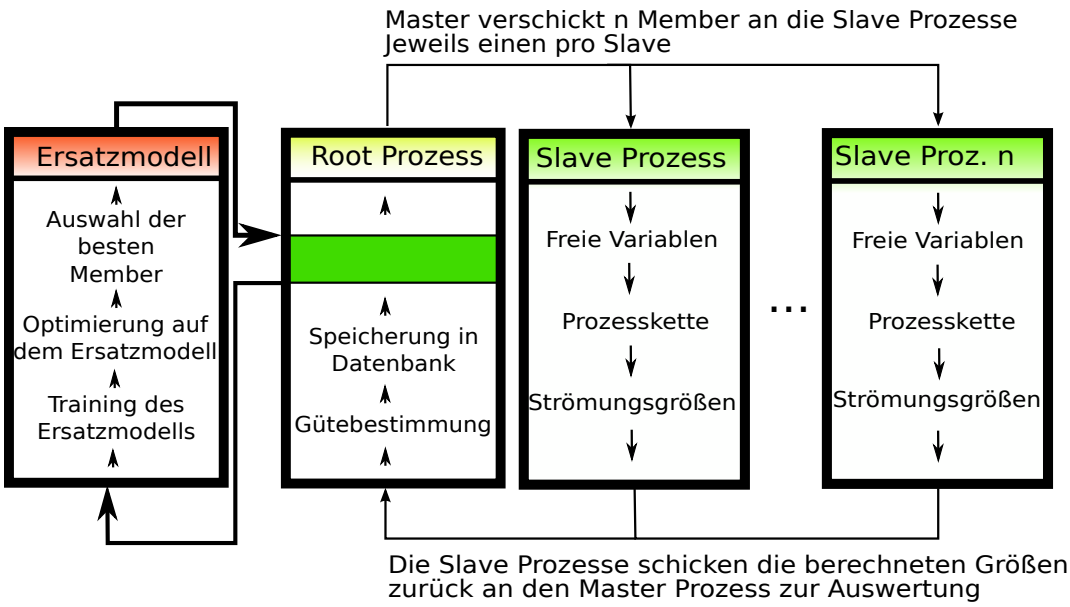


Abbildung 2.3: Nutzung von Ersatzmodellen im Optimierungsprozess

Insbesondere versucht man Member zu erzeugen, die eine möglichst große Verbesserung der Zielfunktionen aufweisen und die zusätzlich alle Nebenbedingungen erfüllen.

Ein weit verbreitetes Mittel hierfür sind Ersatzmodelle oder auch Metamodelle. Ersatzmodelle wie z.B. das Kriging Verfahren approximieren oder interpolieren vorhandene Daten. Die Ersatzmodelle werden innerhalb einer Optimierung dafür verwendet, Vorhersagen der Zielfunktionen und Nebenbedingungen durchzuführen und damit besonders aussichtsreiche Parametersätze zu finden. Um ein solches Ersatzmodell verwenden zu können, muss zuerst ein Training mit den vorhandenen Daten durchgeführt werden. Während eines solchen Trainings werden meistens ersatzmodellspezifische Parameter iterativ bestimmt. Dies ist in der Regel auch der zeitaufwendigste Teil bei der Verwendung von Ersatzmodellen. Nach dem Training können Funktionswerte an beliebigen Stellen vorhergesagt werden. Wobei die Vorhersage von Funktionswerten durch ein Ersatzmodell ist meist um einige Größenordnungen schneller als eine Funktionsauswertung durch die Prozesskette.

Die trainierten Ersatzmodelle werden dann herangezogen, um mit den Vorhersagen neue Member zu erzeugen. Die Erzeugung kann dann z.B. durch eine kleine Optimierung auf den Ersatzmodellen erfolgen. Der aussichtsreichste Member wird anschließend mit der echten Prozesskette nachgerechnet und dann in die Optimierungsdatenbank eingetragen. Durch dieses Verfahren lässt sich der Optimierungsfortschritt stark beschleunigen. Wie stark, hängt von der Güte und Leistungsfähigkeit des verwendeten Modells ab. Mithilfe von Abbildung 2.3 soll der in AutoOpti verwendete Prozess zur Nutzung von Ersatzmodellen genauer beschrieben werden.

1. Training der Ersatzmodelle Die in AutoOpti verwendeten Ersatzmodelle müssen vor der Verwendung trainiert werden. Das Training wird oftmals auch als Lernverfahren bezeichnet. Dieses dient zur Bestimmung von ersatzmodellspezifischen Parametern, dabei kann es sich z.B. um Korrelationslängen oder Gewichtungen handeln. Die Ersatzmodellparameter werden dann mithilfe der vorhandenen Daten eingestellt. Es werden in AutoOpti für jede Zielfunktion und jede Nebenbedingung jeweils ein Ersatzmodell trainiert. Bei zusammengesetzten Zielfunktionen wird für jeden in der Zielfunktion verwendeten Flowparameter ebenfalls ein Ersatzmodell trainiert.

Meistens sind diese Trainingsverfahren aufwendig und benötigen je nach Typ des Ersatzmodells und der Anzahl an Stützstellen viel Zeit. Nach dem Training können die Ersatzmodelle für Vorhersagen der trainierten Funktion verwendet werden, wobei die Vorhersagen erfahrungsgemäß sehr viel schneller sind als das Training.

Wie oft ein Ersatzmodell während einer Optimierung trainiert werden muss, hängt stark vom verwendeten Ersatzmodell ab. Bei dem in AutoOpti hauptsächlich verwendeten Kriging-Verfahren kann man jedoch sagen, dass bei ausreichender Datenlage irgendwann kein Training mehr vonnöten ist.

Das Training kann entweder synchron durch den Root-Prozess geschehen oder asynchron durch einen eigenen Prozess. Bei der synchronen Variante trainiert der Root-Prozess die Ersatzmodelle und ist in dieser Zeit blockiert. Wird in dieser Zeit ein Slave-Prozess frei, so muss dieser warten bis das Training beendet ist.

Bei der asynchronen Variante wird das Training in einem externen Prozess gestartet und der Root-Prozess verwendet für die benötigten Vorhersagen dann immer die jeweils letzten trainierten Modelle. Die asynchrone Variante stellt die schnellere Methode dar und bietet auch eine höhere Stabilität, da der Root-Prozess durch fehlerhafte Trainings nicht beeinflusst wird. Allerdings ist der Verwaltungsaufwand etwas höher und der externe Trainingsprozess muss permanent überwacht werden, da der Root-Prozess keine auftretenden Fehler bemerkt.

Die in AutoOpti verwendeten Ersatzmodelle sind bayesisch trainierte Neuronale Netzwerke [Mackkay, 1991] und verschiedene Kriging-Verfahren (siehe Kapitel 4).

2. Optimierung auf den Ersatzmodellen Ist das Training abgeschlossen, wird eine eigene Optimierung auf den Ersatzmodellen gestartet, wobei die Ersatzmodelle den Teil der echten Prozesskette durch Vorhersagen ersetzen. Da die Funktionsauswertungen auf den Ersatzmodellen deutlich schneller sind, als die Prozesskette selbst, geht dieser Schritt relativ schnell. Ziel dieser Optimierung auf den Ersatzmodellen ist es, einen vielversprechenden Parametersatz zu finden, welcher die Zielfunktionen möglichst stark verbessert und zudem eine hohe Wahrscheinlichkeit besitzt alle Nebenbedingungen einzuhalten. Für die Optimierung auf den Ersatzmodellen, bietet AutoOpti

mehrere Verfahren an. Aufgrund der sehr schnellen Vorhersagen der Ersatzmodelle, haben sich evolutionäre Algorithmen als besonders robust und effizient erwiesen und finden daher hauptsächlich Anwendung. Da die Ersatzmodelle selbst differenzierbar sind, bietet AutoOpti auch die Option gradientenbasierte Optimierungsverfahren anzuwenden. Allerdings bieten diese keine wirklichen Vorteile, da der Geschwindigkeitsvorteil bei den schnellen Ersatzmodellvorhersagen praktisch nicht ins Gewicht fällt und ein evolutionärer Algorithmus eine „globalere“ Suche auf den Ersatzmodellen ermöglicht.

3. Auswahl der vielversprechendsten Member Meistens werden in Turbomaschinenoptimierungen mehrere Zielfunktionen und Nebenbedingungen verwendet. Dies führt dazu, dass bei der Optimierung auf den Ersatzmodellen mehrere vielversprechende Parametersätze generiert werden. Bei der Verwendung eines Gütekriteriums wie dem Paretorang würden viele Parametersätze mit dem Paretorang Eins erzeugt werden und könnten untereinander nicht weiter gewichtet werden. Es ist also ein Gütekriterium nötig, welches mehrere Zielfunktionen und Nebenbedingungen vereinen und auch den zu erwartenden Optimierungsfortschritt zu einer bestehenden Paretofront genauer quantifizieren kann. In AutoOpti findet der „Volumenzugewinn“ als Gütekriterium Anwendung und wird im nächsten Abschnitt vorgestellt. Der Root-Prozess wählt nun den vielversprechendsten Member aus und schickt diesen zur Berechnung an einen freien Slave Prozess. Der Slave Prozess berechnet dann die echte Zielfunktion, sodass diese dann auch mit der Vorhersage des Ersatzmodells verglichen werden kann.

Volumenzugewinn

Wie bereits erwähnt, ist für die Optimierung auf den Ersatzmodellen ein Gütekriterium nötig, welches den Optimierungsfortschritt für Mehrzieloptimierungen genauer quantifizieren kann als bspw. der Paretorang. Das in AutoOpti verwendete Kriterium ist der Volumenzugewinn oder auch „Volume-Gain“.

Abbildung 2.4 zeigt den Volumenzugewinn einer beispielhaften Optimierung mit zwei Zielfunktionen. Gezeigt wird eine Paretofront, dargestellt durch die blauen Punkte. Die zwei grünen Punkte beschreiben neu hinzugekommene Member, auf die das Gütekriterium angewandt werden soll. Beide würden Paretorang Eins bekommen und wären damit gleichwertig. Betrachtet man nun die in grau dargestellte Fläche zwischen der aktuellen Paretofront und den neu hinzugekommenen Membern, so beschreibt diese den Volumenzugewinn. Das Kriterium lässt sich auch für einen höherdimensionalen Zielfunktionsraum anwenden und quantifiziert so den Optimierungsfortschritt für eine Mehrzieloptimierung im Bezug auf eine bestehende Paretofront. Der Volumenzugewinn kann auch in einer Optimierung auf den Ersatzmodellen angewendet werden, wobei für die Berechnung die Vorhersagen der Ersatzmodelle herangezogen werden.

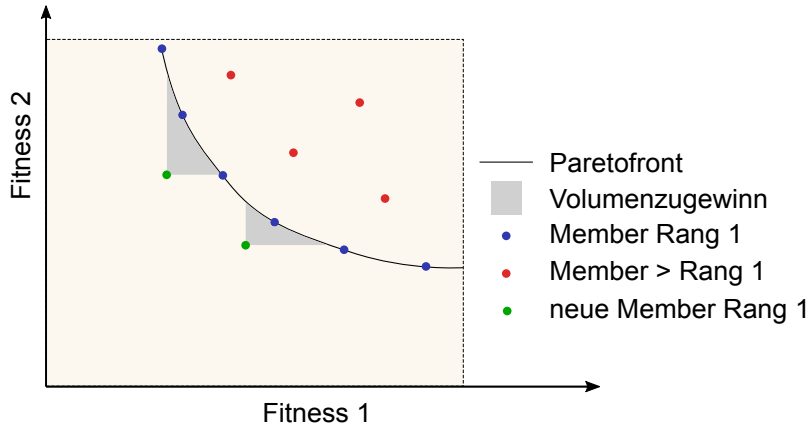


Abbildung 2.4: Exemplarische Darstellung des Volumenzugewinns anhand einer Optimierung mit zwei Zielfunktionen. Quelle: [Reimer, 2016]

Erwarteter Volumenzugewinn

AutoOpti verwendet statistische Ersatzmodelle. Dies bedeutet, dass die Ersatzmodelle statt dem reinen Funktionswert eine statistische Verteilung vorhersagen. Als Verteilungsfunktion wird eine Normalverteilung angenommen. Dies trifft in einigen Fällen zwar nicht zu, stellt in der Regel aber eine vernünftige Annahme dar und ist vom numerischen Aufwand noch im vertretbarem Rahmen. Die Ersatzmodelle sagen also für einen unbekannten Ort $\vec{x} \in \mathbb{R}^k$ einen Erwartungswert $y^*(\vec{x})$ und eine Standardabweichung $\sigma^*(\vec{x})$ der jeweiligen Funktion voraus. Die Berechnung des Volumenzugewinns für einen vorhergesagten Member muss also in diesem Fall die Verteilung berücksichtigen. Abbildung 2.5 zeigt eine exemplarische Optimierung mit zwei Zielfunktionen und einer mit blauen Punkten dargestellten Paretofront. Die beiden grünen Punkte stellen Vorhersagen aus einer Optimierung auf den Ersatzmodellen dar. Die Fehlerbalken sollen die vorhergesagte Standardabweichung andeuten. Mithilfe der Standardabweichung lässt sich nun auch die Unsicherheit in der Vorhersage der Ersatzmodelle berücksichtigen und eine erwartete Verbesserung I und die dazugehörige Wahrscheinlichkeit überhaupt eine Verbesserung zu erzielen. Das in AutoOpti verwendete Erwartete Volumenzugewinn Kriterium EVG eines Members M beschreibt das Produkt aus beiden Werten:

$$EVG(M) = P(M) * I(M)$$

Bei einer Optimierung auf den Ersatzmodellen, wie sie in diesem Kapitel beschrieben wurde, wird versucht dieses Kriterium zu maximieren.

Ein großer Vorteil bei Berücksichtigung der Unsicherheiten der Vorhersage ist das daraus resultierende explorative Verhalten der Optimierung. Ein Optimierungsal-

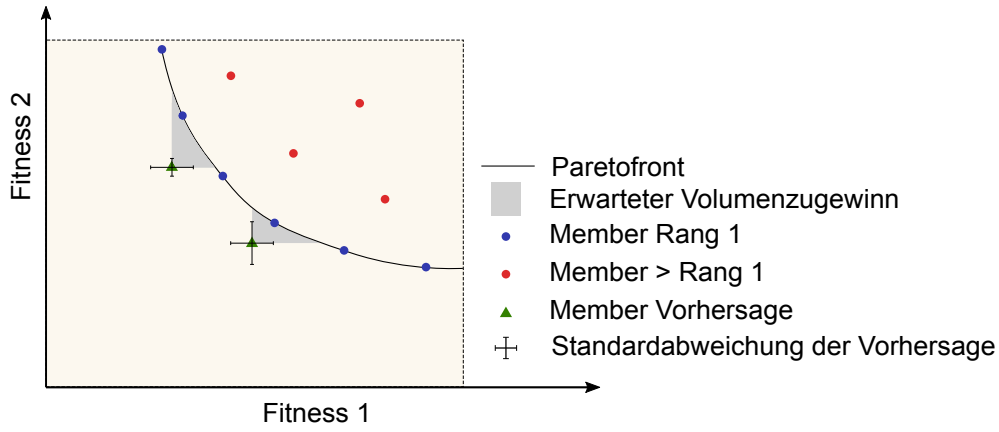


Abbildung 2.5: Exemplarische Darstellung des erwarteten Volumenzugewinns anhand einer Optimierung mit zwei Zielfunktionen und drei Ersatzmodellvorhersagen. Quelle: [Reimer, 2016]

gorithmus, welcher nur die Vorhersage des Funktionswert berücksichtigt, besitzt in der Regel ein rein exploitatives Verhalten. Der Algorithmus sucht also direkt nach einem Optimum und versucht nicht unbekannte Räume auf Optima zu untersuchen (siehe [Forrester et al., 2008])Der EVG Algorithmus bietet hier einen sehr guten Kompromiss zwischen explorativen und expoitativen Verhalten.

Auf die genaue mathematische Herleitung und auch die Behandlung von Nebenbedingungen soll in dieser Arbeit nicht weiter eingegangen werden. Es sei hierfür auf [Aulich & Siller, 2011, Jones et al., 1998, Jones, 2001, Keane, 2006] verwiesen.

3 Multifidelity Optimierungsstrategie Turbomaschine

Im vorhergehenden Kapitel wurde die bisher verwendete Optimierungstrategie im DLR beschrieben. Das folgende Kapitel beschreibt eine Erweiterung dieser Strategie basierend auf der Nutzung von verschiedenen Gütestufen. Diese Art der Optimierungsstrategie wird Multifidelity-Optimierung genannt.

Im ersten Teil dieses Kapitels sollen allgemeine Grundlagen, sowie die Vorteile und Grenzen einer solchen Strategie ausgearbeitet werden. Darauf aufbauend wird im zweiten Teil dieses Kapitels die konkrete Umsetzung innerhalb der Optimierungssoftware AutoOpti beschrieben.

3.1 Gütestufen in der Turbomaschinenauslegung

Um die Idee hinter einem Multifidelity-Verfahren zu verstehen, ist es sinnvoll einen exemplarischen Ablauf einer manuellen Turbomaschinenauslegung zu zeigen. Denn erfahrungsgemäß werden innerhalb von Turbomaschinenauslegungen immer mehrere Gütestufen verwendet. Diese gewinnbringend in einem Optimierungsverfahren zu verwenden erscheint daher nur sinnvoll. Um dies zu verdeutlichen, soll folgend die exemplarische Auslegung eines Turbomaschinenverdichters beschrieben werden (vgl. [Domercq, 2006]).

Die grundlegende Vorgehensweise bei einer solchen Auslegung ist es, die Dimensionalität und die Komplexität der Problemstellung stetig zu erhöhen.

Zu Anfang einer Auslegung sind meist viele Randbedingungen durch die geplante Anwendung der Maschine festgelegt. Bei einem Verdichter können dies Anforderungen an den Betriebsbereich, bestimmte Massenströme und geforderte Druckverhältnisse sein. Mit diesen ersten Randbedingungen können dann die ungefähren geometrischen Ausmaße und auch die benötigte Stufenanzahl grob geschätzt werden. Aufbauend auf diesen ersten Schätzungen wird dann versucht den rotationssymmetrischen Strömungskanal zu bestimmen, hierfür finden meist sogenannte 2D-Throughflow-

Verfahren Anwendung. Diese berechnen ein Meridianströmungsfeld auf der sogenannten S2-Ebene (siehe [Wu, 1952]) und stellen für die Auslegung von mehrstufigen Strömungsmaschinen immer noch ein zentrales Element dar ([Willburger, 2011]). Ein sehr großer Vorteil dieser Verfahren ist es, dass noch keine genaue Kenntnis über die Geometrie der Verdichterschaufeln benötigt wird. Diese werden meist nur über aerodynamische Kenngrößen beschrieben. Typisch sind hier die Umlenkung und Totaldruckverluste einer Schaufel. Ein solches Throughflow-Verfahren ist in der Arbeit von Mönig et al. beschrieben [Mönig et al., 2000] und basiert grundlegend auf der Arbeit von Howard und Gallimore [Howard & Gallimore, 1992].

Nachdem der Strömungskanal und auch die aerodynamischen Kenngrößen für die Schaufelreihen bestimmt worden sind, wird versucht Schaufelgeometrien zu finden, welche diese vorher bestimmten Kenngrößen erfüllen. Hierfür finden S1-Schnittverfahren wie z.B. MISES [Drela & Youngren, 2008] Anwendung. Dabei handelt es sich um ein gekoppeltes Euler-Grenzschichtverfahren welches Profilumströmungen simuliert.

Ist dieser Schritt ebenfalls abgeschlossen, kann die entstandene Geometrie durch ein 3D-Navier-Stokes-Simulationsverfahren aerodynamisch bewertet werden. Hierdurch ist man erstmals in der Lage komplexere dreidimensionale Strömungsphänomene aufzulösen. Wobei für diese 3D-Simulationsverfahren ebenfalls unterschiedliche Modelle und damit auch Genauigkeiten existieren. Beispielsweise können diese Verfahren eine Strömung zeitlich aufgelöst oder zeitlich gemittelt wiedergeben. Grundlegend gilt allerdings, dass eine höhere Genauigkeit meist durch einen enormen Anstieg der benötigten Rechenleistung bezahlt werden muss. Innerhalb einer Optimierung muss aus diesem Grund immer ein Kompromiss zwischen Genauigkeit und Geschwindigkeit eingegangen werden.

In der Regel gibt es eine Low-Fidelity- und eine High-Fidelity-Prozesskette. Die Low-Fidelity-Prozesskette ist zwar sehr schnell berechnet, jedoch ist diese normalerweise auch mit einer größeren Ungenauigkeit behaftet. Die High-Fidelity Prozesskette hingegen ist deutlich aufwendiger, dafür aber genauer. Ein mögliches Beispiel wäre die 3D-Optimierung eines Triebwerksverdichters, mit dem Ziel bei gleichbleibenden Druckverhältnis den Wirkungsgrad zu erhöhen. Eine solche Optimierung erfordert im Normalfall eine Voruntersuchung des Rechennetzes. Auf der einen Seite soll das Rechennetz so grob wie möglich sein, um möglichst viel Zeit einzusparen. Auf der anderen Seite muss es aber noch fein genug sein, um die zu untersuchenden Phänomene mit ausreichender Genauigkeit abbilden zu können.

Wünschenswert wäre es allerdings, Rechennetze mit verschiedener Güte in einer Optimierung zu verwenden und zwar so, dass das Optimierungsverfahren maximale Zeit einspart bei minimalem Verlust an Information. Beispielsweise könnte man ein grobes

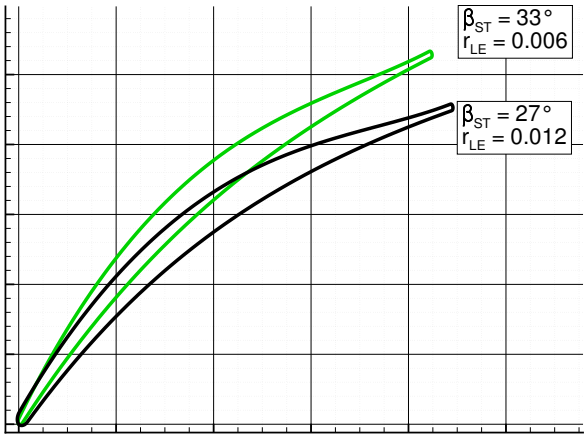


Abbildung 3.1: Beispielhafte Geometrischen Variationen des DCA ähnlichen Profils

(Low-Fidelity) und ein feines Netz (High-Fidelity) zur Verfügung stellen und würde erwarten, dass der grobe Verlauf der Zielfunktion (bspw. der Wirkungsgrad) bereits in den Low-Fidelity-Daten enthalten ist. Diese Information möchte man sich natürlich zunutze machen, was mit dem bereits beschrieben CO-Kriging Verfahren möglich ist. Eine große Schwierigkeit innerhalb der Optimierung besteht allerdings darin zu entscheiden, wann welche Prozesskette verwendet wird. Im zweiten Abschnitt dieses Kapitels sollen verschiedene Lösungen für dieses Problem vorgestellt werden.

3.1.1 Beispiel für verschiedene Gütestufen

Um einen Eindruck von den unterschiedlichen Gütestufen innerhalb aerodynamischer Simulationsverfahren zu bekommen, soll folgend ein einfaches aber aussagekräftiges Beispiel gezeigt werden. Dafür wird ein DCA ähnliches Verdichterprofil (Double Circular Arc, siehe [Lieblein & Seymour, 1955]) mit einem 2D-Strömungslöser vom MIT namens MISES (siehe [Drela & Youngren, 2008]) berechnet. Es sollen geometrische Variationen von diesem Profil erzeugt werden und diese dann mit verschiedenen Gütestufen aerodynamisch simuliert werden. Die geometrischen Variationen werden über zwei Parameter realisiert. Zum einen der Staffelungswinkel β_{st}, der in diesem Fall als eine Festkörperdrehung des Profils verstanden werden kann und zum anderen der Vorderkantenradius r_{LE}. In Abbildung 3.1 werden zwei dieser möglichen Variationen dargestellt.

Um den Einfluss der Rechengitterauflösung auf verschiedene Strömungsgrößen abschätzen zu können, werden in realen Anwendungsfällen häufig Netzstudien durchgeführt. Innerhalb dieses Beispiels sollen daher zwei verschiedene Netzauflösungen miteinander verglichen werden. Ein grobes Netz mit ca. 920 Zellen und ein feines Netz 16680 Zellen. Die Berechnung der niedrigen Gütestufe ist ca. 10x schneller als die

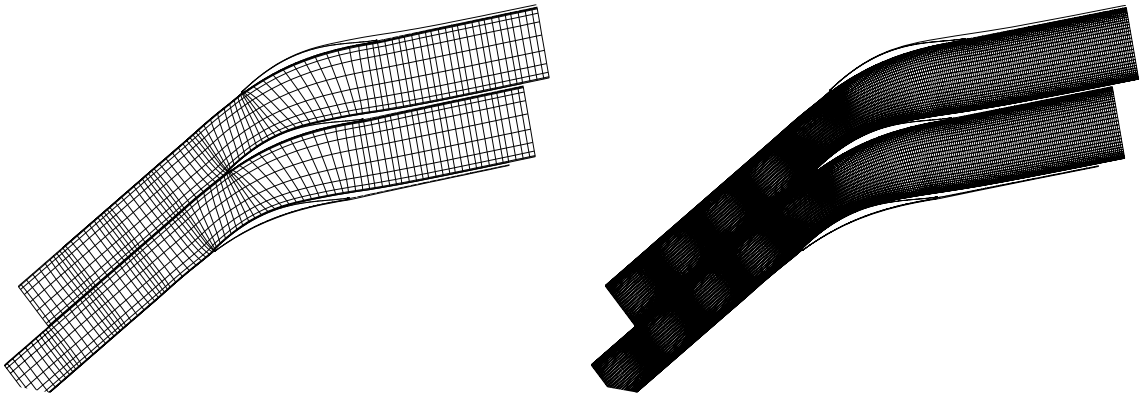


Abbildung 3.2: Darstellung der untersuchten Rechengitterauflösungen

Name	Minimal	Maximal
Anströmmachzahl	0.9	0.9
Anströmwinkel	141°	141°
Stromröhrenkontraktion	1.1	1.1
Staffelungswinkel	27°	32°
Vorderkantenradius	0.006	0.011

Tabelle 3.1: Randbedingungen für den Testfall

Berechnung der höheren Stufe. Die beiden Rechennetze werden in Abbildung 3.2 für eine mögliche Geometrie dargestellt.

Die betrachtete Strömungsgröße ist der massenstromgemittelte Totaldruckverlustbeiwert ω (siehe Anhang A.1).

In Tabelle 3.1 werden die wichtigsten Randbedingungen für den Testfall aufgelistet.

Abbildung 3.3 zeigt die Totaldruckverluste der verschiedenen Gütestufen über dem variierten Staffelungswinkel für jeweils zwei verschiedene Vorderkantenradien. Grundlegend besitzen alle vier Kurvenverläufe eine starke Ähnlichkeit, insbesondere bei kleinen Vorderkatenradien. Bei r=0.011 hat sich das Minimum der Verlustpolaren um ca. 1° verschoben. Außerdem besitzt die Verlustpolare bei höheren Staffelungswinkeln einen steileren Anstieg. Insbesondere die Verschiebung des Minimums könnte in einer Optimierung zu einer anderen Geometrie führen. Dennoch bietet die niedrigere Gütestufe einen sehr hohen Informationsgehalt bei deutlich kürzerer Laufzeit. Im Vergleich zu einer realen Turbomaschinenoptimierung ist das hier gezeigte Beispiel deutlich simpler, da nur zwei Geometrieparameter variiert wurden. Dennoch zeigt das Beispiel die Ähnlichkeit der Gütestufen, welche sich innerhalb eines Multifidelity Verfahren sehr effizient nutzen lassen.

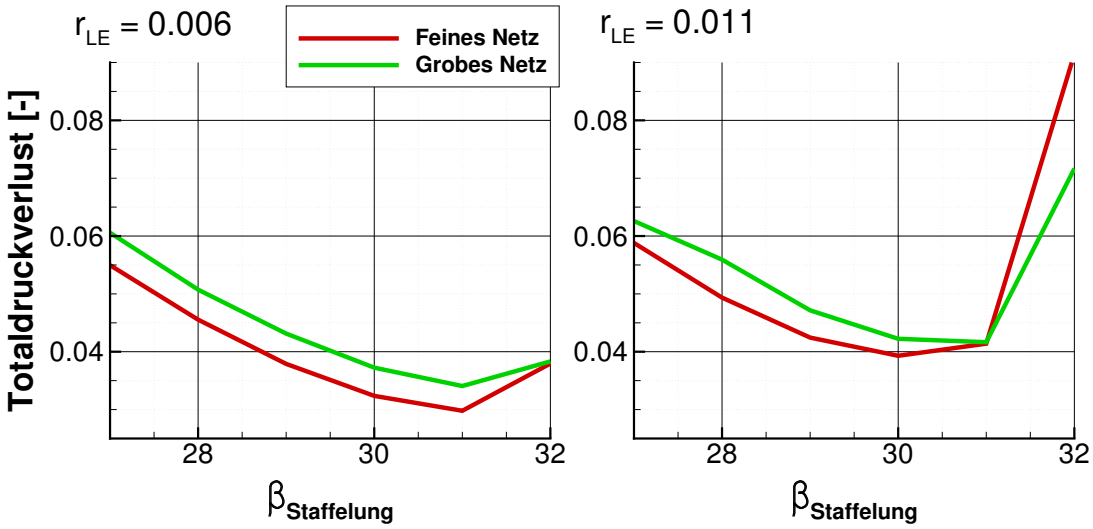


Abbildung 3.3: Verlustpolaren für zwei verschiedene Vorderkantenradien

3.1.2 Verschiedene Simulationsverfahren

Neben der Verwendung von verschiedenen Gütestufen innerhalb eines Simulationsverfahren, lassen sich auch unterschiedliche Simulationsverfahren koppeln. Meistens sind die zeitlichen Unterschiede für solch einen Fall stärker ausgeprägt. Allerdings ist eine gemeinsame Parametrisierung unerlässlich, was bei unterschiedlicher Dimensionalität der Simulationsverfahren problematisch sein kann. Zudem können die Funktionale auch so große Unterschiede aufweisen, dass die Daten nur sehr schwach korreliert sind und der Einsatz eines Mutlifidelity-Verfahrens dann nicht mehr lohnenswert ist.

Schnös et al. [Schnös & Nicke, 2017] stellt in seiner Arbeit einen Vergleich zwischen zwei Simulationsverfahren für dieselbe Profilgeometrie an. Als Beispiel wird ein Vergleich zwischen dem bereits beschriebenen 2D-Euler-Grenzschichtverfahren Mises und einem vollwertigen 3D-Navier-Stokes-Strömungslöser vorgestellt. Als 3D Strömungslöser wird in diesem Fall das im DLR entwickelte TRACE-Verfahren (siehe [Kügeler, 2005, Nürnberger, 2004]) genutzt. Verglichen werden die Verlustpolaren eines Mittelschnittprofils des RIG250 Verdichters (siehe [Schönweitz et al., 2013]). Abbildung 3.4 zeigt die Verlustpolare und den berechneten Abströmwinkel der beiden Verfahren. Während die Verlustpolare eine sehr ähnliche Charakteristik aufweist, sind beim Verlauf des Abströmwinkels deutliche Unterschiede im berechneten Verhalten zu erkennen.

Das vorangestellte Beispiel lässt keine allgemeine Aussage über die Nutzbarkeit unterschiedlicher Simulationsverfahren innerhalb von Multifidelity Optimierungen zu, da die Transferfunktionen zwischen den Gütestufen sehr unterschiedlich ausfallen können und vom jeweiligen Anwendungsfall und den verwendeten Simulationsverfahren

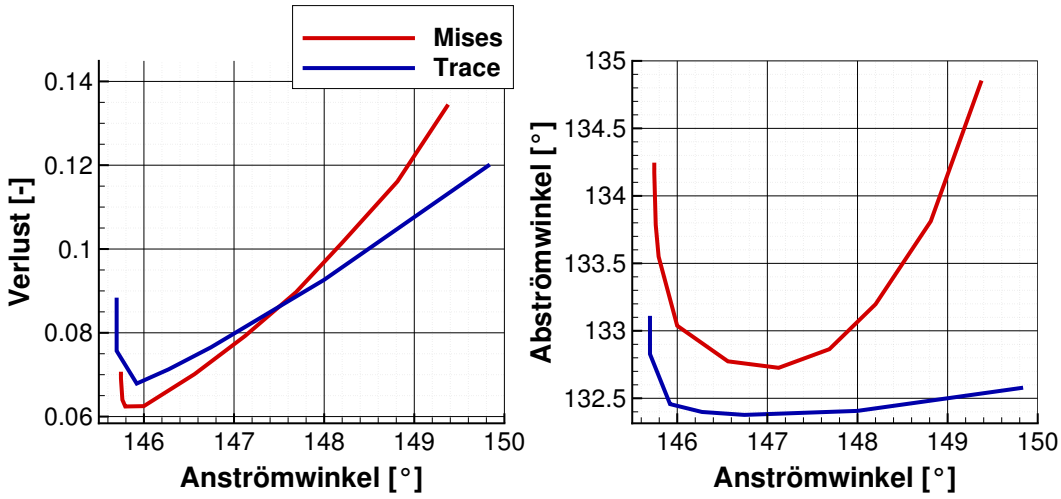


Abbildung 3.4: Vergleich zwischen dem 2D-CFD Verfahren Mises und dem 3D-CFD Verfahren TRACE anhand eines transsonischen Verdichterprofils

abhängen. Der Einsatz von unterschiedlichen Simulationsverfahren innerhalb einer Multifidelity-Optimierung wird in den meisten Fällen aber bereits an der gemeinsamen Parametrisierung scheitern.

Die Verwendung unterschiedlicher Gütestufen innerhalb eines Simulationsverfahrens scheint für ein Multifidelity Verfahren in den meisten Fällen der geeignetere Weg zu sein. Da in diesem Fall meist eine gemeinsame Parametrisierung besteht und auch die Erzeugung verschiedener Gütestufen in der Regel mit weniger Aufwand verbunden ist.

Es sollte aber für jeden einzelnen Fall eine Abwägung gemacht werden, inwiefern die verwendeten Gütestufen für einen Einsatz innerhalb einer Multifidelity-Optimierung geeignet sind.

3.2 Multifidelity-Optimierungsstrategie in AutoOpti

Innerhalb dieses Abschnitts soll die in AutoOpti umgesetzte Multifidelity-Optimierungsstrategie beschrieben werden. Als Basis für diese Strategie wird das in Kapitel 2.3 beschriebene Verfahren verwendet. Die notwendigen Änderungen an dem grundlegenden Optimierungs-Prozess und der Datenhaltung werden im folgenden Unterkapitel 3.2.1 erläutert. Darauffolgend wird in Unterkapitel 3.2.2 ein besonderes Augenmerk auf die Entscheidungsfunktion gelegt. Diese Funktion soll während einer laufenden Optimierung automatisiert entscheiden können, mit welcher Gütestufe der nächste Member berechnet werden soll. Da die Entscheidungsfunktion einen sehr großen Einfluss auf die Effizienz einer Multifidelity-Optimierung hat, wird innerhalb dieses Kapitels ein effizientes automatisiertes Verfahren vorgestellt.

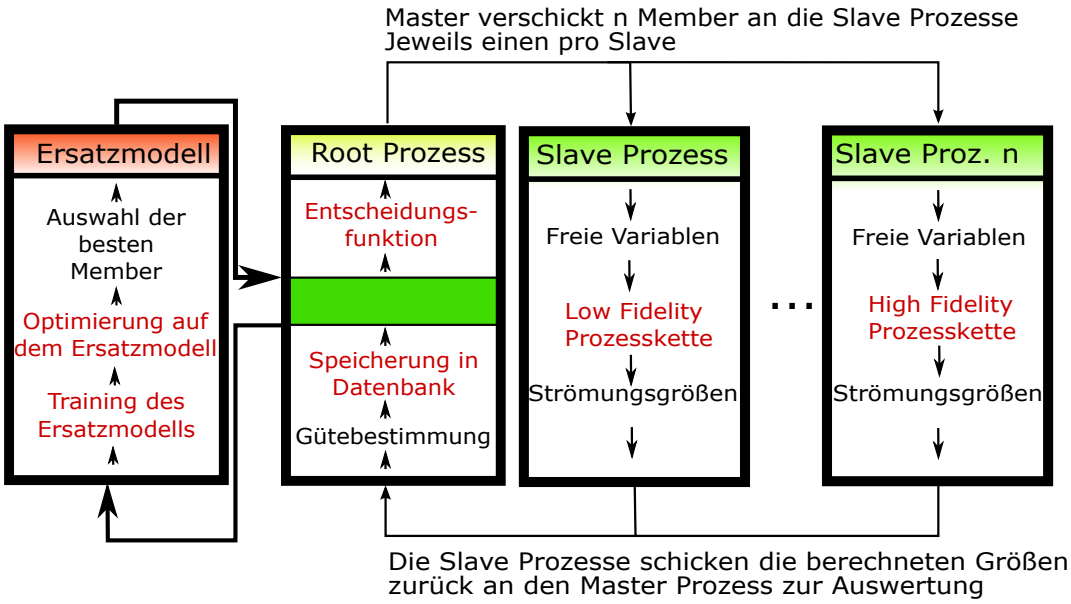


Abbildung 3.5: Mutlfidelity-Optimierungsprozess

3.2.1 Änderungen des Optimierungsprozesses

Im Folgenden sollen die nötigen Änderungen des bisherigen Optimierungsprozesses für die Nutzung eines Multifidelity-Verfahrens beschrieben werden. Der hier vorgestellte Prozess stellt eine Erweiterung des in Kapitel 2.3.1 vorgestellten Optimierungsprozesses dar. Grundlegend wurde bei der Entwicklung besonderer Wert darauf gelegt, eine Abwärtskompatibilität zu gewährleisten. Aus diesem Grund wurde versucht mit nur wenigen Änderungen an der bisherigen Strategie auszukommen. Einfachheitshalber sind in der Abbildung nur zwei Gütestufen dargestellt. Die Strategie bietet aber die Option mehrere Gütestufen zu verwenden.

In Abbildung 3.5 wird der umgesetzte Multifidelity-Optimierungsprozess dargestellt. Die Veränderungen zu dem in Kapitel 2.3.1 beschriebenen Prozess sind in rot markiert.

Änderungen am Training der Ersatzmodelle Wie bereits in Kapitel 2.3.1 beschrieben, müssen die in AutoOpti verwendeten Ersatzmodelle vor der Verwendung trainiert werden. Dieses dient zur Bestimmung von ersatzmodellspezifischen Parametern. Dabei kann es sich z.B. um Korrelationslängen oder Gewichtungen handeln. Diese Parameter werden mithilfe der vorhandenen Daten eingestellt. An dieser Vorgehensweise ändert sich bei der hier vorgestellten Multifidelity-Optimierung prinzipiell nichts. Die größte Änderung liegt in der Verwendung eines anderen Ersatzmodells, welches die

verschiedenen Gütestufen verwerten kann. In diesem Fall handelt es sich um das CO-Kriging Verfahren, welches im nächsten Kapitel noch ausführlich erläutert wird. Bei dem verwendeten CO-Kriging gibt es im Vergleich zum bisher verwendeten Ordinary-Kriging allerdings eine höhere Anzahl an ersatzmodellspezifischen Parametern, sodass das Training komplexer ist.

Änderungen an der Datenbank Da die Member der verschiedenen Gütestufen nur sehr schwer gegeneinander bewertet werden können, ist es am sinnvollsten diese in getrennten Datenbanken zu speichern oder in einer Datenbank getrennt zu behandeln. In AutoOpti werden die bewerteten Member verschiedener Gütestufen in jeweils eigenen Datenbanken gespeichert. Eine Bewertung der niedrigeren Stufen wird nicht vorgenommen, es wird also kein Paretorang oder ähnliche Kriterien berechnet. Die Berechnung eines solchen Kriteriums zwischen verschiedenen Gütestufen ist auch sehr schwierig, da die Transferfunktion zwischen den Stufen nicht bekannt ist. Daher werden die Daten der niedrigeren Stufen nur zur Verbesserung der Ersatzmodelle herangezogen.

Änderungen an der Prozesskette An der eigentlichen Prozesskette muss prinzipiell nichts verändert werden. Die Prozesskette wird in AutoOpti als Liste von nacheinander auszuführenden Prozessen hinterlegt und bei Bedarf wird diese mit einem Parametersatz gestartet und die resultierenden Ergebnisse an den Master-Prozess kommuniziert. Die dafür notwendigen Programme und Dateien werden typischerweise vom Benutzer in einem Template Ordner hinterlegt. Soll die Prozesskette gestartet werden, so kopiert der Slave-Prozess diesen Template Ordner, trägt den vorgegebenen Parametersatz ein und startet dann die jeweiligen Prozesse nacheinander.

Im Falle einer Multifidelity Optimierung, muss für jede Gütestufe eine eigene Prozesskette und ein eigener Template Ordner angelegt werden. In den meisten Fällen sind die Prozessketten und auch der Template Ordner denen der höheren Gütestufen sehr ähnlich, sodass nur minimale Änderungen vorgenommen werden müssen.

Gütebestimmung Für eine Multi-Fidelity Optimierung stellt sich die Frage, wie man die Gütebestimmung der Member durchführt. Beispielsweise die Bestimmung des Paretorangs oder des Volumenzugewinns. Grundsätzlich erscheint es sinnvoll, dafür nur die High-Fidelity Member zu verwenden, da der Vergleich zwischen verschiedenen Fidelities sehr schwer fällt. Außerdem interessiert den Anwender in der Regel nur das hochwertigste Ergebnis.

Aus diesen Gründen wird die Gütebestimmung nur an der höchsten Gütestufe durchgeführt. Das wiederum bedeutet, dass nur Member der höchsten Stufe einen direk-

ten Optimierungsfortschritt in Form eines Volumenzugewinns bringen können. Low-Fidelity Member können also nur indirekten Einfluss auf den Optimierungsfortschritt nehmen, indem sie das Ersatzmodell verbessern und den weiteren Optimierungsverlauf so günstig beeinflussen.

Entscheidungsfunktion Wurde nach der Optimierung auf dem Ersatzmodell ein vielversprechender Member erzeugt und soll nun mittels Prozesskette bewertet werden, so stellt sich die Frage, mit welcher Gütestufe der Member berechnet werden soll. Die Entscheidung sollte so ausfallen, dass der weitere Optimierungsfortschritt günstig beeinflusst wird. Dieses Themengebiet ist allerdings recht umfangreich und soll daher im folgenden Abschnitt 3.2.2 näher erläutert werden.

Änderungen an der Optimierung auf den Ersatzmodellen Wie auch bei einer Optimierung mit nur einer Gütestufe, wird nach erfolgreichem Training eine eigene Optimierung auf den Ersatzmodellen gestartet, wobei die Ersatzmodelle den Teil der echten Prozesskette durch Vorhersagen ersetzen. An diesem Punkt ändert sich prinzipiell nichts. Allerdings stellt sich dabei natürlich die Frage, wie die verschiedenen Gütestufen in die Optimierung auf dem Ersatzmodell integriert werden können. Im Normalfall soll nur die Zielfunktion der höchsten Güte optimiert werden und die Daten niedrigerer Güte dienen lediglich zur Beschleunigung. Durch das verwendete Ersatzmodell lässt sich eine solche Strategie relativ einfach umsetzen: Die Optimierung auf dem Ersatzmodell trifft nur Vorhersagen der höchsten Gütestufe, welche aber verbessert werden durch die bereits vorhandenen Daten der niedrigeren Stufen. Diese Strategie bietet außerdem den Vorteil, dass an der Optimierung auf dem Ersatzmodell praktisch keine Änderungen vorgenommen werden müssen.

3.2.2 Entscheidungsfunktion

Ein großes Problem bei der Durchführung einer Multi-Fidelity-Optimierung besteht darin zu entscheiden, mit welcher Prozesskette und damit Gütestufe ein neu erzeugter Member berechnet werden soll. Folgend sollen zwei beispielhafte Gütestufen betrachtet werden. Eine niedrige Gütestufe, welche mit dem Index low gekennzeichnet wird und eine höherwertige Gütestufe welche mit dem Index $high$ bezeichnet wird. Grundlegend wird davon ausgegangen, dass alle Gütestufen aus einem Simulationsprozess stammen und somit die Realität nicht fehlerfrei wiedergeben. Jeder dieser Prozesse hat also einen mittleren Fehler F.

Um eine sinnvolle Entscheidung treffen zu können, müssen zwei Dinge berücksichtigt werden. Auf der einen Seite sollte die Auswertung eines Members mit niedriger

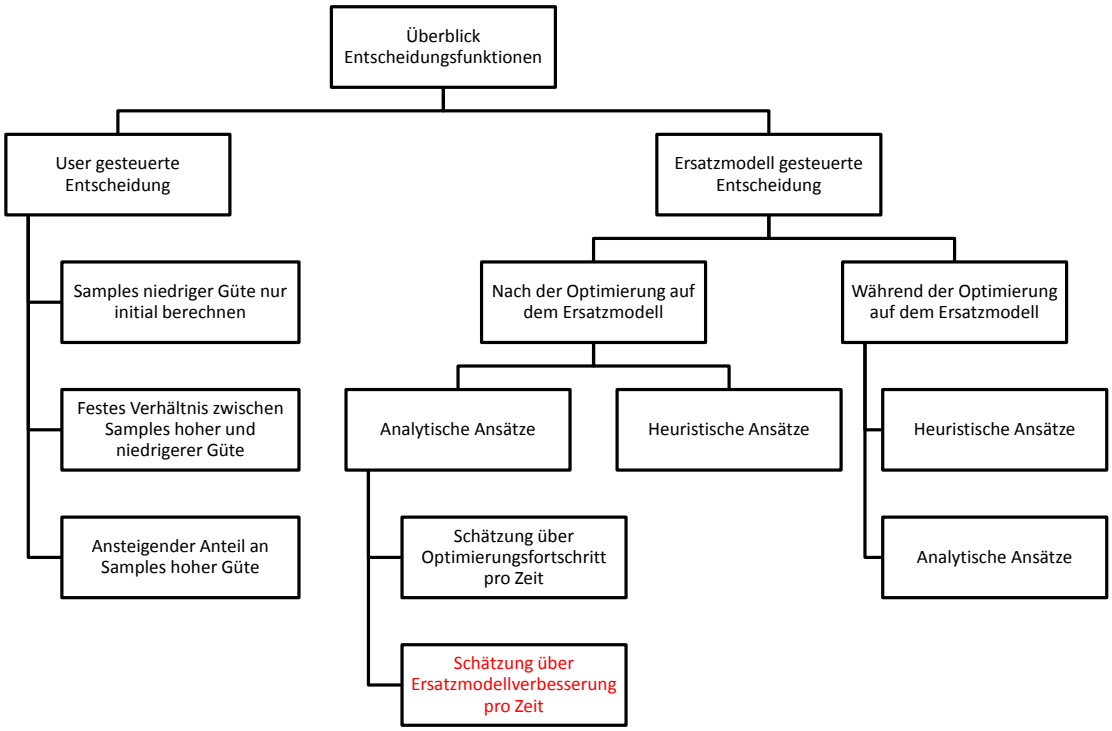


Abbildung 3.6: Möglichkeiten der Membererzeugung mit mehreren Fidelities

Gütestufe eine vergleichsweise kürzere Zeit $t_{low} < t_{high}$ als eine höhere Gütestufe benötigen. Auf der anderen Seite aber einen größeren mittleren Fehler $F_{low} \geq F_{high}$ haben als ein Member mit höherer Gütestufe.

Während einer laufenden Multifidelity-Optimierung muss also permanent eine Entscheidung getroffen werden, welche Prozesskette als nächstes durchlaufen werden soll. Benötigt wird also eine Entscheidungsfunktion f die diese Entscheidung während einer laufenden Optimierung vornimmt. Zudem ist natürlich eine Entscheidungsfunktion wünschenswert, die den Optimierungsverlauf möglichst günstig beeinflusst. Eine Entscheidungsfunktion zu finden, die einen optimalen Verlauf findet ist allerdings unrealistisch, da hierfür jeder mögliche Verlauf der gesamten Optimierung bekannt sein müsste. Aus diesem Grund, müssen einige Vereinfachungen und Annahmen über die Entscheidungsfunktion getroffen werden.

In Abbildung 3.6 sind einige Möglichkeiten der Membererzeugung kategorisiert und sollen folgend beschrieben werden.

3.2.2.1 Benutzergesteuerte Entscheidungsfunktionen

Die einfachste Variante wäre es, den Informationszugewinn nicht zu modellieren. In diesem Fall würde man die Auswahl der Fidelity bei der Membererzeugung in irgendeiner Form vorgeben, ohne die benötigte Zeit oder den Informationszugewinn zu berücksichtigen. Sinnvoll wäre hier z.B. am Anfang relativ viele Low-Fidelity Member zu erzeugen, um den groben Funktionsverlauf darzustellen und dann gegen Ende der Optimierung nur noch High-Fidelity Member zu erzeugen.

Eine weitere und auch häufig verwendete Methode ist es, vor dem Beginn der eigentlichen Optimierung einen festen Datensatz an Low-Fidelity Membern zu erzeugen und diese dem Ersatzmodelltraining hinzuzufügen. Während der Optimierung wird dann nur noch die High-Fidelity Prozesskette verwendet. Durch die anfangs hinzugefügten Low-Fidelity Member sollte sich das Ersatzmodell verbessern und dies wiederum die eigentliche Optimierung beschleunigen. Diese Methode ist sehr einfach umzusetzen, sollte aber nur am Anfang der Optimierung einen Zugewinn bringen. Zudem muss die Beschleunigung der Optimierung größer sein als die anfangs investierte Arbeit für die Erzeugung der Low-Fidelity Member. In einem sehr ungünstigen Fall kann diese Methodik also sogar zu einer Verlangsamung der Optimierung führen.

3.2.2.2 Ersatzmodellgesteuerte Entscheidungsfunktionen

In diesem Abschnitt sollen einige mögliche ersatzmodellbasierte Entscheidungsfunktionen vorgestellt werden. Der Fokus liegt allerdings bei den Entscheidungsfunktionen, welche nach der Optimierung auf dem Ersatzmodell durchgeführt werden. Dennoch soll kurz auf die Möglichkeit der Entscheidung während der Optimierung auf dem Ersatzmodell eingegangen werden.

Während der Optimierung auf dem Ersatzmodell

Die Optimierung auf dem Ersatzmodell schlägt in der Regel einen neuen Ort für einen vielversprechenden Member vor. Dieser Ort wird über die Ersatzmodellvorhersagen der Zielfunktionen und Nebenbedingungen bestimmt. Innerhalb von AutoOpti ist die Maximierung des „Expected Volume Gain“ das Mittel der Wahl. Es ist in diesem Zusammenhang denkbar diese Optimierung zusätzlich von den vorhandenen Gütestufen abhängig zu machen, indem versucht wird neben der Maximierung des „Expected Volume Gain“ den Informationsgehalt pro Zeit zu maximieren. Dieser Fall würde dann einer Mehrzieloptimierung entsprechen und ließe letztlich immer noch die Frage offen, welcher Member dieser so erzeugten Paretofront verwendet werden soll. Zudem wäre der numerische Aufwand mit jetziger Prozessorleistung nur schwer handhabbar.

Eine heuristische Lösung für dieses Problem kann in der Arbeit von Huang et. al. (siehe [Huang et al., 2006]) gefunden werden. Dieser multipliziert drei verschiedene Terme an den „Expected Volume Gain“ Term. Die Terme beinhalten einmal den zeitlichen Faktor zwischen einer niedrigeren Gütestufe und der höchsten Gütestufe. Weiterhin wird ein Term eingeführt, der die örtliche Vorhersageunsicherheit einbezieht und zuletzt ein Term der die Korrelation zwischen einer niedrigeren Gütestufe zur höchsten Gütestufe darstellt. Die Terme werden alle gleich gewichtet und sind auf einen Wertebereich von $[0, 1]$ normiert. Durch die Ortsabhängigkeit, verändert sich natürlich auch die Optimierung auf dem Ersatzmodell und damit auch der gesamte Optimierungsverlauf. Inwiefern dieser heuristischer Ansatz immer zu einer Beschleunigung führt, kann allerdings nur schwer abgeschätzt werden.

Nach der Optimierung auf dem Ersatzmodell

Für diese Art der Entscheidungsfunktion wird von der Optimierung auf dem Ersatzmodell ein fester Ort $\vec{x} \in \mathbb{R}^k \, wobei \, EVG\left(\vec{x}\right) \geq EVG\left(\vec{y}\right) \forall \vec{y} \in \mathbb{R}^k$ übergeben und dann entschieden welche Prozesskette ausgeführt wird.

Wie bereits beschrieben, ist ein Maß für den Zugewinn an Information durch ein Sample notwendig. Daher wird eine abstrakte Größe $I \in \{I_{low}, I_{high}\}$ definiert, welche diesen Informationszugewinn durch einen Member beschreibt. Der Informationszugewinn der niedrigeren Gütestufe soll kleiner oder gleich dem Informationszugewinn der höheren Stufe sein $I_{low} \leq I_{high}$. Außerdem gibt es für jede Fidelity jeweils nur einen Informationszugewinn, jedoch viele Zielfunktionale und Nebenbedingungen. Zusätzlich benötigt man eine Funktion, welche den Ortsvektor auf den Informationsgewinn abbildet:

$$\begin{aligned} I : \mathbb{R}^k &\rightarrow \mathbb{R}^2 \\ \vec{x} &\mapsto \begin{pmatrix} I_{low} \\ I_{high} \end{pmatrix} \end{aligned}$$

Die Entscheidungsfunktion selbst bildet den Informationsgewinn und die entsprechenden Zeiten auf eine Entscheidung ab:

$$
\begin{aligned}
&f_{dec} : \mathbb{R}^4 \rightarrow \{low, high\} \\
&\begin{pmatrix} I_{low} \\ t_{low} \\ I_{high} \\ t_{high} \end{pmatrix} \mapsto \begin{cases} low & , \frac{I_{low}}{t_{low}} \geq \frac{I_{high}}{t_{high}} \\ high & , sonst \end{cases}
\end{aligned} \tag{3.1}
$$

Dies lässt sich auch umformulieren zu einem Kriterium:

$$
\begin{pmatrix} I_{low} \\ t_{low} \\ I_{high} \\ t_{high} \end{pmatrix} \mapsto \begin{cases} low & , \frac{I_{low}}{t_{low}} \frac{t_{high}}{I_{high}} \geq 1 \\ high & , \frac{I_{low}}{t_{low}} \frac{t_{high}}{I_{high}} < 1 \end{cases} \tag{3.2}
$$

Die Entscheidungsfunktion bewertet nur zwei mögliche Gütestufen anhand des Informationsgewinns pro Zeit. Bei einer größeren Anzahl an Gütestufen kann die Funktion aber öfter angewendet werden und somit eine Rangliste erzeugt werden.

Ein sehr wichtiger Punkt ist die Modellierung des Informationszugewinns. Dafür eignet sich die Unsicherheitsvorhersage statistischer Ersatzmodelle sehr gut. Die meisten statistischen Ersatzmodelle sagen neben einem Erwartungswert für einen unbekannten Ort $\vec{x_0} \in \mathbb{R}^k$ auch eine Unsicherheit in Form einer Standardabweichung $\sigma \in \mathbb{R}$ voraus. Fügt man einem solchen Ersatzmodell an einem unbekannten Ort ein neues Sample hinzu, so sollte die vorhersagte Standardabweichung dort reduziert werden. Oftmals wird angenommen, dass die Unsicherheit auf Null reduziert wird, wenn an einer Stelle ein Sample hoher Güte bekannt ist.

Bei einem Ersatzmodell welches mehrere Gütestufen verarbeiten kann, verändert sich die Situation. Um einen Eindruck davon zu bekommen, wird in Abbildung 3.7 ein einfaches Beispiel gezeigt. Die blaue gestrichelte Kurve stellt die reale Funktion dar, in diesem Fall eine Sinus Schwingung. Die Samples hoher Güte werden durch rote Quadrate dargestellt und die Samples niedriger Güte durch blaue Dreiecke. Den Samples niedriger Güte wurde ein normalverteilter Fehler hinzugefügt. Da es sich nur um ein qualitatives Beispiel handelt, soll auf die verwendete Gleichung verzichtet werden. Bei dem verwendeten Ersatzmodell handelt es sich um ein CO-Kriging wie es im nächsten Kapitel vorgestellt wird. Die rote Kurve stellt die Vorhersage dieser Funktion anhand von den Samples hoher und niedriger Güte dar. Im unteren Diagramm wird die vorhergesagte Unsicherheit des Ersatzmodells in Form einer Standardabweichung darge-

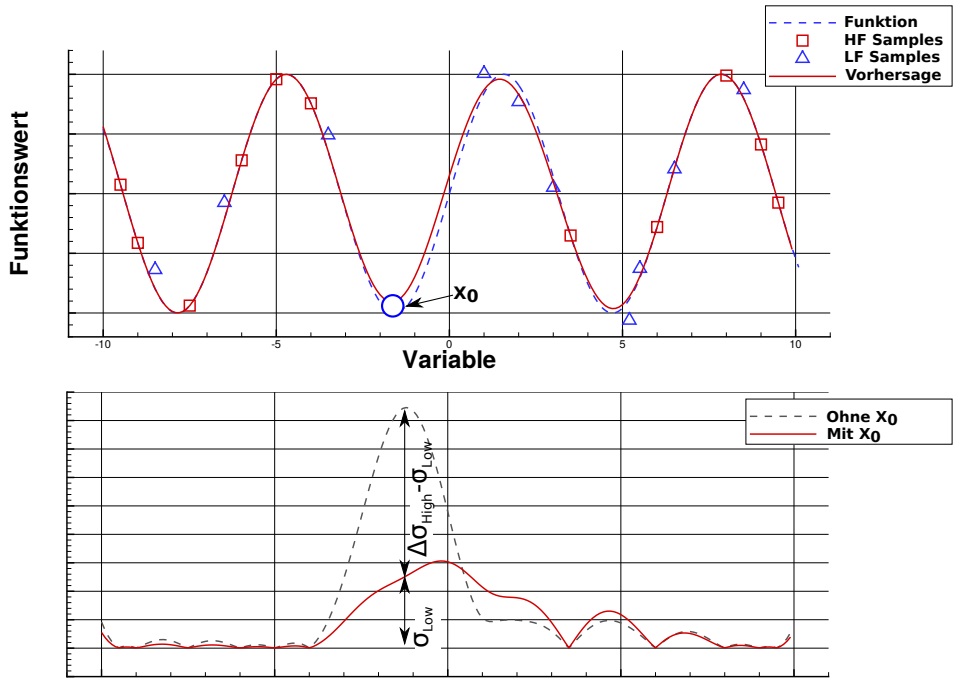


Abbildung 3.7: Beispielhafte Entwicklung der vorhergesagten Standardabweichung beim Hinzufügen von Samples verschiedener Gütestufen

stellt. Die schwarze gestrichelte Linie zeigt die Standardabweichung ohne ein Sample an der markierten Stelle $\vec{x_0}$ und die rote Kurve die Standardabweichung mit einem zusätzlichen Sample niedriger Güte.

Wichtig ist hierbei, dass sich die vorhergesagte Standardabweichung beim Hinzufügen eines Samples niedriger Güte reduziert. Es bleibt in der Regel also eine Restunsicherheit bestehen. Beim Hinzufügen eines Samples der höchsten Güte wird die Standardabweichung im Normalfall mit Null vorhergesagt. Die Reduzierung der Standardabweichung ist also je nach Gütestufe unterschiedlich groß, wobei die niedrigeren Gütestufen eine größere Restunsicherheit aufweisen als die höheren.

Um den Informationszugewinn zu modellieren ist die Reduktion der Standardabweichung ein geeignetes Maß. Um diese Reduktion zu berechnen müssen einige Definitionen und Annahmen getroffen werden.

Das innerhalb dieser Arbeit verwendete CO-Kriging kann prinzipiell für jede der verwendeten Gütestufen Vorhersagen treffen. Während des Optimierungsprozesses sind in diesem Fall nur die Vorhersagen der höchsten Gütestufe von Interesse. Aus diesem Grund wird im weiteren Verlauf grundlegend von Vorhersagen der höchsten Gütestufe ausgegangen.

Die Vorhersage der Standardabweichung der höchsten Gütestufe an einer Stelle $\vec{x_0} \in \mathbb{R}^k$ wird im Folgenden mit $\sigma_h \in \mathbb{R}$ benannt und die Vorhersage der Standardabweichung unter der Annahme, dass an der Stelle $\vec{x_0}$ ein Sample niedriger Güte bekannt ist, wird

folgend mit $\sigma_{hl} \in \mathbb{R}$ bezeichnet. Ist an der der Stelle $\vec{x_0}$ ein Sample der höchsten Güte bekannt, so wird weiterhin davon ausgegangen, dass die Unsicherheit $\sigma_{hh} \in \mathbb{R}$ auf Null reduziert wird $\sigma_{hh} = 0$.

Der Informationszugewinn für eine unbekannte Stelle x_0 kann damit als Reduktion der vorhergesagten Standardabweichung modelliert werden:

$$I_{low}\left(\vec{x_0}\right) = \sigma_h - \sigma_{hl} \tag{3.3}$$

$$I_{high}\left(\vec{x_0}\right) = \sigma_h - \sigma_{hh} = \sigma_h - 0 \tag{3.4}$$

Ein weiterer wichtiger Punkt ist die jeweils benötigte Zeit für die Erzeugung eines neuen Members $t_{low}, t_{high} \in \mathbb{R}$. Der grundlegende Ablauf einer Erzeugung wird in Abbildung 3.5 beschrieben und setzt sich im Wesentlichen aus drei Schritten zusammen. Der erste Schritt umfasst das Training der Ersatzmodelle und benötigt die Zeit $t_{train} \in \mathbb{R}$. Der zweite Schritt besteht aus der Optimierung auf dem Ersatzmodell und benötigt die Zeit $t_{opti} \in \mathbb{R}$. Der dritte und damit letzte Schritt beinhaltet die Berechnung der Prozesskette. Für diesen Schritt benötigen die Gütestufen auch jeweils unterschiedliche Zeiten welche mit $t_{prl} \in \mathbb{R}$ für die niedrigere Gütestufe und $t_{prh} \in \mathbb{R}$ für die höhere Gütestufe bezeichnet werden. Damit können die jeweiligen Erzeugungszeiten definiert werden:

$$t_{low} = t_{train} + t_{opti} + t_{prl}$$

$$t_{high} = t_{train} + t_{opti} + t_{prh}$$

Mit dem Informationszugewinn und den Zeiten, kann man die in Gleichung 3.1 definierten Verhältnisse von Informationszugewinn pro Zeit festlegen:

$$\frac{I_{low}}{t_{low}} = \frac{\sigma_h - \sigma_{hl}}{t_{train} + t_{opti} + t_{prl}}$$

$$\frac{I_{high}}{t_{high}} = \frac{\sigma_h}{t_{train} + t_{opti} + t_{prh}}$$

Diese Verhältnisse gelten für jeweils ein Ersatzmodell. Für mehrere Ersatzmodelle bietet sich eine gewichtete Summe aus den Einzelwerten an. Die Gewichte werden mit $w_i \in \mathbb{R}$ bezeichnet. Weiterhin wird davon ausgegangen, dass für jede der $z \in \mathbb{N}$ Zielfunktionen und $c \in \mathbb{N}$ Nebenbedingungen jeweils ein Ersatzmodell existiert.

$$\frac{I}{t} = \sum_{i}^{c+z} \frac{w_i}{\sum_i^{c+z} w_i} \frac{I_i}{t_i}$$

Vorerst sollen die Gewichte vernachlässigt werden, daher wird im Folgenden von $w_i = 1$ ausgegangen. Eine sinnvolle Belegung der Gewichte wird im späteren Verlauf des Kapitels besprochen. Daraus lässt sich nun das Kriterium aus Gleichung 3.2 vollständig definieren zu:

$$crit = \frac{I_{low}}{t_{low}} \frac{t_{high}}{I_{high}} = \frac{t_{train} + t_{opti} + t_{prh}}{t_{train} + t_{opti} + t_{prl}} \sum_{i}^{c+z} \frac{1}{\sum_i^{c+z} 1} \sum_{i}^{c+z} \left(\frac{\sigma_{i,h} - \sigma_{i,hl}}{\sigma_{i,h}} \right) \tag{3.5}$$

Mit dieser Formulierung kann das Kriterium aus Gleichung 3.1 Anwendung finden.

Gewichtete Ersatzmodellunsicherheit pro Prozesskettenzeit minimieren Bei der vorgestellten Entscheidungsfunktion, werden die verschiedenen Ersatzmodelle alle gleich gewichtet . Diese Annahme trifft in der Regel allerdings nicht zu. Um dies näher zu erläutern werden die Ersatzmodelle in zwei unterschiedliche Gruppen eingeteilt.

1. Ersatzmodelle für Zielfunktionen
2. Ersatzmodelle für Nebenbedingungen

Typischerweise gibt es in Turbomaschinenoptimierungen zahlreiche Nebenbedingungen welche am Anfang der Optimierung nicht erfüllt werden. Wie in Kapitel 2.2 beschrieben, ist die Erfüllung der Nebenbedingungen das wichtigste Kriterium. Dies führt dazu, dass das Optimierungsverfahren als erstes versucht die Nebenbedingungen zu erfüllen und die eigentlichen Zielfunktionen nur zweitrangig behandelt, zumindest bis die Nebenbedingungen erfüllt worden sind.

Abbildung 3.8 zeigt den typischen Verlauf einer solchen Optimierung. Als Beispiel wird die Benchmark-Optimierung aus Kapitel 6.5 vorgegriffen. Anhand dieses Beispiels lässt sich der Wechsel von Restriktionen zu Zielfunktionen sehr gut darstellen. Die grüne Kurve stellt den summierten Restriktionswert dar, ein Wert von Null bedeutet also die Erfüllung aller Restriktionen und ein Wert größer Null die Nichterfüllung mindestens einer Nebenbedingung. Die rote Kurve beschreibt den kummulierten Volumenzugewinn und kann somit als eine Art Optimierungsfortschritt bezogen auf die Zielfunktionen angesehen werden.

Die Abbildung zeigt sehr deutlich, wie im Verlauf der Optimierung ein Wechsel von den Nebenbedingungen hin zu den Zielfunktionen stattfindet. Sobald der Optimierungsal-

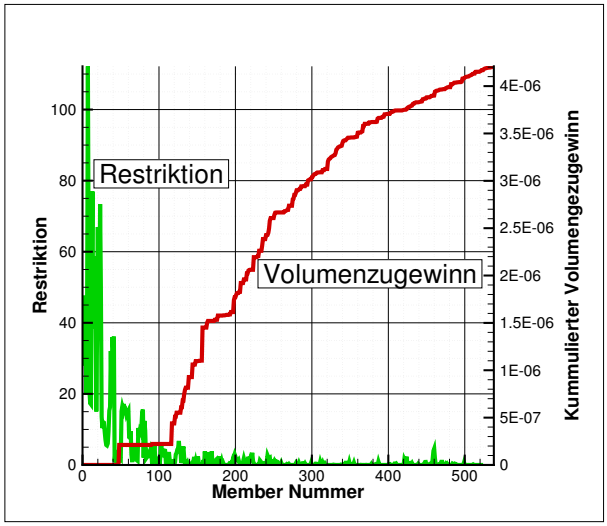


Abbildung 3.8: Typischer Verlauf einer Optimierung im Bezug auf die Restriktionen und Zielfunktionen

gorithmus die Nebenbedingungen einmal erfüllt hat, ist die Wahrscheinlichkeit diese weiterhin zu erfüllen erheblich größer. Dies sollte innerhalb der Entscheidungsfunktion natürlich berücksichtigt werden.

Weiterhin sind die verschiedenen Nebenbedingungen auch unterschiedlich schwer zu erfüllen und sollten daher ebenfalls unterschiedlich stark gewichtet werden.

Als geeignete Gewichtung für die Nebenbedingungen kann die Wahrscheinlichkeit $P_i \in \mathbb{R}$ für die Erfüllung der jeweiligen Nebenbedingungen verwendet werden. Je höher die Wahrscheinlichkeit die Nebenbedingung zu Erfüllen, desto niedriger das Gewicht der Nebenbedingung. Wenn z.B. die Wahrscheinlichkeit bei 100% liegt, so ist diese Nebenbedingung für die Entscheidung unerheblich, da diese unabhängig von der Entscheidung erfüllt wird. Die Zielfunktionen untereinander werden gleich gewichtet. Letztlich bleibt dann nur noch die Frage offen, wie Zielfunktionen und Nebenbedingungen zueinander gewichtet werden. Hier bietet sich die Gesamtwahrscheinlichkeit für die Erfüllung der Nebenbedingungen an. Geht man davon aus, dass diese unabhängig sind, so lässt sich die Gesamtwahrscheinlichkeit für die Erfüllung aller Nebenbedingungen als Produkt der Einzelwahrscheinlichkeiten darstellen $\prod_{i=1}^{c} P_i$. Mit dieser Gesamtwahrscheinlichkeit lassen sich dann auch die Zielfunktionen und Nebenbedingungen zueinander gewichten.

$$crit = T * \left[\left(1 - \prod_{i=1}^{c} P_i \right) \sum_{i=1}^{c} \frac{w_i}{\sum_{i=1}^{c} w_i} \left(\frac{\sigma_{i,h} - \sigma_{i,hl}}{\sigma_{i,h}} \right) + \left(\prod_{i=1}^{c} P_i \right) \sum_{j=1}^{z} \frac{1}{z} \left(\frac{\sigma_{j,h} - \sigma_{j,hl}}{\sigma_{j,h}} \right) \right] \tag{3.6}$$

$$w_i = (1 - P_i) \; \forall i \in \{1, ..., c\}$$
$$T = \frac{t_{train} + t_{opti} + t_{prh}}{t_{train} + t_{opti} + t_{prl}}$$

Globale Varianzreduktion Die bisher vorgestellten Kriterien beziehen sich alle auf einen lokalen Ort $\vec{x_0} \in \mathbb{R}^k$. Grundlegend ist es natürlich auch möglich den Einfluss der Entscheidung auf den gesamten Parameterraum zu schätzen oder zumindest auf die unmittelbare Umgebung. Um dies zu erreichen muss die Formulierung aus Gleichung 3.3 zu einem räumlichen Integral erweitert werden. Die Formulierung $\sigma_h(\vec{x}) \mid y_h(\vec{x}_0)$ soll die Vorhersage der Unsicherheit der höheren Gütestufe $\sigma_h(\vec{x}) \in \mathbb{R}$ an einer beliebigen Stelle $\vec{x} \in \mathbb{R}^k$ darstellen und zwar unter der Voraussetzung, dass an der Stelle $\vec{x_0} \in \mathbb{R}^k$ ein Sample $y_h(\vec{x}_0) \in \mathbb{R}$ hoher Güte oder ein Sample niedrigerer Güte $y_l(\vec{x}_0) \in \mathbb{R}$ bekannt ist.

$$I_{high}(\vec{x_0}) = \int \left(\sigma_h(\vec{x}) - [\sigma_h(\vec{x}) \mid y_h(\vec{x}_0)] \right) d\vec{x}$$

$$I_{low}(\vec{x_0}) = \int \left(\sigma_h(\vec{x}) - [\sigma_h(\vec{x}) \mid y_l(\vec{x}_0)] \right) d\vec{x}$$

Damit ändert sich Gleichung 3.6 zu:

$$crit = T * \left[\left(1 - \prod_{i=1}^{c} P_i \right) \sum_{i=1}^{c} \frac{w_i}{\sum_{i=1}^{c} w_i} \left(\frac{I_{low}}{I_{high}} \right) + \left(\prod_{i=1}^{c} P_i \right) \sum_{j=1}^{z} \frac{1}{z} \left(\frac{I_{low}}{I_{high}} \right) \right] \tag{3.7}$$
$$w_i = (1 - P_i) \; \forall i \in \{1, ..., c\}$$
$$T = \frac{t_{train} + t_{opti} + t_{prh}}{t_{train} + t_{opti} + t_{prl}}$$

Die mögliche lokale Änderungen der Gewichtungen $w_i \in \mathbb{R}$ soll an dieser Stelle allerdings unberücksichtigt bleiben. Da die lokale Varianzreduktion naturgemäß den größten Beitrag liefert, ist anzunehmen, dass diese Art des Kriteriums in den meisten Fällen zu keiner anderen Entscheidung führen wird. In Anhang A.11 wird ein Beispiel gezeigt, in dem eine solche Formulierung zu einer anderen Entscheidung führen würde, als die in Gleichung 3.6 beschriebene. Die Konstruktion eines solchen Beispiels hat sich allerdings als äußerst schwierig erwiesen.

Eine weitere Schwierigkeit bei der praktischen Umsetzung dieser Formulierung liegt in der numerischen Berechnung der Integrale, hierfür bietet sich eine Monte Carlo Integration an. Da die Dimension der Parameterräume und auch die Anzahl der Zielfunktionen und Nebenbedingungen bei typischen Turbomaschinenoptimierungen sehr hoch sind, ist die Berechnung eines solchen Integrals mit einem enormen Aufwand verbunden. Es ist unwahrscheinlich, dass dieser erhebliche Mehraufwand die Optimierung derart beschleunigen kann, dass insgesamt eine kürzere Optimierungszeit erreicht wird.

Einfluss der Entscheidungsfunktion auf den Optimierungsverlauf Abschließend muss die Frage gestellt werden, inwiefern die gewählte Entscheidungsfunktion überhaupt einen Einfluss auf die Optimierungszeit hat und wie stark dieser ist. Um eine Idee davon zu bekommen, sollen hier mehrere Testoptimierungen mit verschiedenen Entscheidungsverläufen miteinander verglichen werden. Es handelt sich hierbei um die 3D-CFD Optimierungen aus Kapitel 6.5. Diese Optimierungen eignen sich für einen Vergleich, da viele verschiedene Entscheidungsverläufe getestet wurden und es sich zudem bereits um eine realitätsnahe aeromechanische Optimierung handelt. Abbildung 3.9 zeigt den Vergleich dieser Optimierungen. Jeder Punkt stellt eine Optimierung dar und jede dieser Optimierungen hatte dieselbe Laufzeit zur Verfügung. Auf der Abszisse wird der am Optimierungsende erreichte Anteil zwischen Samples hoher Güte und Samples niedriger Güte dargestellt. Auf der Ordinate wird der erreichte Volumenzugewinn gezeigt.

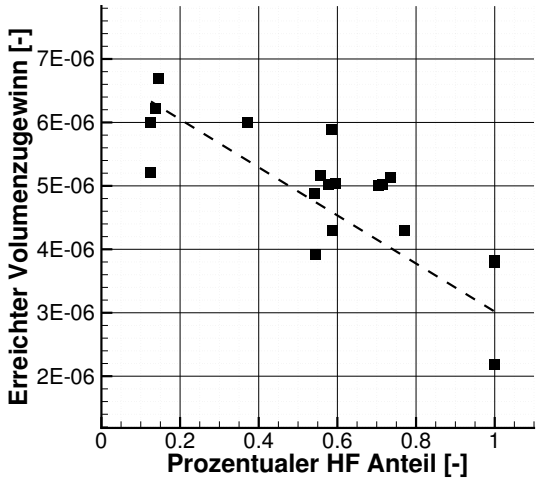


Abbildung 3.9: Vergleich verschiedener Optimierungen mit unterschiedlichen Entscheidungsverläufen

Es ist erkennbar, dass ein geringerer Anteil an Samples hoher Güte in diesem Fall eine deutliche Beschleunigung bewirkt. Die Prozesskette niedriger Güte muss also

bereits einen sehr hohen Informationsgehalt besitzen. Die Streuung der einzelnen Optimierungen kommt durch einige Zufälligkeiten innerhalb des Optimierungsprozesses zustande. Diese Zufälligkeiten entstehen bei der Optimierung auf dem Ersatzmodell und auch bei der Initialisierung des Trainings der Ersatzmodelle.

4 Die Kriging Verfahren

Den allgemeinsten Ansatz in dieser Arbeit stellt das CO-Kriging dar, das Simple-, Ordinary- und Gradient-Enhanced-Kriging können als Spezialfall es des CO-Krigings mit einer Fidelity angesehen werden. Berücksichtigt man diesen Umstand in der Herleitung der Verfahren, so kann man sich dies in der Planung der Softwarestruktur zu nutze machen und alle Verfahren effizient in einem Code unterbringen. Aus diesem Grund wird im ersten Abschnitt dieses Kapitels das CO-Kriging behandelt und in den darauffolgenden Abschnitten folgen die Herleitungen der anderen Kriging Verfahren auf Basis des CO-Krigings.

Da die in dieser Arbeit entwickelte Kriging-Software einen sehr hohen Anwendungsbezug hat und auch industriell eingesetzt werden soll, wurde bei der mathematischen Herleitung ein besonderes Augenmerk auf eine möglichst gute softwaretechnische Umsetzbarkeit gelegt. Dieser Punkt wird in der einschlägigen Literatur oftmals außer Acht gelassen, was die Effizienz der Software sowie die Generalisierbarkeit negativ beeinflusst.

Zusätzlich zu den Herleitungen soll ein Abschnitt auch die Verbindung des Simple-Kriging und einer Bedingten Normalverteilung herstellen, dieses Kapitel dient insbesondere zum besseren Verständnis aller Verfahren.

Der letzte Abschnitt behandelt einen Regularisierungsterms, welcher neben der numerischen Stabilisierung ebenfalls das Approximationsverhalten der Kriging Verfahren verändert.

CO-Kriging Formulierungen, welche auf einem Differenzmodell beruhen (z.B. [J. Toal & Keane, 2011]) haben den Nachteil, dass jeder Member mit der Low-Fidelity Prozesskette berechnet werden muss. Eine Entscheidungsfunktion müsste in diesem Fall nur noch entscheiden, ob ein Member nach der Low-Fidelity Berechnung noch die High-Fidelity Prozesskette durchlaufen soll oder nicht. Dieser Ansatz lohnt sich insbesondere dann, wenn man auf Basis des Low-Fidelity Members, die High-Fidelity Prozesskette fortsetzen kann. Beispielsweise könnte man eine Strömungslösung

nicht auskonvergieren lassen und diese Lösung als Low-Fidelity Member verwenden. Möchte man diesen Member dann noch mit der High-Fidelity Prozesskette berechnen, so kann man die bereits vorhandene Low-Fidelity Lösung einfach auskonvergieren lassen und verliert so keine zusätzliche Zeit. Um eine Entscheidung zu treffen, wann ein Member mit der High-Fidelity Prozesskette berechnet werden soll, kann man den erhaltenen Low-Fidelity Member dem Ersatzmodell hinzufügen und dann eine High-Fidelity Vorhersage mit dem neuen Ersatzmodell treffen. Anhand dieser Vorhersage kann dann eine sinnvolle Entscheidung getroffen werden, ob der entsprechende Member zusätzlich noch mit der High-Fidelity Prozesskette berechnet werden soll oder nicht.

4.1 Grundlagen Kriging

Unter Kriging versteht man statistische Verfahren zur Interpolation oder Approximation von Werten an unbeprobten Orten. Der Name des Verfahrens stammt von dem südafrikanischen Bergbauingenieur Daniel Krige (1951), dieser versuchte eine optimale Interpolationsmethode für den Bergbau zu entwickeln, die auf der räumlichen Abhängigkeit von Messpunkten [Krige, 1953] basiert. Das Verfahren wurde später nach ihm benannt. Der französische Mathematiker Georges Matheron (1963) entwickelte aus Kriges Arbeit, schließlich die Kriging Theorie[Matheron, 1963]. Das Kriging Verfahren hat heute in den Geowissenschaften sowie vielen anderen Forschungsbereichen Verwendung gefunden.

Die Vorteile von Kriging gegenüber anderer Methoden, wie z.B. Inverser Distanzwichtung [Shepard, 1968], insbesondere bei der Nutzung innerhalb eines Optimierungsverfahrens, wie es im Institut für Antriebstechnik existiert, sind:

- Die Initialisierung ist sehr einfach, d.h. das Verfahren liefert so gut wie immer gute Ergebnisse unabhängig von irgendwelchen Startparametern. Bei Neuronalen Netzwerken z.B. muss vor dem Training die Netztopologie, also die Anzahl der Gewichte und die Struktur des Netzes angegeben werden. Werden diese Parameter ungünstig gewählt, liefert das Neuronale Netz schlechte oder gar keine Ergebnisse.
- Das Extrapolationsverhalten ist bei einer Optimierung sehr vorteilhaft. Da man bei einer Optimierung sehr schnell in Bereiche kommt, wo das Ersatzmodell extrapolieren muss, sollten die vorhergesagten Werte möglichst sinnvoll sein. Beim Kriging liefert das Modell bei einer Extrapolation einen Erwartungswert. Ein Neuronales Netzwerk bspw. würde hier zufällige Werte vorhersagen, dies könnte bei einer Optimierung zu Schwierigkeiten führen.

- Kriging ist ein BLUE Schätzer [Plackett, 1950], Best Linear Unbiased Estimator. Also ein linearer Erwartungstreuer Schätzer minimaler Varianz. Im nächsten Abschnitt wird dies genauer erläutert.
- Viele nichtstatistische Verfahren, wie z.B. die Inverse Distanzwichtung [Shepard, 1968], beachten eine lokale Häufung von Stützstellen nicht. Bei einer lokalen Häufung von Stützstellen sollten diese weniger stark gewichtet werden. Ein statistisches Verfahren wie Kriging beinhaltet die gesamte räumliche Verteilung der Stützstellen [Krüger, 2012].
- Statistische Verfahren wie das Kriging können zusätzlich zur Vorhersage des eigentlichen Funktionswertes auch eine Unsicherheit vorhersagen. Dies kann innerhalb einer Optimierung ausgenutzt werden. Weitere Erläuterungen dazu sind in den Kapiteln 2 und 3 gefunden werden.
- Das Kriging Verfahren lässt sich sehr gut erweitern, beispielsweise die Nutzung von Gradienteninformationen (Gradient Enhanced Kriging) oder verschiedener Güteklassen (CO-Kriging). Hierdurch ist es möglich alle diese Verfahren in einen Code unterzubringen.

4.2 Co-Kriging

Der in diesem Kapitel vorgestellte Kriging Ansatz ist angelehnt an die Arbeiten von [Z.-H. Han, R. Zimmermann, 2010, Han et al., 2012] ,[Kennedy & O'Hagan, 2000] und [Krüger, 2012]. Die Unterschiede des hier vorgestellten Ansatzes zu den anderen Arbeiten werden in Kapitel 4.7 kurz erläutert.

Angenommen man hat ein Programm (bspw. einen Strömungslöser) welches in s verschiedenen Gütestufen (bspw. verschiedene Rechennetzauflösungen) unterteilt werden kann. Dieses Programm kann zu einem Parametersatz $\vec{x} \in \mathbb{R}^{\mathrm{k}}$ jeweils einen Output für jede Gütestufe $y_1(\vec{x}), ..., y_s(\vec{x})$ berechnen. Der Informationsgehalt I einer jeweiligen Gütestufe soll bei steigender Gütezahl sinken $I_i > I_{i+1}, i \in \{1, ..., s-1\}$, die erste Gütestufe ist somit die hochwertigste. Ebenfalls wird die Annahme gestellt, dass eine hochwertigere Gütestufe eine höhere Zeit $t_i, i \in \{1, ..., s\}$ zum Berechnen eines Outputs $y_i, i \in \{1, ..., s\}$ benötigt, es gilt also $t_i > t_{i+1}, i \in \{1, ..., s-1\}$.

Da jedes Kriging Modell ein statistisches Interpolationsverfahren darstellt, werden die Stützstellen der einzelnen Gütestufen $y_i(\vec{x}_j), i \in \{1, ..., s\}, j \in \{1, ..., n_i\}$ als Realisierung eines Zufallsprozesses $Z_i(\vec{x}), i \in \{1, ..., s\}$ angesehen. Desweiteren sollen die Zufallsprozesse der einzelnen Gütestufen stationär sein, also einen konstanten Erwartungswert $E[Z_i] = const.$ besitzen. Annahmen über die Verteilung der Zufallsprozesse

sollen an dieser Stelle noch nicht getroffen werden, diese folgen im späteren Verlauf des Kapitels.

Das Kriging Verfahren soll an einer unbekannten Stelle $\vec{x} \in \mathbb{R}^{\mathrm{k}}$ eine Schätzung über den Funktionswert $y_i^*(\vec{x})$ einer bestimmten Gütestufe $i \in \{1, ..., s\}$ anstellen. Die grundlegende Annahme dabei ist, dass der zu schätzende Wert $y_i^*(\vec{x})$ durch eine gewichtete Summe bestimmt werden kann. Diese Summe wird aus den bekannten Stützstellen $y_i(\vec{x}_j), i \in \{1, ..., s\}, j \in \{1, ..., n_i\}$ und den noch zu bestimmenden Gewichten $w_{i,j}, i \in \{1, ..., s\}, j \in \{1, ..., n_i\}$. Der Einfachheit halber werden die Stützstellen wie auch die Gewichte zukünftig durch die folgenden Vektoren beschrieben$\vec{y}_i = (y_i(\vec{x}_1), ..., y_i(\vec{x}_{n_i}))^T, i \in \{1, ..., s\}$ und $\vec{w}_i = (w_{i,1}, ..., w_{i,n_i})^T, i \in \{1, ..., s\}$.

Das Kriging Modell soll immer eine spezielle Gütestufe vorhersagen, daraus folgt die Gleichung für die Vorhersage der Gütestufe $k \in \{1, ..., s\}$:

$$y_k^*(\vec{x}) = \sum_{i=1}^{s} \vec{y}_i^T \vec{w}_i \tag{4.1}$$

Da die Stützstellen $y_i(\vec{x}_j), i \in \{1, ..., s\}, j \in \{1, ..., n_i\}$ einer Gütestufe i als Realisierung des jeweiligen stochastischen Prozesses $Z_i(\vec{x}), i \in \{1, ..., s\}$angesehen werden, stammen diese aus dem folgenden Vektor $\vec{Z}_i = (Z_i(\vec{x}_1), ..., Z_i(\vec{x}_{n_i}))^T, i \in \{1, ..., s\}$. Der Krige Schätzer $Z_k^*(\vec{x})$ für die Gütestufe k wird damit zu:

$$Z_k^*(\vec{x}) = \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i \tag{4.2}$$

Bestimmung der Gewichte

In diesem Abschnitt sollen einige grundlegende Annahmen getroffen werden, über diese und weitere Annahmen können dann die gesuchten Gewichte bestimmt werden. F sei der Schätzfehler, $Z_k(\vec{x})$ sei der reale stochastische Prozess der vorherzusagenden Gütestufe k.

$$F(\vec{x}) = Z_k(\vec{x}) - Z_k^*(\vec{x}) = Z_k(\vec{x}) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i$$

Kriging ist ein linearer erwartungstreuer Schätzer, was bedeutet, dass der Erwartungswert des Schätzfehlers 0 ist. Wobei $E[.]$ den Erwartungswert darstellt.

$$E\left[F\left(\vec{x}\right)\right] = 0 \tag{4.3}$$

$$\Leftrightarrow E\left[Z_k\left(\vec{x}\right)\right] - E\left[Z_k^*\left(\vec{x}\right)\right] = 0$$

$$E\left[Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right] = 0$$

$$E\left[Z_k\left(\vec{x}\right)\right] - E\left[\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right] = 0$$

$$E\left[Z_k\left(\vec{x}\right)\right] - E\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} Z_i\left(\vec{x}_j\right) w_{i,j}\right] = 0$$

$$E\left[Z_k\left(\vec{x}\right)\right] - \sum_{i=1}^{s}\sum_{j=1}^{n_i} E\left[Z_i\left(\vec{x}_j\right)\right] w_{i,j} = 0$$

Aufgrund der Stationarität soll gelten $E\left[Z_i\left(\vec{x}_j\right)\right] = E\left[Z_i\left(\vec{x}_k\right)\right] = \beta_i, \forall j,k \in \{1,...,n_i\}$:

$$\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} = \beta_k \tag{4.4}$$

Eine weitere Nebenbedingung des Kriging Verfahrens ist die minimale Varianz des Schätzfehlers:

$$\operatorname{var}\left[F\left(\vec{x}\right)\right] = \min_{w_1,...,w_n} \operatorname{var}\left[Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right] \tag{4.5}$$

Die Varianz des Schätzfehlers kann mit Hilfe des Verschiebungssatz aus der Statistik folgendermaßen umformuliert werden:

$$\mathrm{var}\left[F\left(\vec{x}\right)\right] = \mathrm{var}\left[Z\left(\vec{x}_0\right) - Z^*\left(\vec{x}_0\right)\right] = E\left[\left(\left(Z\left(\vec{x}\right) - Z^*\left(\vec{x}\right)\right) - E\left[Z\left(\vec{x}\right) - Z^*\left(\vec{x}\right)\right]\right)^2\right]$$

Im Anhang A.2 ist die vollständige Herleitung zu finden. Die folgende Formulierung stellt das Ergebnis dar, wobei $\vec{w} \in \mathbb{R}^{\mathrm{n_{all}}}$ den Gewichtsvektor, $\mathbf{Cov} \in \mathbb{R}^{\mathrm{n_{all}} \times \mathrm{n_{all}}}$ die Kovarianzmatrix und $\overrightarrow{c} \in \mathbb{R}^{\mathrm{n_{all}}}$ $\overrightarrow{c} = \left(cov\left(Z_i\left(\vec{x}_j\right), Z\left(\vec{x}\right)\right), ..., cov\left(Z_s\left(\vec{x}_{n_s}\right), Z\left(\vec{x}\right)\right)\right)^T, i \in \{1, ..., s\}, j \in \{1, ..., n_i\}$den Kovarianzvektor zwischen $Z_i\left(\vec{x}_j\right), i \in \{1, ..., s\}, j \in \{1, ..., n_i\}$ und $Z\left(\vec{x}\right)$ darstellt. Die Variable n_{all} beschreibt die Anzahl aller Stützstellen der verschiedenen Gütestufen:

$$\mathrm{var}\left[F\left(\vec{x}\right)\right] = \mathrm{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathrm{c}}^T\vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \tag{4.6}$$

Damit lässt sich folgende Minimierungsaufgabe mit Nebenbedingung stellen:

$$\begin{cases} \min_{w_1,...,w_n} \mathrm{var}\left[Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right] \\ \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k \qquad\qquad = 0 \end{cases} \tag{4.7}$$

Mit dieser Formulierung kann nun die Minimierungsaufgabe aus Gleichung **??** gemäß der Multiplikatorenmethode von Lagrange gelöst werden [Bronstejn & Semendjajew, 2008]. Als Lagrange Funktion $\Lambda\left(w, \lambda\right)$ mit dem Lagrange Multiplikator λ ergibt sich:

$$\Lambda\left(w, \lambda\right) := \mathrm{var}\left[F\left(\vec{x}\right)\right] + \lambda\left(\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k\right)$$

$$\nabla_{\lambda,w}\left(\mathrm{var}\left[F\left(\vec{x}\right)\right] + \lambda\left(\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k\right)\right) = 0$$

Daraus lässt sich das folgende Gleichungssystem ableiten:

$$\nabla_w\left(\mathrm{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathrm{c}}^T\vec{w}stoxx + \vec{w}^T * \mathbf{Cov} * \vec{w} + \lambda\left(\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k\right)\right) = 0$$

$$\wedge\ \nabla_\lambda \left(\text{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathbf{c}}^T\vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} + \lambda \left(\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k \right) \right) = 0 \qquad (4.8)$$

Es werden folgende Variablen eingeführt:

$$\text{var}\left[Z\left(\vec{x}\right)\right] = \sigma_r^2$$

Damit ergibt sich:

$$\nabla_w \left(\sigma_r^2 - 2\overrightarrow{\mathbf{c}}^T\vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \right) + \nabla_w \left(\lambda \left(\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k \right) \right) = 0$$

$$\wedge\ \nabla_\lambda \left(\sigma_r^2 - 2\overrightarrow{\mathbf{c}}^T\vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \right) + \nabla_\lambda \left(\lambda \left(\sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k \right) \right) = 0$$

Aufgelöst:

$$\left(-2\nabla_w \overrightarrow{\mathbf{c}}^T\vec{w} + \nabla_w\vec{w}^T * \mathbf{Cov} * \vec{w} \right) + \nabla_w\lambda \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \nabla_w\lambda\beta_k = 0$$

$$\wedge\ \left(-2\nabla_\lambda \overrightarrow{\mathbf{c}}^T\vec{w} + \nabla_\lambda\vec{w}^T * \mathbf{Cov} * \vec{w} \right) + \nabla_\lambda\lambda \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \nabla_\lambda\lambda\beta_k = 0$$

Zwischenschritt:

$$\left(-2\overrightarrow{\mathbf{c}}^T + 2\vec{w}^T * \mathbf{Cov} \right) + \nabla_w\lambda \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \nabla_w\lambda\beta_k = 0$$

$$\wedge\ \nabla_\lambda\lambda \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \nabla_\lambda\lambda\beta_k = 0$$

Zwischenschritt:

$$\left(-2\overrightarrow{\mathrm{c}}^T + 2\vec{w}^T * \mathbf{Cov}\right) + \lambda \nabla_w \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} = 0$$

$$\wedge \sum_{i=1}^{s} \beta_i \sum_{j=1}^{n_i} w_{i,j} - \beta_k = 0$$

Definiere $\vec{F}$:

$$\vec{F} = \begin{bmatrix} \beta_{1,1} \\ \vdots \\ \beta_{1,n_1} \\ \vdots \\ \beta_{s,n_s} \end{bmatrix}$$

Zwischenschritt:

$$-2\vec{c}^T + 2\vec{w}^T * \mathbf{Cov} + \lambda \vec{F} = 0$$

$$\wedge \vec{F}^T \vec{w} = \beta_k$$

Umgeformt:

$$\vec{w}^T * \mathbf{Cov} + \frac{\lambda \vec{F}}{2} = \vec{c}^T$$

$$\wedge \vec{F}^T \vec{w} = \beta_k$$

In Matrix Schreibweise:

$$\begin{pmatrix} \mathbf{Cov} & \vec{F} \\ \vec{F}^T & 0 \end{pmatrix} \begin{pmatrix} \vec{w}^T \\ \frac{\lambda}{2} \end{pmatrix} = \begin{pmatrix} \vec{c}^T \\ \beta_k \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} \vec{w}^T \\ \frac{\lambda}{2} \end{pmatrix} = \begin{pmatrix} \mathbf{Cov} & \vec{F} \\ \vec{F}^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} \vec{c}^T \\ \beta_k \end{pmatrix} \tag{4.9}$$

Die Auflösung der Gleichung nach den Gewichten bleibt prinzipiell dieselbe wie im Ordinary Kriging.

Die Inverse der Blockmatrix ergibt sich nach [Thornburg, 2006] zu:

$$\begin{pmatrix} \mathbf{Cov} & \vec{F} \\ \vec{F}^T & 0 \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{Cov}^{-1} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} & \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{R}^{-1}\vec{F}} \\ \frac{\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}F} & -\frac{1}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} \end{pmatrix}$$

Eingesetzt in Gleichung 4.9:

$$\begin{pmatrix} \vec{w}^T \\ \frac{\lambda}{2} \end{pmatrix} = \begin{pmatrix} \mathbf{Cov}^{-1} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} & \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} \\ \frac{\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}F} & -\frac{1}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} \end{pmatrix} \begin{pmatrix} \vec{c}^T \\ \beta_k \end{pmatrix}$$

Daraus ergibt sich folgende Formulierung der gesuchten Gewichte:

$$\vec{w}^T = \vec{c}^T\mathbf{Cov}^{-1} - \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

$$\vec{w} = \mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k \tag{4.10}$$

4.2.1 Allgemeine Kriging Vorhersage

In diesem Abschnitt sollen die Gleichungen für die Vorhersage des Erwartungswerts, der Varianz und der Kovarianz zwischen zwei Vorhersagen aufgestellt werden. Ein besonderes Augenmerk wurde hierbei darauf gelegt, dass die Gleichungen im Ordinary-, Gradient-Enhanced- und im CO-Kriging verwendbar sind ohne Änderungen daran vornehmen zu müssen.

4.2.1.1 Erwartungswert

Setzt man die Gewichte in Formel 4.2ein, erhält man folgende Gleichung. Wobei die Vektoren die Gewichte und Stützstellen aller Gütestufen enthalten.

$$Z^*\left(\vec{x}\right) = \vec{c}^T \mathbf{Cov}^{-1}\vec{y} - \vec{c}^T \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{y} + \frac{\vec{F}^T\mathbf{Cov}^{-1}\vec{y}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

$$\mu = \frac{\vec{F}^T\mathbf{Cov}^{-1}\vec{y}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}$$

$$Z^*\left(\vec{x}\right) = \vec{c}^T \mathbf{Cov}^{-1}\left(\vec{y} - \mu\vec{F}\right) + \mu\beta_k \tag{4.11}$$

In Anhang A.8 kann gezeigt werden, dass folgende Gleichung gilt:

$$\mu = 1$$

Daraus ergibt sich dann die letztendliche Formel für den Krige-Schätzer:

$$Z^*\left(\vec{x}\right) = \vec{c}^T \mathbf{Cov}^{-1}\left(\vec{y} - \vec{F}\right) + \beta_k \tag{4.12}$$

Diese Art der Formulierung bietet einen Vorteil bei der Vorhersage. Insbesondere dann, wenn man verschiedene Fidelities vorhersagen möchte. Da bei dieser Art der Formulierung der Vektor $\vec{F}$ und der Wert μ für alle Kriging Methoden gleich bleiben.

4.2.1.2 Varianz des Schätzfehlers direkte Herleitung

Varianz des 4.6

$$\vec{w} = \mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k \tag{4.13}$$

$$\mathrm{var}\left[F\left(\vec{x}\right)\right] = \mathrm{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathbf{c}}^T\vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \tag{4.14}$$

$$= \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T \mathbf{Cov}^{-1} \vec{c} + \frac{1}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}} \left(\vec{c}^T \mathbf{Cov}^{-1} \vec{F} - \beta_k\right)^2$$

4.2.1.3 Kovarianz zwischen zwei Vorhersagen

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right)$$

$$\Leftrightarrow cov\left(\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_{1i}, \sum_{j=1}^{s} \vec{Z}_j^T \vec{w}_{2j}\right)$$

$$cov\left(\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_{1i}, \sum_{j=1}^{s} \vec{Z}_j^T \vec{w}_{2j}\right) = E\left[\left(\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_{1i} - E\left[\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_{1i}\right]\right) * \left(\sum_{j=1}^{s} \vec{Z}_j^T \vec{w}_{2j} - E\left[\sum_{j=1}^{s} \vec{Z}_j^T \vec{w}_{2j}\right]\right)\right.$$

$$\Leftrightarrow E\left[\left(\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_{1i} - \sum_{i=1}^{s} E\left[\vec{Z}_i^T \vec{w}_{1i}\right]\right) * \left(\sum_{j=1}^{s} \vec{Z}_j^T \vec{w}_{2j} - \sum_{j=1}^{s} E\left[\vec{Z}_j^T \vec{w}_{2j}\right]\right)\right]$$

$$\Leftrightarrow E\left[\sum_{i=1}^{s} \left(\vec{Z}_i^T \vec{w}_{1i} - E\left[\vec{Z}_i^T \vec{w}_{1i}\right]\right) * \sum_{j=1}^{s} \left(\vec{Z}_j^T \vec{w}_{2j} - E\left[\vec{Z}_j^T \vec{w}_{2j}\right]\right)\right]$$

$$\Leftrightarrow E\left[\sum_{i=1}^{s} \sum_{j=1}^{s} \left(\vec{Z}_i^T \vec{w}_{1i} - E\left[\vec{Z}_i^T \vec{w}_{1i}\right]\right) * \left(\vec{Z}_j^T \vec{w}_{2j} - E\left[\vec{Z}_j^T \vec{w}_{2j}\right]\right)\right]$$

$$\Leftrightarrow \sum_{i=1}^{s} \sum_{j=1}^{s} E\left(\left(\vec{Z}_i^T \vec{w}_{1i} - E\left[\vec{Z}_i^T \vec{w}_{1i}\right]\right) * \left(\vec{Z}_j^T \vec{w}_{2j} - E\left[\vec{Z}_j^T \vec{w}_{2j}\right]\right)\right)$$

$$\Leftrightarrow \sum_{i=1}^{s} \sum_{j=1}^{s} cov\left(\vec{Z}_i^T \vec{w}_{1i}, \vec{Z}_j^T \vec{w}_{2j}\right)$$

$$\Leftrightarrow \sum_{i=1}^{s} \sum_{j=1}^{s} cov\left(\sum_{l=1}^{n_i} Z_i\left(\vec{x_l}\right) w_{1i,l}, \sum_{m=1}^{n_i} Z_j\left(\vec{x_m}\right) w_{2j,m}\right)$$

$$\Leftrightarrow \sum_{i=1}^{s} \sum_{j=1}^{s} \sum_{l=1}^{n_i} \sum_{m=1}^{n_i} cov\left(Z_i\left(\vec{x_l}\right) w_{1i,l}, Z_j\left(\vec{x_m}\right) w_{2j,m}\right)$$

$$\Leftrightarrow \sum_{i=1}^{s} \sum_{j=1}^{s} \sum_{l=1}^{n_i} \sum_{m=1}^{n_i} w_{1i,l} w_{2j,m} cov\left(Z_i\left(\vec{x_l}\right), Z_j\left(\vec{x_m}\right)\right)$$

Die folgende Formulierung stellt das Ergebnis dar, wobei $\vec{w} \in \mathbb{R}^{\mathrm{n_{all}}}$ den Gewichtsvektor,

$\mathbf{Cov} \in \mathbb{R}^{\mathrm{n_{all}} \times \mathrm{n_{all}}}$ die Kovarianzmatrix darstellt. Die Variable n_{all} beschreibt die Anzahl aller Stützstellen der verschiedenen Gütestufen, in dieser Formulierung sind also die Stützstellen und Gewichte aller Gütestufen auf einmal formuliert. In der softwaretechnischen Umsetzung wird dies ebenfalls so gehandhabt.

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{w_1}^T \mathbf{Cov} \vec{w_2} \tag{4.15}$$

Das Ergebnis sieht wie folgt aus, die genaue Herleitung kann in Anhang A.3 nachgelesen werden.

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{c_1}^T \mathbf{Cov}^{-1} \vec{c_2} - \frac{\vec{c_1}^T \mathbf{Cov}^{-1} \vec{F} \vec{F}^T \mathbf{Cov}^{-1} \vec{c_2}}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}} + \frac{\beta_k^2}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}}$$

4.2.1.4 Zusammenhang zwischen Vorhersage-Kovarianz und Fehler Varianz

In diesem Abschnitt soll in kurzer Form der Zusammenhang zwischen der vorhergesagte Fehler Varianz $\mathrm{var}\left[F\left(\vec{x}\right)\right]$ und der Vorhersage der Kovarianz zwischen zwei unbekannten Orten hergestellt werden. Die Vorhersage der Kovarianz sieht wie folgt aus:

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{w_1}^T \mathbf{Cov} \vec{w_2} = \vec{c_1}^T \mathbf{Cov}^{-1} \vec{c_2} - \frac{\vec{c_1}^T \mathbf{Cov}^{-1} \vec{F} \vec{F}^T \mathbf{Cov}^{-1} \vec{c_2}}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}} + \frac{\beta_k^2}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}}$$

Und die Vorhersage der Fehler Varianz sieht folgendermaßen aus:

$$\mathrm{var}\left[F\left(\vec{x}\right)\right] = \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T \mathbf{Cov}^{-1} \vec{c} + \frac{1}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}} \left(\vec{c}^T \mathbf{Cov}^{-1} \vec{F} - \beta_k\right)^2$$

Die Varianz einer Kriging Vorhersage kann über die Kovarianzformel einfach beschrieben werden:

$$\mathrm{var}\left[Z_k^*\left(\vec{x}\right)\right] = cov\left(Z_k^*\left(\vec{x}\right), Z_k^*\left(\vec{x}\right)\right) = \vec{w}^T \mathbf{Cov} \vec{w}$$

Die Fehlervarianz setzt sich wie folgt zusammen:

$$\mathrm{var}\left[F\left(\vec{x}\right)\right] = \mathrm{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathbf{c}}^{T}\vec{w} + \vec{w}^{T}\mathbf{Cov}\vec{w}$$

Es ist nun ersichtlich, dass die Varianz $\mathrm{var}\left[Z_k^*\left(\vec{x}\right)\right]$ in die Fehler Varianz Formel eingesetzt werden kann. In Anhang A.5 wird dies getan und es kann gezeigt werden, dass daraus die ursprüngliche Fehler Varianz Gleichung wieder hergestellt wird.

4.2.2 Kovarianzmodell

Der vorgestellte CO-Kriging Ansatz basiert in den Grundzügen auf den Arbeiten von Kennedy und Forrester [Kennedy & O'Hagan, 2000][Forrester et al., 2007]. Um diesen Ansatz zu vervollständigen ist es notwendig eine Bildungsvorschrift für die Kovarianzmatrix $\mathbf{Cov} \in \mathbb{R}^{\mathrm{n_{all}} \times \mathrm{n_{all}}}$ zu bestimmen. In diesem Punkt ergeben sich auch die größten Unterschiede zwischen den verschiedenen CO-Kriging Ansätzen.

Der erste Schritt besteht darin, die notwendigen Kovarianzfunktionen zu definieren und dann eine geeignete Modellierung für diese zu finden. Geht man von dem gaußschen Prozess Z_i mit den unterschiedlichen Gütestufen $i \in \{1, ..., s\}$ aus, wobei die höhere Gütestufe mit dem niedrigeren Index gekennzeichnet ist, so sind für die vollständige Beschreibung der Kovarianzmatrix an einer Stelle$\vec{x}$ folgende Kovarianzfunktionen zu modellieren:

$$\mathrm{cov}\left(\mathrm{Z_{i-1}}\left(\tilde{\mathrm{x}}\right), \mathrm{Z_{i-1}}\left(\tilde{\mathrm{x}}'\right)\right) \tag{4.16}$$

$$\mathrm{cov}\left(\mathrm{Z_{i}}\left(\tilde{\mathrm{x}}\right), \mathrm{Z_{i-1}}\left(\tilde{\mathrm{x}}'\right)\right) \tag{4.17}$$

$$\mathrm{cov}\left(\mathrm{Z_{i}}\left(\tilde{\mathrm{x}}\right), \mathrm{Z_{i}}\left(\tilde{\mathrm{x}}'\right)\right) \tag{4.18}$$

Im Folgenden sollen zwei verschiedene Gütestufen betrachtet werden, wobei 0 die höhere Güte beschreibt und 1 die niedrigere Gütestufe. Ein häufig gewählter Ansatz für ein Kovarianzmodell ist der Folgende:

$$Z_0\left(\vec{x}\right) = aZ_1\left(\vec{x}\right) + Z_{diff}\left(\vec{x}\right) \tag{4.19}$$

Der Prozess hoher Güte Z_0 wird also modelliert aus einem skalierten Prozess niedrigerer Güte Z_1, wobei $a \in \mathbb{R}$ den Skalierungsfaktor darstellt und einem Differenzprozess Z_{diff}. Setzt man dieses Modell nun in die Kovarianzfunktion aus Gleichung 4.16 ein:

$$\mathrm{cov}\left(\mathrm{Z_0}\left(\tilde{\mathrm{x}}\right), \mathrm{Z_0}\left(\tilde{\mathrm{x}}'\right)\right) = \mathrm{cov}\left(\mathrm{aZ_1}\left(\tilde{\mathrm{x}}\right) + \mathrm{Z_{diff}}\left(\tilde{\mathrm{x}}\right), \mathrm{aZ_1}\left(\tilde{\mathrm{x}}'\right) + \mathrm{Z_{diff}}\left(\tilde{\mathrm{x}}'\right)\right)$$

$$= E\left[\left(\left(aZ_1\left(\vec{x}\right) + Z_{diff}\left(\vec{x}\right)\right) - E\left[aZ_1\left(\vec{x}\right) + Z_{diff}\left(\vec{x}\right)\right]\right)\left(\left(aZ_1\left(\vec{x}'\right) + Z_{diff}\left(\vec{x}'\right)\right) - E\left[aZ_1\left(\vec{x}'\right) + Z_{diff}\right.\right.\right.$$

$$= E\left[\left(aZ_1\left(\vec{x}\right) - aE\left[Z_1\left(\vec{x}\right)\right] + Z_{diff}\left(\vec{x}\right) - E\left[Z_{diff}\left(\vec{x}\right)\right]\right)\left(aZ_1\left(\vec{x}'\right) - aE\left[Z_1\left(\vec{x}'\right)\right] + Z_{diff}\left(\vec{x}'\right) - E\left[Z\right.\right.\right.$$

$$= E\left[\left(\left(aZ_1\left(\vec{x}\right) - aE\left[Z_1\left(\vec{x}\right)\right]\right) + \left(Z_{diff}\left(\vec{x}\right) - E\left[Z_{diff}\left(\vec{x}\right)\right]\right)\right)\left(\left(aZ_1\left(\vec{x}'\right) - aE\left[Z_1\left(\vec{x}'\right)\right]\right) + \left(Z_{diff}\left(\vec{x}'\right)\right.\right.\right.$$

$$= E\left[a^2\left(Z_1\left(\vec{x}\right) - E\left[Z_1\left(\vec{x}\right)\right]\right)\left(Z_1\left(\vec{x}'\right) - E\left[Z_1\left(\vec{x}'\right)\right]\right)\right] + E\left[a\left(Z_1\left(\vec{x}\right) - E\left[Z_1\left(\vec{x}\right)\right]\right)\left(Z_{diff}\left(\vec{x}'\right) - E\left[Z\right.\right.\right.$$
$$+ E\left[a\left(Z_{diff}\left(\vec{x}\right) - E\left[Z_{diff}\left(\vec{x}\right)\right]\right)\left(Z_1\left(\vec{x}'\right) - E\left[Z_1\left(\vec{x}'\right)\right]\right)\right] + E\left(\left(Z_{diff}\left(\vec{x}\right) - E\left[Z_{diff}\left(\vec{x}\right)\right]\right)\left(Z_{diff}\right.\right.$$

$$= a^2 cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}'\right)\right) + a cov\left(Z_1\left(\vec{x}\right), Z_{diff}\left(\vec{x}'\right)\right) + a cov\left(Z_{diff}\left(\vec{x}\right), Z_1\left(\vec{x}'\right)\right) + cov\left(Z_{diff}\left(\vec{x}\right), Z_{\right.}$$

Nimmt man ferner an, dass der Differenzprozess Z_{diff} und der Prozess $Z_1\left(\vec{x}\right)$ unkorreliert sind $cov\left(Z_1, Z_{diff}\right) = 0$, so gilt:

$$cov\left(Z_1\left(\vec{x}\right), Z_{diff}\left(\vec{x}'\right)\right) = 0$$

$$cov\left(Z_{diff}\left(\vec{x}\right), Z_1\left(\vec{x}'\right)\right) = 0$$

daraus folgt:

$$\operatorname{cov}\left(\mathrm{Z}_0\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_0\left(\tilde{\mathrm{x}}^{'}\right)\right) = a^2 cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right) + cov\left(Z_{diff}\left(\vec{x}\right), Z_{diff}\left(\vec{x}^{'}\right)\right)$$

Modelliert man nun $\operatorname{cov}\left(\mathrm{Z_i}\left(\tilde{\mathrm{x}}\right), \mathrm{Z_{i-1}}\left(\tilde{\mathrm{x}}^{'}\right)\right)$ nach demselben Schema, ergibt sich:

$$\operatorname{cov}\left(\mathrm{Z}_0\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_1\left(\tilde{\mathrm{x}}^{'}\right)\right) = \operatorname{cov}\left(aZ_1\left(\tilde{x}\right) + Z_{diff}\left(\tilde{x}\right), Z_1\left(\tilde{x}^{'}\right)\right)$$

$$= E\left[\left(\left(aZ_1\left(\vec{x}\right) + Z_{diff}\left(\vec{x}\right)\right) - E\left[aZ_1\left(\vec{x}\right) + Z_{diff}\left(\vec{x}\right)\right]\right)\left(Z_1\left(\vec{x}^{'}\right) - E\left[Z_1\left(\vec{x}^{'}\right)\right]\right)\right]$$

$$= E\left[\left(\left(aZ_1\left(\vec{x}\right) - E\left[aZ_1\left(\vec{x}\right)\right]\right) + \left(Z_{diff}\left(\vec{x}\right) - E\left[Z_{diff}\left(\vec{x}\right)\right]\right)\right)\left(Z_1\left(\vec{x}^{'}\right) - E\left[Z_1\left(\vec{x}^{'}\right)\right]\right)\right]$$

$$= E\left[\left(\left(aZ_1\left(\vec{x}\right) - E\left[aZ_1\left(\vec{x}\right)\right]\right)\left(Z_1\left(\vec{x}^{'}\right) - E\left[Z_1\left(\vec{x}^{'}\right)\right]\right) + \left(Z_{diff}\left(\vec{x}\right) - E\left[Z_{diff}\left(\vec{x}\right)\right]\right)\left(Z_1\left(\vec{x}^{'}\right) - E\left[Z_1\right.\right.\right.$$

$$= a cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right) + cov\left(Z_{diff}\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right)$$

Es gilt wieder:

$$cov\left(Z_{diff}\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right) = 0$$

Daraus ergibt sich

$$\operatorname{cov}\left(\mathrm{Z}_0\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_1\left(\tilde{\mathrm{x}}^{'}\right)\right) = a cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right)$$

Analog gilt für die letzte Kovarianzfunktion (siehe Gleichung 4.16):

$$\operatorname{cov}\left(\mathrm{Z}_1\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_1\left(\tilde{\mathrm{x}}^{'}\right)\right) = \operatorname{cov}\left(\mathrm{Z}_1\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_1\left(\tilde{\mathrm{x}}^{'}\right)\right)$$

Durch diese Umformungen ergibt sich letztlich für die drei notwendigen Kovarianzfunk-

tionen:

$$\mathrm{cov}\left(\mathrm{Z}_0\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_0\left(\tilde{\mathrm{x}}^{'}\right)\right) = a^2 cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right) + cov\left(Z_{diff}\left(\vec{x}\right), Z_{diff}\left(\vec{x}^{'}\right)\right) \quad (4.20)$$

$$\mathrm{cov}\left(\mathrm{Z}_0\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_1\left(\tilde{\mathrm{x}}^{'}\right)\right) = a cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right) \quad (4.21)$$

$$\mathrm{cov}\left(\mathrm{Z}_1\left(\tilde{\mathrm{x}}\right), \mathrm{Z}_1\left(\tilde{\mathrm{x}}^{'}\right)\right) = cov\left(Z_1\left(\vec{x}\right), Z_1\left(\vec{x}^{'}\right)\right) \quad (4.22)$$

Durch diese Annahme sind nur noch zwei Kovarianzfunktionen zu modellieren, alle anderen Abhängigkeiten ergeben sich durch diese beiden Funktionen. Da in einem realen CO-Kriging Modell in der Regel ortsabhängige Stützstellen approximiert werden sollen und die reale Kovarianzfunktion nicht bekannt ist, wird die Kovarianzfunktion durch ortsabhängige Korrelationsfunktionen modelliert. Diese Korrelationsfunktionen werden im Folgenden mit $c\left(\vec{x}_j, \vec{x}_k\right)$ bezeichnet, wobei $y_i\left(\vec{x}_j\right), i \in \{1, ..., s\}; j, k \in \{1, ..., n_i\}$ die jeweilige Stützstelle an dem Ort $\vec{x}$ darstellt. Wobei der Zusammenhang zwischen den Korrelationsfunktionen und den Kovarianzen der folgende ist $c_d\left(\vec{x}_j, \vec{x}_k\right) = \frac{1}{\sigma_d^2} cov\left(Z_{diff}\left(\vec{x}_j\right), Z_{diff}\left(\vec{x}_k\right)\right)$ und analog $c_l\left(\vec{x}_j, \vec{x}_k\right) = \frac{1}{\sigma_l^2} cov\left(Z_1\left(\vec{x}_j\right), Z_1\left(\vec{x}_k\right)\right)$. Damit ist es möglich eine Kovarianzmatrix zu modellieren, das folgende Beispiel stellt eine Kovarianzmatrix für zwei verschiedene Samples aus jeweils zwei Gütestufen dar:

$$\mathbf{Cov} = \left[\begin{array}{c|cc} & y_0\left(\vec{x}_1\right) & y_1\left(\vec{x}_2\right) \\ \hline y_0\left(\vec{x}_1\right) & a^2\sigma_l^2 c_l\left(\vec{x}_1, \vec{x}_1\right) + \sigma_d^2 c_d\left(\vec{x}_1, \vec{x}_1\right) & a\sigma_l^2 c_l\left(\vec{x}_1, \vec{x}_2\right) \\ y_1\left(\vec{x}_2\right) & a\sigma_l^2 c_l\left(\vec{x}_2, \vec{x}_1\right) & \sigma_l^2 c_l\left(\vec{x}_2, \vec{x}_2\right) \end{array}\right] \quad (4.23)$$

4.3 Andere Kriging Verfahren

Im folgenden Abschnitt, soll dem Leser ein Überblick über verschiedene Kriging Verfahren gegeben werden. Diese Verfahren wurden alle in der hier entwickelten Software umgesetzt. Die effiziente Verwendung all dieser Verfahren innerhalb einer geschlossenen Software, stellt eine besondere Herausforderung dar und soll im Kapitel 5.1.1 kurz angeschnitten werden.

4.3.1 Ordinary Kriging

Das Ordinary-Kriging kann als Untermenge des CO-Kriging angesehen werden. Der größte Unterschied besteht darin, dass nur von einer möglichen Gütestufe $s = 1$ ausgegangen wird. Ansonsten behalten alle Gleichungen aus Kapitel 4.2 ihre Gültigkeit und können vollständig weiterverwendet werden. Das in Kapitel 4.2.2 beschrieben Kovari-

anzmodell vereinfacht sich jedoch sehr stark. Es wird nur noch eine Kovarianzfunktion $cov\left(Z\left(\vec{x}\right), Z\left(\vec{x}^{'}\right)\right)$ für das gesamte Modell benötigt. Ebenfalls gibt es so nur noch eine stationäre Varianz σ^2 und somit auch nur noch eine ortsabhängige Korrelationsfunktion $c\left(\vec{x}_j, \vec{x}_k\right) = \frac{1}{\sigma^2} cov\left(Z\left(\vec{x}_j\right), Z\left(\vec{x}_k\right)\right)$. Die benötigte Kovarianzmatrix vereinfacht sich somit zu:

$$\mathbf{Cov} = \left[\begin{array}{c|cc} & y\left(\vec{x}_1\right) & y\left(\vec{x}_2\right) \\ \hline y\left(\vec{x}_1\right) & \sigma^2 c\left(\vec{x}_1, \vec{x}_1\right) & \sigma^2 c\left(\vec{x}_1, \vec{x}_2\right) \\ y\left(\vec{x}_2\right) & \sigma^2 c\left(\vec{x}_2, \vec{x}_1\right) & \sigma^2 c\left(\vec{x}_2, \vec{x}_2\right) \end{array}\right] \tag{4.24}$$

4.3.2 Gradient Enhanced Kriging

Das Gradient Enhanced Kriging ist eine Erweiterung des Ordinary Kriging. Beim GEK gehen partielle Ableitungen in der Form $\frac{\partial y(x)}{\partial x}$ an einigen beprobten Orten mit in die Bildung des Modells ein. Der Trainingsvektor $\vec{y}_s \in \mathbb{R}^{\mathrm{n+m}}$ kann in diesem Fall also die folgende Form annehmen:
$\vec{y}_s = \left(y\left(\vec{x}_1\right), ..., y\left(\vec{x}_n\right), \frac{\partial y(\vec{x}_{n+1})}{\partial x^j}, ..., \frac{\partial y(\vec{x}_{n+m})}{\partial x^j}\right), j \in \{1, ..., k\}$, wobei der obere Index von x die Variable darstellt, nach der abgeleitet wird und m ist die Anzahl der gegebenen partiellen Ableitungen. Beim Aufstellen der Kovarianzmatrix $\mathbf{Cov}$ müssen also auch Kovarianzen/Korrelationen zwischen den Gradienten und den Funktionswerten gebildet werden können. Die Kovarianzfunktion zwischen zwei Zufallsvariablen ist wie folgt definiert:

$$\mathrm{cov}(Z\left(\vec{x}_1\right), Z\left(\vec{x}_2\right)) = E\left[\left(Z\left(\vec{x}_1\right) - E\left[Z\left(\vec{x}_1\right)\right]\right)\left(Z\left(\vec{x}_2\right) - E\left[Z\left(\vec{x}_2\right)\right]\right)\right] \tag{4.25}$$

Um im Kriging Modell partielle Ableitungen verarbeiten zu können ist es notwendig, dass die Kovarianzfunktionen in der folgenden Form gebildet werden:

$$\mathrm{cov}(\frac{\partial Z\left(\vec{x}_1\right)}{\partial x_1^p}, \frac{\partial Z\left(\vec{x}_2\right)}{\partial x_2^l}), p, l \in \{1, ..., k\} \tag{4.26}$$

$$\mathrm{cov}(\frac{\partial Z\left(\vec{x}_1\right)}{\partial x_1^p}, Z\left(\vec{x}_2\right)), p \in \{1, ..., k\} \tag{4.27}$$

$$\mathrm{cov}(Z(\vec{x}_1), \frac{\partial Z(\vec{x}_2)}{\partial x_2^l}), l \in \{1, ..., k\} \tag{4.28}$$

Per Definition ergibt sich für Gleichung 4.26:

$$\mathrm{cov}(\frac{\partial Z(\vec{x}_1)}{\partial x_1^p}, Z(\vec{x}_2)) = E\left[\left(\frac{\partial Z(\vec{x}_1)}{\partial x_1^p} - E\left[\frac{\partial Z(\vec{x}_1)}{\partial x_1^p}\right]\right)(Z(\vec{x}_2) - E[Z(\vec{x}_2)])\right] \tag{4.29}$$

Im Folgenden soll gezeigt werden, dass sich die Kovarianzfunktion $\mathrm{cov}(\frac{\partial Z(\vec{x}_1)}{\partial x_1^p}, Z(\vec{x}_2))$ auch durch äußeres Differenzieren dieser abbilden lässt.

$$\frac{\partial}{\partial x_1^p}\mathrm{cov}(Z(\vec{x}_1), Z(\vec{x}_2)) = \frac{\partial}{\partial x_1^p}(E[(Z(\vec{x}_1) - E[Z(\vec{x}_1)])(Z(\vec{x}_2) - E[Z(\vec{x}_2)])])$$

$$\begin{aligned} =& E\left[(Z(\vec{x}_2) - E[Z(\vec{x}_2)])\frac{\partial}{\partial x_1^p}(Z(\vec{x}_1) - E[Z(\vec{x}_1)])\right] \\ &+ E\left[(Z(\vec{x}_1) - E[Z(\vec{x}_1)])\frac{\partial}{\partial x_1^p}(Z(\vec{x}_2) - E[Z(\vec{x}_2)])\right] \end{aligned}$$

$$\begin{aligned} =& E\left[(Z(\vec{x}_2) - E[Z(\vec{x}_2)])\frac{\partial}{\partial x_1^p}(Z(\vec{x}_1) - E[Z(\vec{x}_1)])\right] \\ &+ E\left[(Z(\vec{x}_1) - E[Z(\vec{x}_1)])\frac{\partial}{\partial x_1^p}(Z(\vec{x}_2) - E[Z(\vec{x}_2)])\right] \end{aligned}$$

Es gilt $\frac{\partial}{\partial x_1^p}(Z(x_2) - E[Z(x_2)]) = 0$, da der Ausdruck unabhängig von x_1 ist

$$= E\left[(Z(\vec{x}_2) - E[Z(\vec{x}_2)])\left(\frac{\partial}{\partial x_1^p}Z(\vec{x}_1) - E\left[\frac{\partial}{\partial x_1^p}Z(\vec{x}_1)\right]\right)\right]$$

Dieser Ausdruck entspricht Gleichung 4.29, also gilt:

$$\mathrm{cov}(\frac{\partial Z(\vec{x}_1)}{\partial x_1^p}, Z(\vec{x}_2)) = \frac{\partial}{\partial x_1^p}\mathrm{cov}(Z(\vec{x}_1), Z(\vec{x}_2)) \tag{4.30}$$

für die Gleichungen 4.26 und 4.28 gilt entsprechendes:

$$\mathrm{cov}(\frac{\partial Z\left(\vec{x}_1\right)}{\partial x_1^p}, \frac{\partial Z\left(\vec{x}_2\right)}{\partial x_2^l}) = \frac{\partial}{\partial x_1^p \partial x_2^l}\mathrm{cov}(Z\left(\vec{x}_1\right), Z\left(\vec{x}_2\right))$$

$$\mathrm{cov}(Z\left(\vec{x}_1\right), \frac{\partial Z\left(\vec{x}_2\right)}{\partial x_2^l}) = \frac{\partial}{\partial x_2^l}\mathrm{cov}(Z\left(\vec{x}_1\right), Z\left(\vec{x}_2\right))$$

Da die Korrelationsfunktion vorgegeben ist, kann diese entsprechend abgeleitet werden (sofern diese differenzierbar ist) und man erhält den nötigen Zusammenhang zwischen den Funktionswerten und Gradienten. Beispiele für Korrelationsfunktionen und deren Ableitungen werden in Kapitel 5.1.3 behandelt.

Auf die Herleitung des Krige Schätzers und der Vorhersage der Varianz soll hier im Weiteren nicht eingegangen werden, da diese im Wesentlichen der Herleitung des Ordinary Kriging entsprechen. Der Leser sei aber auf [Krüger, 2012, Han et al., 2009] verwiesen.

Die Verwendung dieses Verfahrens innerhalb einer Turbmaschinenoptimierung stellt eine aktuelle Thematik der Forschung dar. Grundlegend kann für diese Verfahren gesagt werden, dass eine effiziente Erzeugung der partiellen Ableitungen notwendig ist, um diese gewinnbringend einsetzen zu können. Daher beschäftigt sich ein großer Teil der aktuellen Forschung mit diesem Thema. Diese Thematik spielt prinzipiell nur eine Rolle für die Softwaretechnische Gestaltung (siehe Kapitel 5.1.1) und soll daher nicht weiter ausgeführt werden. Für weitere Informationen sei der Leser auf [Backhaus et al., 2012, Backhaus et al., 2017, Peter & Dwight, 2010] verwiesen.

4.4 Kovarianzmodelle

Um die Kovarianzmatrix aufzustellen, ist es nötig, einen Korrelationswert zwischen allen bekannten Stützstellen zu berechnen und auch zwischen den Stützstellen und deren partiellen Ableitungen. Um diesen Wert zu berechnen, wird eine Korrelationsfunktion verwendet. Wie bereits in Kapitel **??** beschrieben, kann die Korrelationsfunktion nur angenähert werden und ist abhängig vom Abstand zweier Stützstellen zueinander.

In diesem Abschnitt soll eine sehr häufig verwendete Korrelationsfunktion beschrieben und deren softwaretechnische Umsetzung gezeigt werden. Da diese Korrelationsfunktion einer gaußschen Normalverteilung sehr ähnlich ist, wird diese im Folgenden als gaußsche Korrelationsfunktion bezeichnet. Die Formel sieht wie folgt aus:

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(e^{\theta_l}|x_{1l}-x_{2l}|^2\right)} \tag{4.31}$$

Die Vektoren $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^{\mathrm{k}}$ stellen hier die Ortsvektoren der Stützstellen dar. Die Funktion liefert immer Werte zwischen Null und Eins, wobei eine Eins bei zwei identischen Stützpunken und eine Null bei unendlich weit entfernten Stützstellen herauskommen würde. Der Vektor $\vec{\theta} \in \mathbb{R}^{\mathrm{k}}$ (auch Hyperparameter genannt) legt fest, wie stark die Korrelation mit steigendem Abstand der Stützstellen zueinander abfällt, wobei ein hoher Wert einen stärkeren Abfall bewirkt. In Abbildung 4.1 ist eine beispielhafte Korrelationsfunktion mit zwei verschiedenen θ Werten zu sehen, wobei die Stützstellen in der Abbildung jeweils nur eine freie Variable haben. Auf der X-Achse des Diagramms ist die Differenz der beiden Stützstellen zueinander aufgetragen und auf der Y-Achse der entsprechende Korrelationswert. Die rote durchgezogene Kurve zeigt die Korrelationsfunktion mit einem $\theta = 0.01$ und die grüne gestrichelte Kurve hat ein $\theta = 0.1$. Nimmt man eine Differenz der beiden Stützstellen von z.B. $\Delta x = -5$ an, dann ergeben sich für einen Hyperparameter von $\theta = 0.01$ ein Korrelationswert von 0.53 und für $\theta = 0.1$ ein Korrelationswert von 0.08. In dem einen Fall wird also eine stärkere Abhängigkeit der beiden Stützstellen angenommen und im anderen Fall eine sehr schwache Abhängigkeit. Für das gesamte Kriging Modell gibt es genau einen $\vec{\theta}$ Vektor, die Bestimmung dieser Werte ist Aufgabe des Trainings.

In Abbildung 4.2 wird eine Korrelationsfunktion zwischen zwei Stützstellen $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^{\mathrm{k}}$ mit zwei freien Variablen $k = 2$ gezeigt. Für diesen Fall hat der Vektor $\vec{\theta} \in \mathbb{R}^{\mathrm{k}}$ ebenfalls zwei Komponenten. Auf der X-Achse ist die Differenz zwischen den beiden ersten Komponenten der beiden Stützstellen aufgetragen, auf der Y-Achse die Differenz zwischen den zweiten Komponenten der Stützstellenvektoren und die Z-Achse zeigt den entsprechenden Korrelationswert.

Einige mögliche Korrelationsfunktionsmodelle und auch die Bestimmung der Varianzen σ_l^2, σ_d^2 werden im Kapitel 4.5 und 5.1.3 weiter erläutert.

Die effiziente Umsetzung dieser Kovarianzmodelle wird in Kapitel 5.1.3 beschrieben.

In diesem Abschnitt soll darauf eingegangen werden, welche Probleme negative Hyperparameter hervorrufen und wie man diese vermeiden kann.

Zudem wurde in vorherigen Kapiteln bereits ein Diagonalaufschlag erwähnt, dieser wird auf die Diagonalelemente der Korrelationsmatrix addiert und soll die Kondition der Matrix verbessern. Die genaue Vorgehensweise soll in diesem Abschnitt erläutert werden.

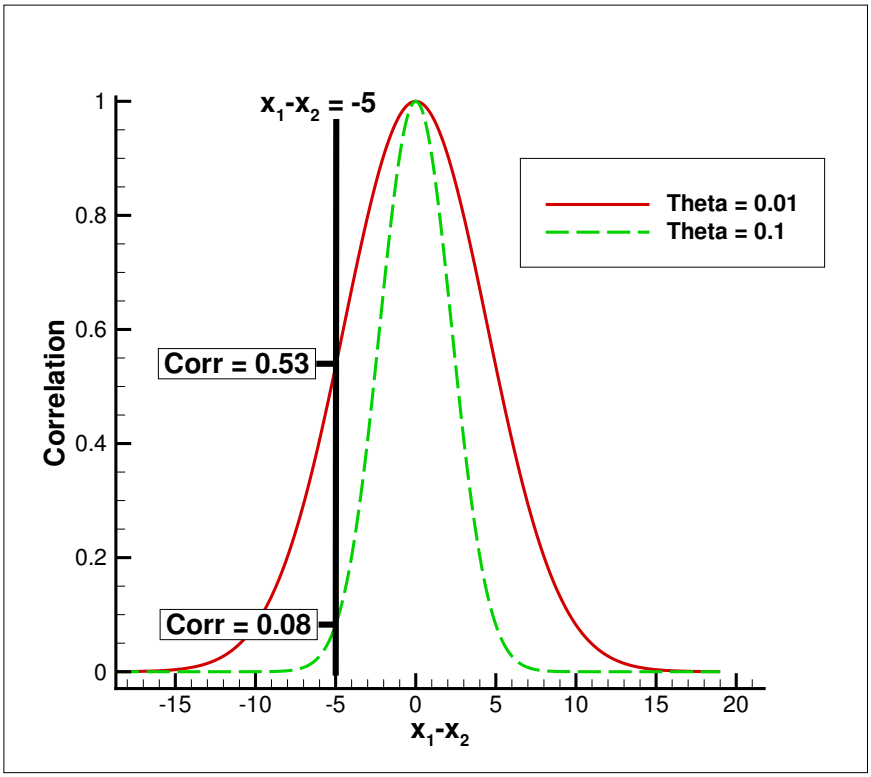


Abbildung 4.1: Beispiel einer Gauss Korrelationsfunktion mit einer freien Variable und zwei unterschiedlichen θ Einstellungen und deren Einfluss auf den Korrelationswert

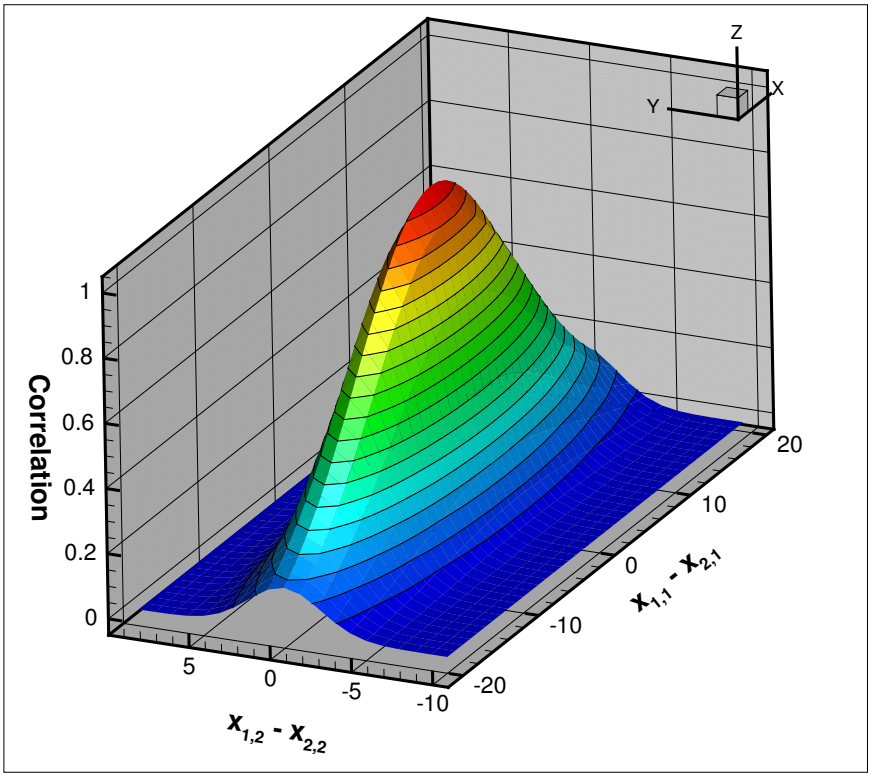


Abbildung 4.2: Beispiel einer Gauss Korrelationsfunktion mit zwei freien Variablen

Negative Hyperparameter

Während der Minimierung der Likelihood Funktion kann es je nach gewähltem Minimierungsverfahren oftmals dazu kommen, dass die Hyperparameter negativ werden können. Bei der Gauss Korrelationsfunktion würde dies dazu führen, dass bei großen Abständen zwischen den Membern auch große Korrelationen vorhergesagt werden würden. Die Korrelationen könnten so auch einen Wert über Eins erhalten, was keinen Sinn macht. Daher sind negative Hyperparameter bei der Gauss Funktion zwingend zu vermeiden.

Eine einfache Möglichkeit dies zu tun, wäre während der Minimierung die Hyperparameter nach unten zu begrenzen. Dies kann bei einigen Minimierungsverfahren allerdings zu schwerwiegenden Problemen führen, sodass diese nicht mehr konvergieren würden.

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(\theta_l |x_{1l} - x_{2l}|^2\right)}$$

Wünschenswert wäre es also, dass der gesamte Bereich der reellen Zahlen verwendet werden könnte. Um dies zu erreichen, wurde als erstes versucht, das Quadrat der Hyperparameter zu verwenden:

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(\theta_l^2 |x_{1l} - x_{2l}|^2\right)}$$

Diese Formulierung führte allerdings zu einem unerwünschten Verhalten und zwar sieht die partielle Ableitung dieser Funktion wie folgt aus:

$$\frac{\partial c}{\partial \theta_l} = c\left(\vec{x}_1, \vec{x}_2\right)\left(-\theta_l \left|x_{1l} - x_{2l}\right|^2\right)$$

Das Problem hierbei ist, dass wenn der Hyperparameter während der Optimierung Null wird, auch die entsprechende partielle Ableitung Null wird. Während eines Minimierungsverfahrens kann es also passieren, dass die partiellen Ableitungen der Hyperparameter alle zu Null werden und die Minimierung als beendet angesehen wird. Um dieses Problem zu vermeiden, wurde die Exponentialfunktion verwendet:

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(e^{\theta_l}|x_{1l}-x_{2l}|^2\right)}$$

$$\frac{\partial c}{\partial \theta_l} = c\left(\vec{x}_1, \vec{x}_2\right)\left(-\frac{1}{2}e^{\theta_l}\left|x_{1l}-x_{2l}\right|^2\right)$$

Diese Formulierung der Gauss Korrelationsfunktion hat sich bisher als vorteilhaft herausgestellt, da der gesamte Raum der reellen Zahlen verwendet werden kann. Zudem ist die Funktion stetig und differenzierbar.

4.5 Maximum Likelihood für alle Kriging Verfahren

Noch keine Hyoperparameter genannt ????

An dieser Stelle wird nun die Annahme getroffen, dass die Daten einer Multivariaten Normalverteilung entsprechen. Diese Art der Verteilung bietet einige Vorteile:

- Blabla
- blabla

Zudem wird vorausgesetzt, dass die Erwartungswerte des stationären Zufallsprozesses bereits aus einem Likelihood Schätzer bekannt sind. Die anderen notwendigen Hyperparameter, welche für die Verteilung notwendig sind, sollen mit einer Maximierung des Likelihoods geschätzt werden.

Diese Art des Trainings unterscheidet sich von der Arbeit von Kennedy & O'Hagan und ebenfalls von der Arbeit von Keane.

Die Likelihood Funktion für das Co-Kriging hat dieselbe Form wie die Likelihood Funktion für das Ordinary Kriging. Wobei die Bildungsvorschrift des $\vec{F}$ Vektors in Gleichung **??** zu finden ist. Außerdem muss das β durch den High Fidelity Wert β_1 ersetzt werden. Der Term $\beta_1 \overrightarrow{F}$ entspricht hier dem $\vec{\beta}$ Vektor aus Kapitel **??**.

$$N = \frac{1}{(2\pi)^{\frac{n}{2}}\det\left(\mathbf{Cov}\right)^{\frac{1}{2}}} e^{-\frac{1}{2}\left(\vec{y}_s-\vec{F}\right)^T \mathbf{Cov}^{-1}\left(\vec{y}_s-\vec{F}\right)}$$

$$\log(N) = \log(1) - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log\left(\det(\mathbf{Cov})\right) - \frac{1}{2}\left(\vec{y}_s - \overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\left(\vec{y}_s - \overrightarrow{F}\right)$$

Da das Maximum der Funktion gesucht ist, können die Konstanten ignoriert werden:

$$\log(N) = -\log\left(\det(\mathbf{Cov})\right) - \left(\vec{y}_s - \overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\left(\vec{y}_s - \overrightarrow{F}\right)$$

Die Ableitung sieht dann wie folgt aus:

$$\frac{\partial L\left(\vec{h}\right)}{\partial h_l} = -\operatorname{Spur}\left(\mathbf{Cov}^{-1}\frac{\partial \mathbf{Cov}}{\partial h_l}\right) + \left(\left(\vec{y}_s - \overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\frac{\partial \mathbf{Cov}}{\partial h_l}\mathbf{Cov}^{-1}\left(\vec{y}_s - \overrightarrow{F}\right)\right)$$

Möchte man aus diesem reduzierten Likelihood, den Wert der Normalverteilung bestimmen:

$$\tilde{N} = -\log\left(\det(\mathbf{Cov})\right) - \left(\vec{y}_s - \overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\left(\vec{y}_s - \overrightarrow{F}\right)$$

$$\log(N) = \log(1) - \frac{n}{2}\log(2\pi) + \frac{1}{2}\tilde{N}$$

$$N = e^{\log(1) - \frac{n}{2}\log(2\pi) + \frac{1}{2}\tilde{N}}$$

Beispiel einer Dichtefunktion für ein CO-Kriging Model

Die Dichtefunktion eines CO-Kriging Modells der folgenden Parabelfunktion. Die Low-Fidelity Funktion wurde für jedes Sample mit einem zufälligen Wert zwischen -0.1 und 0.1 gestört und zusätzlich um einen konstanter Wert von 10 verschoben.

$$f_{high}(x) = x^2$$

$$f_{low}(x) = x^2 + Random(-0.1, 0.1) - 10$$

Auf den Achsen sind die Hyperparameter $\theta_{low}, \theta_{err}$ zu sehen und der entsprechende Dichtefunktionswert der Likelihoodfunktion ist farbig dargestellt. In den verschiedenen Diagrammen wurden die Varianzen $\sigma^2_{err}, \sigma^2_{low}$ der Kovarianzfunkionen cov_{err}, cov_{low} ver-

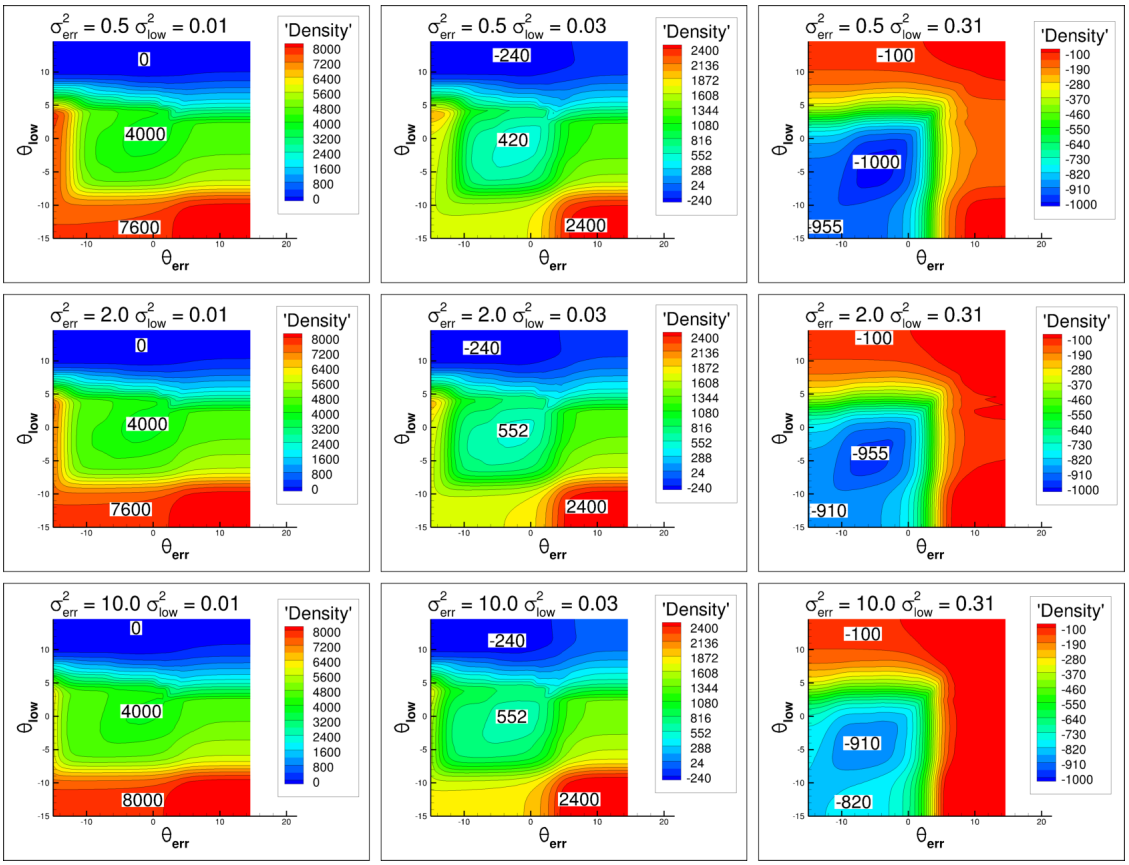


Abbildung 4.3:

ändert (siehe Gleichung **??**).

Dieses Beispiel ist interessant, da man die Hyperparameter und die Varianzen $\sigma^2_{err}, \sigma^2_{low}$ des CO-Kriging Modells in irgendeiner Form initialisieren muss. Beim Ordinary Kriging sind es nur die Hälfte der Hyperparameter und die Varianzen werden über einen Maximum Likelihood Schätzer bestimmt. Besonders schwer ist die Initialisierung der Varianzen

Training ist schwer, aufgrund lokaler Minima, daher ist die Initialisierung besonders wichtig.

Optimalwerte für THeta und mit angeben

Das Produkt der Inversen und der Korrelationsmatrix, ergibt die Einheitsmatrix. Da die Einheitsmatrix n Diagonalelemente besitzt, welche alle den Wert 1.0 haben, muss die Spur der multiplizierten Matrizen n ergeben. Gibt es numerische Ungenauigkeiten innerhalb der Invertierung, wird dieser Wert wahrscheinlich von n abweichen. Dies wird überprüft und bei Überschreitung eines Grenzwertes wird ebenfalls eine Exception geworfen. Diese wird mit allen anderen unbekannten Exceptions in Zeile 21 gefangen und als Reaktion ein sehr hoher Likelihood Wert zurückgegeben.

Nachdem alle Vektoren und Werte berechnet sind, wird in Zeile 26 die eigentliche

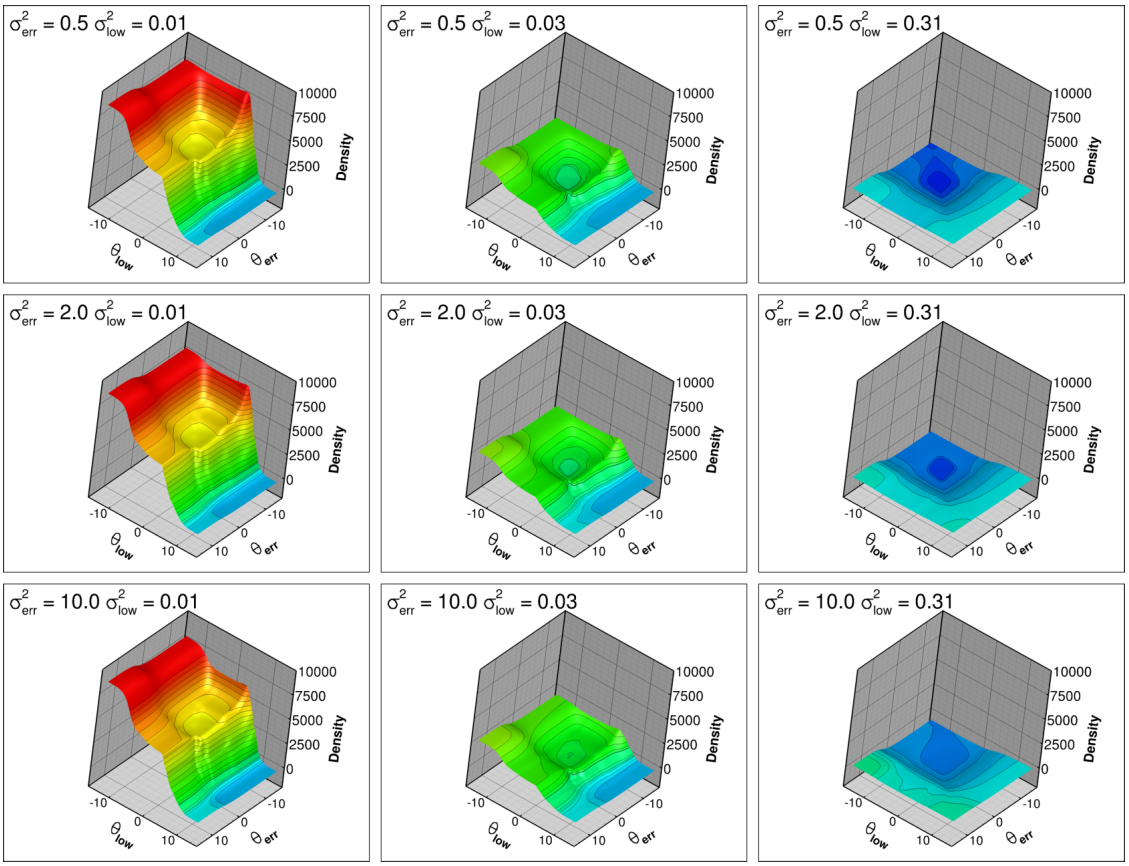


Abbildung 4.4:

Likelihood Funktion berechnet (siehe Gleichung **??**) und zurückgegeben.

4.5.1 Zusammenhang zwischen Bedingter Normalverteilung und Kriging

Wofür? Mit Christian besprechen

Grundsätzlich steckt in der Kriging Herleitung vorerst keinerlei Verteilungsannahme. Erst im Likelihood Verfahren kommt diese hinzu. Dennoch lässt sich eine starke Ähnlichkeit feststellen.

4.6 Regularisierungsterm und Nugget für alle Verfahren

4.7 Vergleiche zu anderen CO-Kriging Modellen

Unabhängig unsinnig -> Verweisen auf Han

Verschiedene Ansätze das zu modellieren:

- Gratiert

- Toal

Bei beiden an jeder High auch Low gerechnet werden muss

Ähnlich wie Kennedy und OHagang hier muss an jedem Hoigh auch Low

Keane setzt die Vorhersage an den unbekannten Stellen ein.

Wir trainieren alles in einem

Low Fidelity Daten beinhalten bereits einen Großteil der Information und die High-Fidelity Daten beinhalten Zusatzinformationen. Die LF Funktion wird als „glatter“ oder simplere Funktion angenommen. Eine komplexe LF Funktion und eine simplere HF Funktion sind für das hier verwendete Modell schwieriger zu trainieren.

Um die Kovarianzmatrix des CO-Kriging aufzustellen, sollte berücksichtigt werden, dass die Kovarianzfunktionen zwischen den verschiedenen Fidelities nicht völlig unabhängig voneinander sind. Beachtet man dies nicht, können die Kovarianzmatrizen nicht positiv definit werden und damit zu numerischen Problemen führen.

In [Han et al., 2012] wird ein Kriging Modell mit drei verschiedenen Korrelations- / Ko-

varianzfunktionen mit jeweils eigenen Hyperparametern trainiert. Dies führt zu erheblichen numerischen Problemen. In [Han et al., 2009] wurde vorgeschlagen, nur ein Hyperparametersatz für alle Kovarianzfunktionen zu trainieren. Vergleicht man dieses Modell mit einem Ordinary Kriging welches alle Fidelity Samples beinhaltet, besteht der einzige Unterschied nur noch in einem getrennten Erwartungswert/Varianz für die verschiedenen Fidelities. Dies stellt jedoch eine erhebliche Vereinfachung des Co-Kriging Modells dar.

Das eigentliche Problem hierbei besteht in der Abhängigkeit der Kovarianzfunktionen untereinander. Sprich, hat man zwei Kovarianzfunktionen festgelegt, kann die dritte nicht mehr frei gewählt werden, sondern wird durch die anderen beiden bestimmt.

Bei den braunschweigern, wird keine Normalverteilungsannahme getroffen. Das Modell ist dadurch in unserem enthalten, aber ....

Die Braunschweiger nutzen auch den MSE statt ....

Keane verletzt die Unabhänbgigkeitsbedinung aus O'Hagan

O'Hagan benötigt an jedem HF Punkt ien LF Sample zum Aufstellen der Differenz Kovarianzfunktion. Wir nicht

Gratiet ebenfalls ein eigener Ansatz, dieser soll hier aber nicht weiter diskutiert werden.

Erwartungstreue noch durch

4.4Die Formulierung in O'Hagan ist etwas anders, da dort das Beta2 dem Erwartungswert der Differenzfunktion entspricht. In diesem Fall entsprich das Beta2 dem HF Erwartungswert, am Ergebnis ändert dies allerdings nichts.

4.5 anders als bei den Braunschweigern, hier wurde der MSE der Vorhersage der HF Daten minimiert

-Braunschweiger [Zimmermann & Han, 2009][Han et al., 2012][Z.-H. Han, R. Zimmermann, 2010]

- Bei O'Hagan ist es zwingend, dass an jeder HF Stelle auch ein LF Sample existiert. Darin sehe ich einen relativ wichtigen Unterschied zu unserer Arbeit. (Nachzulesen am Ende von 2.2, die Definition der DesignPoints) - Dass die Likelihood Funktion mit einer vollständigen Matrix und den Varianzen als Hyperparameter dasselbe Minimum haben soll, wie die Definition der beiden einzelnen und unabhängigen Likelihood Funktionen bei O'Hagan, fällt mir nur schwer zu glauben. Zudem es bei uns auch keine direkten Schätzer für die Sigmas gibt. SigmaLF und SigmaHF können in unserem Fall durchaus von den jeweils anderen Samples beeinflusst werden und das passiert auch. Was das angeht bin ich mir allerdings nicht sicher, ob das überhaupt in Ordnung ist.

- Bei Keane werden die Vorhersagen aus dem reinen LoFi Modell verwendet um die Differenz-Kovarianzfunktion aufzustellen. Wenn ich mich nicht täusche, verletzt er damit die Bedingung von O'Hagan p(z|N) = p(z2|z1,s1,h2,sigma2)*p(z1|h1,sigma1), da p(z2|z1,s1,h2,sigma2) dadurch von h1,sigma1 und beta1 abhängt. Zudem bin ich mir

nicht sicher, inwiefern man die Vorhersagen ohne Beachtung der Unsicherheiten einsetzen sollte. Das wird wahrscheinlich schon funktionieren, mir geht es aber wie gesagt um die Unterschiede zu meiner Arbeit und auch darin sehe ich einen.

In [Kennedy & O'Hagan, 2000] werden die verschiedenen Daten als unabhängig angesehen (siehe Seite. Zudem können dadurch Likelihood Schätzer für die Erwartungswerte und Varianzen des Kriging Modells β_i, σ_i^2 mit $i = 1...m_f$ bestimmt werden. Durch diese Annahme kann man die Hyperparameter für das die Datensätze unabhängig voneinander bestimmen und im Nachhinein als gesamte Kovarianzfunktion zusammensetzen. Das Training der Hyperparameter wird dadurch vereinfacht und auch der numerische Aufwand sinkt dadurch. Dies kommt daher, da man zwei unabhängige Trainings durchführt mit jeweils einer Kovarianzmatrix der Größe des jeweiligen Datensatzes. Als Beispiel müssten bei einer Anzahl von n_{high} High-Fidelity und n_{low} Low-Fidelity Stützstellen jeweils ein Training mit den Kovarianzmatrizen $\mathbf{Cov_{high}} \in \mathbb{R}^{\mathrm{n_{high}} \times \mathrm{n_{high}}}$,$\mathbf{Cov_{low}} \in \mathbb{R}^{\mathrm{n_{low}} \times \mathrm{n_{low}}}$durchgeführt werden. Jedes dieser Trainings hat in etwa eine Komplexität von $\mathcal{O}\left(n^3\right)$.

Diese Annahme soll in dem Modell dieser Arbeit allerdings nicht getroffen werden. Der Grund dafür liegt darin, dass man den möglichen Raum der Hyperparameter (wobei o die Anzahl der Hyperparameter angibt) von $\mathbb{R}^{\mathrm{o_{high}} + \mathrm{o_{low}}}$ auf die beiden Räume $\mathbb{R}^{\mathrm{o_{high}}}$ und $\mathbb{R}^{\mathrm{o_{high}}}$auftrennt und damit mögliche Lösungen für die Hyperparameter ausschließt. Um dies zu verdeutlichen soll ein Beispiel herangezogen werden:

Zusammengesetzte Kovarianzfunktion zeigen, vielleicht zusätzlich ein Beispiel suchen welches mit dem anderen Modell nicht möglich wäre !!!!

Die Vorteile bei der Wahl dieser Methode liegen bei:

- Mehr „Lösungswege“ für das Training durch einen größeren Hyperparameterraum
- Unabhängigkeit der Daten ist in den meisten Fällen nicht gegeben
- Nur eine Kovarianzmatrix und ein Training, softwaretechnisch besser umsetzbar
- Höherwertigeres Modell
- An High Fidelity Punkten müssen keine Low-Fidelity Samples bekannt sein

- An High-Fidelity Punkten können Low-Fidelity Samples liegen

Die Nachteile bei der Wahl dieser Methode liegen bei:

- Numerisch aufwendiger
- Kein analytischer Likelihood Schätzer für σ_i^2 mit $i = 1...m_f$ gefunden

Welches Beta man einsetzt jängt davon ab, was man vorhersagen möchte. Möchte ich, dass mein $E\left[Z\left(\vec{x}_0\right)\right]$ dem HF Erwartungswert entsprich,. so muss dieser eingesetzt werden.

Die Formulierung in O'Hagan ist etwas anders, da dort das Beta2 dem Erwartungswert der Differenzfunktion entspricht. In diesem Fall entsprich das Beta2 dem HF Erwartungswert, am Ergebnis ändert dies allerdings nichts.

4.8 Vorhersagen Beispiele

Verweisen auf Benchmark Kapitel

5 Implementierung

In diesem Kapitel soll die Softwaretechnische Umsetzung des Algorithmus zur Bildung der Korrelationsmatrix dargestellt werden. Da in den Algorithmen sehr viele Matrix Operationen verwendet werden, wurde eine Matrix Klasse eingeführt. Diese wird am Anfang des Kapitels erläutert.

Im nächsten Abschnitt wird dann die Bildungsvorschrift der Matrix gezeigt und darauf folgend eine häufig genutzte Korrelationsfunktion und deren softwaretechnische Umsetzung erläutert.

Der letzter Abschnitt zeigt dann den eigentlichen Algorithmus zur Bildung der Korrelationsmatrix.

- Implementierung
    - SoftwareDesign (wenn Platz)
        - Programmiersprache, Umgebung, grob
        - AutoOpti Interface
        - Korrelationsfunktionen und Matrix für alle Verfahren
        - Training
        - UMLs und Code
    - Algorithmische Effizienz steigern
        - Vorwort (warum usw. GEK große Matrizen und COkriging v
        - Wiederverwenden von Korrelationswerten
        - SSE Beschleunigung Korrelationsfunktionen
        - Minimierungsverfahren beleuchten QuasiNewton / RPROP /
        - Verschiedenen Initialisierungen / Ultra Restart
        - Inverse durch Gleichungssystem ersetzen
        - Approximation der Spur
        - Rückwärtsdifferenzierung der partiellen Ableitungen
    - Verwendung von GPUs
        - GPU Computing vorstellen (Flops/Euro unschlagbar, zukü
        - CuBlas
        - Transferproblematik

- Symmetrie Copy Kernel
- K80 MultiGPU notwendig, ungeeignet für das Kriging
- CuBlasXT kann nicht sinnvoll verwendet werden,
- Mehrere Kriging Trainings auf verschiedenen / gleichen
- AdjointChodec als Beispiel

5.1 Softwaredesign

5.1.1 Offenes Softwaredesign

Integration von Supporting Vector Machines und GEK, COKriging, Ordinary Kriging in einem Code usw.

z.B. Kovarianzmatrix statt Korrelationsmatrix verwenden. Dadurch gibt es allerdings neue Schwieirigkeiten in der Bestimmung von den Sigmas, das günstigste war es diese mitzutrainieren.

5.1.2 Klasse zur Berechnung von Matrix Operationen

Da für das Training und Vorhersagen des Kriging Ersatzmodells hauptsächlich Matrix Operationen verwendet werden, ist es sinnvoll diese in einer Klasse zusammenzufassen. Außerdem sind für die Matrix Operationen verschiedene Implementierungen möglich, z.B. könnte es eine Klasse für OpenMP parallelisierte Operationen geben und eine andere Klasse wo dieselben Operationen über eine GPU (Graphics Processing Unit) berechnet werden. Daher werden die gemeinsamen Elemente in einer Superklasse Matrix zusammengefasst, siehe Abbildung 5.1. In dem UML Diagramm sind noch zwei andere Klassen erkennbar, eine Superklasse SaveableOnServer und eine Spezialisierung namens OpenMPMatrix.

SaveableOnServer sollte in einer modernen objektorientierten Programmiersprache wie z.B. Java ein kontextspezifisches anbietendes Client/Server Interface [Steimann et al., 2012] sein. Dies ist zur Verdeutlichung nochmals als UML Diagramm in Abbildung 5.2 dargestellt. In C++ muss ersatzweise eine abstrakte Klasse verwendet werden. Das Interface SaveableOnServer muss implementiert werden, um in späteren Anwendungen eine prozessweite Parallelisierung über eine Parallelisierungsbibliothek vornehmen zu können. Diese Parallelisierungsbibliothek wurde ebenfalls im Institut für Antriebstechnik entwickelt, allerdings außerhalb des Rahmens dieser Masterarbeit und soll daher nicht näher erläutert werden.

Die Matrix Klasse besitzt genau drei Attribute, welche als protected deklariert sind.

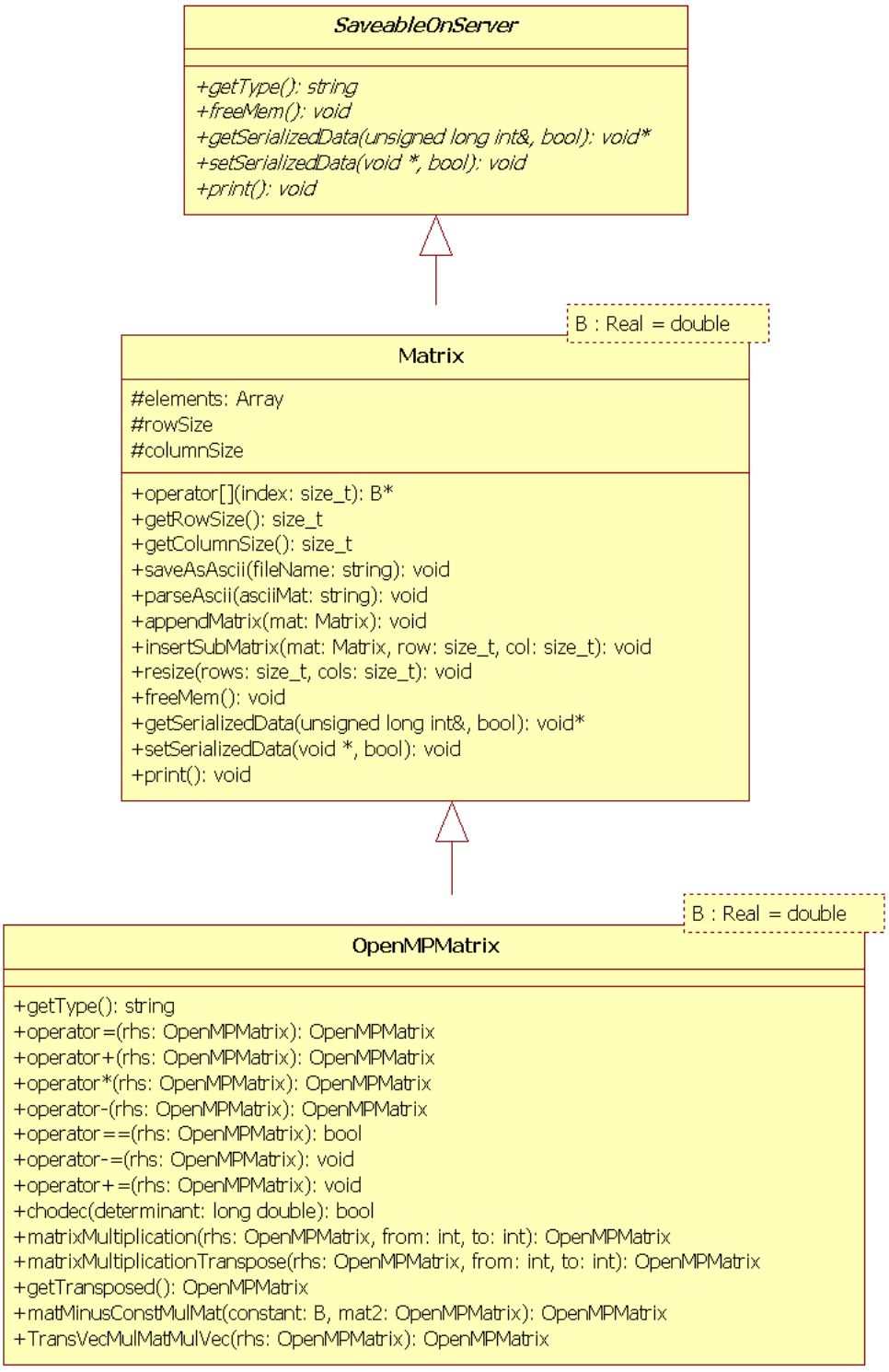


Abbildung 5.1: UML Diagramm der Matrixklasse mit einer Spezialisierung (OpenMPMatrix) und der abstrakten Klasse SaveableOnServer, welche prinzipiell ein Interface darstellt. Interfaces weden in C++ jedoch nicht unterstützt, daher wird eine abstrakte Klasse verwendet.

Das erste Attribut “elements” stellt ein eindimensionales Array dar und beinhaltet die Matrixelemente. Da eine Matrix einem zweidimensionalen Array entspricht, wird das eindimensionale Array auf ein zweidimensionales Array übersetzt. Dies wird gemacht, da so ein linearer Aufbau des Arrays garantiert gewährleistet wird und dies kann in einigen Systemen Geschwindigkeitsvorteile bringen. Der Aufbau des Array “elements” sieht wie folgt aus:

Elementnr.	Funktion
1	Zeilenanzahl
2	Spaltenanzahl
3	Dummy
4	Dummy
5 bis Spaltenanzahl	Erste Zeile der Matrix
Spaltenanzahl bis x*Spaltenanzahl	x’te Zeile der Matrix

Die Anzahl der Zeilen und Spalten sind also ebenfalls im Array gespeichert. Dies wird gemacht, um die Daten als einen Block zu serialisieren und damit als einen Datenstrom über das Netzwerk verschicken zu können.

Zusätzlich sind die Anzahl der Zeilen und Spalten als Attribut in der Klasse Matrix gespeichert, da auf diese Attribute sehr oft lesend zugegriffen wird und der Zugriff auf eine einzelne Variable schneller ist, als der Zugriff auf die beiden ersten Elemente des Arrays. Da die Attribute nur über Getter und Setter Methoden zugreifbar sind, kann immer gewährleistet werden, dass diese identisch sind. Die Übersetzung auf ein zweidimensionales Array wird dadurch erreicht, dass man den “[]” Operator der Matrix Klasse in folgender Weise überlädt:

```
T* operator[] (int row) {
    return this->elements + 4 + (row * this->getColumnSize());
}
```

Bei einem Zugriff auf den “[]” Operator wird also ein Zeiger auf das erste Element der entsprechenden Zeile zurückgegeben. Dies wird in C++ als ein Array interpretiert und das Array kann über den Operator “[]” die entsprechende Spalte liefern. Die Verwendung ist also dieselbe wie bei einem “normalen” zweidimensionalen C Array oder C++ Vektor.

Zusätzlich gibt es noch die Methoden saveAsAscii und parseAscii, welche es ermöglichen die Daten in eine Textdatei zu speichern und auszulesen. Zur Änderung der Größe der Matrix gibt es die Methode resize, welche die Matrix entsprechend vergrößert oder verkleinert. Die Methode appendMatrix hängt eine Matrix an die bestehende an und zwar an die letzte Zeile. Dies geht also nur, wenn die Spaltenanzahl identisch ist. Um eine Submatrix als Block in eine bestehende Matrix zu integrieren, gibt es die

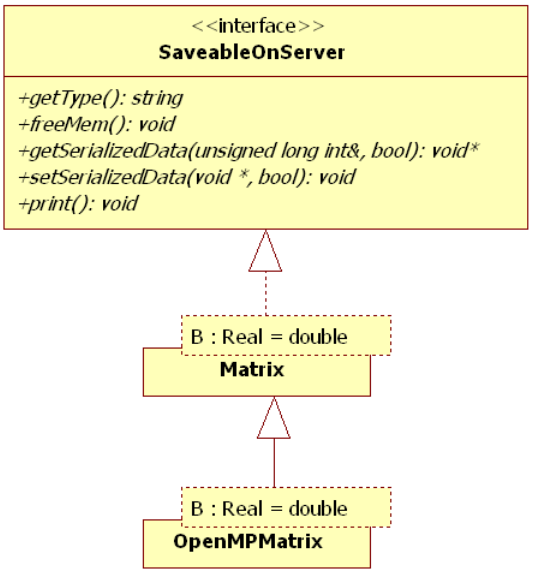


Abbildung 5.2: UML Diagramm der Matrix Klasse, wobei SaveableOnServer in diesem Fall durch ein kontextspezifisches Client Server Interface dargestellt wird.

Funktion insertSubMatrix. Diese fügt die als Parameter angegebene Matrix in die Bestehende ein.

In Abbildung 5.1 ist zusätzlich noch eine Spezialisierung namens OpenMPMatrix eingezeichnet, diese nutzt OpenMP zur Thread Parallelisierung und wird auch für das Kriging Modell verwendet. Im Wesentlichen wird in der Klasse OpenMPMatrix die Superklasse Matrix durch verschiedene Operatoren zur Addition, Multiplikation usw. erweitert und stellt so alle nötigen Operationen für das Kriging Modell bereit.

5.1.3 Korrelationsfunktionen

Wie bereits in Kapitel 4.4 beschrieben, sind für die Aufstellung der Kovarianzmatrix verschiedene Kovarianzfunktionsmodelle notwendig. Da diese für eine große Anzahl an Samples sehr häufig berechnet werden müssen, sollte deswegen ein besonderes Augenmerk auf die Effizienz dieser Berechnung gelegt werden.

Die programmiertechnische Umsetzung ist sehr simpel und sieht wie folgt aus:

```
for(size_t i=0; i<point1.getNumVars() ; i++){
    correl += fmath::exp(thetas[i]) * sqr(point1.getVarsRef(i) - point2.getVarsRef(i));
}
correl=fmath::exp(-0.5*correl);
```

Da diese Funktion während eines Trainings sehr häufig aufgerufen wird, macht es Sinn, diese zu beschleunigen. Um dies zu erreichen, wurden SSE (Streaming SIMD Extensions) CPU Befehle verwendet. Die Lesbarkeit des Codes leidet zwar recht stark darunter, da sich diese Methode zukünftig aber kaum noch ändern wird, ist dies vertretbar.

Streaming SIMD Extensions (SSE)

Die Streaming SIMD Extensions (SSE) sind eine von Intel entwickelte Befehlssatzerweiterung der x86-Architektur. Mit Einführung des Pentium-III-(Katmai)-Prozessors wurde diese 1999 vorgestellt. Aufgabe der SSE Befehle ist es Programme durch Parallelisierung auf Instruktionslevel zu beschleunigen, auch SIMD (Single Instruction Multiple Data) genannt. Die SSE-Befehlssatzerweiterung umfasst ursprünglich 70 Instruktionen und 8 neue Register, genannt XMM0 bis XMM7. Ursprünglich wurden die 128 Bit breiten Register allerdings nicht in einem Schritt verarbeitet. Bei heutigen CPUs (z.B. Intel Core CPUs) werden die Register in einem Schritt verarbeitet, zudem wurde die Anzahl der Register von 8 auf 16 erhöht.

Es gibt zahlreiche Umsetzungen der SSE Befehle. Diese reichen von SSE bis SSE5, wobei ab SSE3 AMD und Intel jeweils eigene Implementationen der SSE Architektur vornahmen. Der Nachfolger von SSE heißt AVX (Advanced Vector Extensions) und verbreitert die Register auf 16x 256 Bit.

Innerhalb dieser Arbeit wurden nur SSE Befehle verwendet, da diese praktisch von allen aktuellen CPUs und auch Compilern unterstützt werden. Für die Verwendung von AVX sind relativ neue Kompiler und CPUs notwendig, dies kann bei einigen Kunden zu Problemen führen. Durch die 128 Bit Register können nun in einem Rechenschritt vier float (32 Bit) oder zwei double (64 Bit) Werte gleichzeitig verarbeitet werden. Um diese Funktionen zu nutzen, müssen im C++-Code spezielle SSE Befehle verwendet werden [Fog, 2013, K.A., 2011]. Das folgende Listing zeigt die Umsetzung der Gauss Korrelationsfunktion mit SSE Befehlen, wobei die Parallelisierung hier über die Hyperparameter gemacht wird. Diese Methode wird zur Berechnung der Einträge der Kovarianzmatrix verwendet und wird dementspchrend oft aufgerufen. Aus diesem Grund ist es sinnvoll diese Methode zu Beschleunigen.

```
1  __m128d correlSSE=_mm_setzero_pd();
2  __m128d thetasExpSSE, point1SSE, point2SSE, pointDiffSSE;
3
4  array thetasExpArray;
5  array point1Array;
6  array point2Array;
7
8  for(i=0; i<point1.getNumVars()-1 ; i+=2){
9          thetasExpSSE =_mm_load_pd(&(thetasExpArray[i]));
10         point1SSE =_mm_load_pd(&(point1Array[i]));
11         point2SSE =_mm_load_pd(&(point2Array[i]));
12
13         pointDiffSSE = _mm_sub_pd(point1SSE,point2SSE);
14         pointDiffSSE = _mm_mul_pd(pointDiffSSE,pointDiffSSE);
15         pointDiffSSE = _mm_mul_pd( thetasExpSSE, pointDiffSSE  );
16         correlSSE = _mm_add_pd( correlSSE,  pointDiffSSE );
17 }
18
19 correlSSE = _mm_hadd_pd(correlSSE,correlSSE);
20 _mm_store_sd(&correl,correlSSE);
```

0	1	2	3	4	5
64Bit	64Bit	64Bit	64Bit	64Bit	64Bit
$\underbrace{128Bitaligned}$					

Tabelle 5.1: 128Bit Speicherausrichtung

```
21 for(; i<point1.getNumVars() ; i++) {
22         correl += thetasExp[0][i] * (point1.getVar(i) - point2.getVar(i))*(point1.getVar(i) - point2.getVar(
23 }
24
25 correl=fmath::expd(-0.5*corre);
```

In den Zeilen 1-2 werden verschiedene Variablen definiert vom Typ "__m128d", dieser Typ stellt ein 128 Bit großes SSE Datentypen dar und kann zwei 64 Bit double Werte aufnehmen. Zudem wird die Variable correlSSE mithilfe der Funktion_mm_setzero_pd() auf 0 gesetzt. Die Zeilen 4-6 stellen Arrays dar, welche für die Berechnung der Korrelationsfunktion benötigt werden. Das Array thetasExpArray beinhaltet die berechneten Hyperparameter e^{θ_l}. Da diese Werte für alle Einträge in der Kovarianzmatrix identisch sollten sie daher vor dem Belegen der Kovarianzmatrix berechnet werden. Danach werden die beiden Arrays initialisiert, welche die Ortsvariablen $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^k$ beinhalten. Die darauffolgende for-Schleife iteriert über die Anzahl an freien Variablen. Der Zähler wird hier immer um den Wert 2 erhöht, da mit den SSE Routinen 2 double Werte gleichzeitig berechnet werden können. In den Zeilen 9-11 werden 128Bit aus den Arrays an der Stelle i in die SSE Register übertragen. Aus diesem Grund muss der Speicher zwingend 128Bit Speicherausrichtung besitzen.

Einschub: Speicherausrichtung In aktuellen C++ Compilern wird eine Speicherausrichtung von 128Bit im Normalfall gewährleistet. Die folgende Tabelle soll die Speicherausrichtung und die damit entstehende Problematik beim Programmieren verdeutlichen:

Die erste Zeile der Tabelle beschreibt den Index eines normalen Arrays mit 6 Einträgen und jeder Eintrag hat die Größe eines 64Bit (z.B. double) Werts. Der Compiler garantiert in diesem Fall einen zusammenhängenden Speicher von 128Bit, dargstellt durch die dritte Zeile.

In Zeile 5 wird die Variable correlSSE mit der Funktion auf Null gesetzt. Da in einem Schritt immer zwei Befehle gleichzeitig ausgeführt werden, muss die Anzahl durch zwei teilbar sein. Ist dies nicht der Fall, muss der Rest mit normalen Befehlen durchgeführt werden. Zeile 7 setzt die Anzahl der Schleifendurchläufe auf die Anzahl der freien Variablen geteilt durch zwei. Ist die Anzahl der freien Variablen ungerade, wird eins subtrahiert und dann durch zwei geteilt. Die Zeilen 8-16 stellen die Summation aus der Gauss Formel dar. Zuerst wird in den Zeilen 9-10 zwei Werte der Stützstellen in

die SSE Variablen geladen. Dies geschieht mit der Funktion _mm_loadu_pd(). Als Parameter erwartet die Funktion eine Speicheradresse und transferiert dann 128 Bit ab dieser Adresse in die entsprechende SSE Variable. In den Zeilen 11-12 wird die Exponentialfunktion der Hyperparameter berechnet. Dies geschieht auf konventionelle Weise, da es keinen SSE Befehl für die Exponentialfunktion gibt. Die beiden berechneten Werte werden dann in Zeile 13 ebenfalls in eine SSE Variable geladen. Die Differenz der Stützstellenkomponenten wird anschließend in Zeile 14 vorgenommen. Hier werden wie bereits erwähnt, direkt zwei Differenzen gleichzeitig berechnet. In Zeile 15 wird dann die Differenz quadriert, mit den Hyperparametern multipliziert und aufsummiert, alles mit SSE Befehlen.

Ist der Schleifendurchlauf beendet, wird in Zeile 17 das Ergebnis in eine normale Variable zurück transferiert und in den restlichen Zeilen wird (falls die Anzahl der freien Variablen nicht durch zwei teilbar war) das letzte Element mit konventionellen Methoden berechnet. Die gemessenen Geschwindigkeitsvorteile lagen bei ca. 20 % - 30 %.

5.1.4 Algorithmus zur Bildung der Korrelationsmatrix für alle Kriging Verfahren

Beim Aufstellen der Korrelationsmatrix müssen alle Korrelationen zwischen Membern berechnet werden. Ein Member wird durch die Klasse Point beschrieben, in Abbildung 5.3 ist das UML Diagramm der entsprechenden Klasse dargestellt.

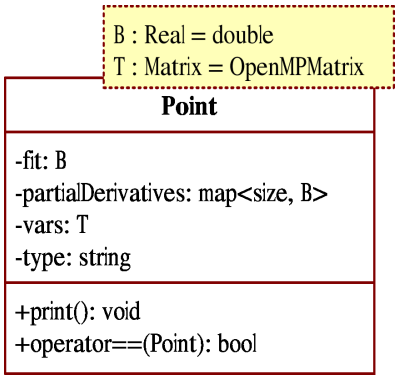


Abbildung 5.3: UML Diagramm der Klasse Point, welche einen Member/Stützstelle beschreibt

Die Klasse verwendet zwei Template Parameter. Zum einen den Parameter B, dieser gibt den verwendeten Fließkommazahlentyp an und zum anderen gibt es den Parameter T, welcher den verwendeten Matrix Typ darstellt. Der Matrix Typ muss ein Subtyp der Matrix Klasse aus Kapitel **??** sein. Die Point Klasse besteht aus vier Attributen und zwei öffentlichen Methoden. Das Attribut fit beschreibt den Funktionswert des Members. PartialDerivatives beinhaltet die vorgegebenen partiellen Ableitungen eines Members. Der Typ des Attributs ist eine map (Hashtabelle), wobei der Schlüssel die Nummer der Variable darstellt, nach der abgeleitet wurde. In vars werden die Variablenwerte des Members gespeichert, der Typ des Attributs ist eine Matrix. Eigentlich wird nur ein Vektor als Typ benötigt, da man aber gerne die Matrix Operationen nutzen möchte, wird vars als einspaltige Matrix verwendet, was letztlich wieder einem Vektor entspricht. Das Attribut type beinhaltet den Typ des Members, dies ist für Variable Fidelity Methods (siehe Kapitel **??**) wichtig. In diesem Attribut wird dann über einen Identifier angegeben, ob es sich z.B. um einen Member handelt, welcher mit hoher Güte berechnet wurde oder niedriger.

Eine Korrelationsfunktion soll zwischen zwei solcher Point Objekte einen Korrelationswert bestimmen. Daher macht es Sinn, eine gemeinsame abstrakte Superklasse einzuführen, der Aufbau ist in Abbildung 5.4 dargestellt. Da jedes Point Objekt auch die partiellen Ableitungen enthält ist es sinnvoll, dass die Korrelationsfunktion alle Korrelationen zwischen zwei Point Objekten zurück gibt, also auch die Korrelationen zwischen partiellen Ableitungen und Funktionswerten. Zu diesem Zweck wird die Korrelationsmatrix durch Zeilen- und Spaltentausch umsortiert, um die Korrelationen zwischen

zwei Point Objekten direkt als Submatrix in die Gesamtmatrix einzufügen:

$$\mathbf{R} = \left[\begin{array}{c|cccc} & Z\left(\vec{x}_1\right) & \frac{\partial Z(\vec{x}_1)}{\partial x_1^1} & Z\left(\vec{x}_2\right) & \frac{\partial Z(\vec{x}_2)}{\partial x_1^1} \\ \hline Z\left(\vec{x}_1\right) & c\left(\vec{x}_1,\vec{x}_1\right) & \frac{\partial c(\vec{x}_1,\vec{x}_1)}{\partial x_1^1} & c\left(\vec{x}_1,\vec{x}_2\right) & \frac{\partial c(\vec{x}_1,\vec{x}_2)}{\partial x_2^1} \\ \frac{\partial Z(\vec{x}_1)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_1)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_1)}{\partial x_1^1 \partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_2)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_2)}{\partial x_1^1 \partial x_2^1} \\ Z\left(\vec{x}_2\right) & c\left(\vec{x}_2,\vec{x}_1\right) & \frac{\partial c(\vec{x}_2,\vec{x}_1)}{\partial x_1^1} & c\left(\vec{x}_2,\vec{x}_2\right) & \frac{\partial c(\vec{x}_2,\vec{x}_2)}{\partial x_2^1} \\ \frac{\partial Z(\vec{x}_2)}{\partial x_1^1} & \frac{\partial c(\vec{x}_2,\vec{x}_1)}{\partial x_2^1} & \frac{\partial c(\vec{x}_2,\vec{x}_1)}{\partial x_2^1 \partial x_1^1} & \frac{\partial c(\vec{x}_2,\vec{x}_2)}{\partial x_2^1} & \frac{\partial c(\vec{x}_2,\vec{x}_2)}{\partial x_2^1 \partial x_2^1} \end{array}\right] \tag{5.1}$$

Durch diese Anordnung stehen alle Korrelationen zwischen zwei bestimmten Punkten immer zusammen, dies vereinfacht den Algorithmus zum Aufstellen der Matrix erheblich.

Kapitel/C:/home/andreas/documents/GoogleDrive/Promotion/Diss/images/UML/

Abbildung 5.4: UML Diagramm der abstrakten Klasse CorrelationFunction mit zwei Subklassen, diese stellen spezifische Korrelationsfunktionen dar

Die Methode CreateCorrelationMatrix

Das UML Diagramm in Abbildung 5.4 zeigt die abstrakte Superklasse CorrelationFunction mit zwei Spezialisierungen namens CorrelationFunctionGauss und CorrelationsFunctionSpline. Die Spezialisierungen stellen jeweils verschiedene Korrelationsfunktionen dar. Die Umsetzung der Gauss Korrelationsfunktion wurde im Abschnitt 5.1.3 dargestellt, auf die Beschreibung der Spline Funktion wird hier verzichtet. Die Methode getAllCorrelation schreibt alle Korrelationswerte zwischen den Stützpunkten P1 und P2 in die ebenfalls übergebene Korrelationsmatrix corrMat. Da die Korrelationswerte zwischen zwei Stützstellen eine Submatrix der Korrelationsmatrix darstellen (siehe Formel 5.1), wird der Methode getAllCorrelation die entsprechende Stelle der Submatrix in der Korrelationsmatrix über die Indizes i und j mit angegebenen. Die beiden Parameter stellen hier die Position der ersten Zeile und Spalte der Submatrix in der Korrelationsmatrix dar. Es wird also im ersten Schritt die Korrelation zwischen den beiden Punkten bestimmt, dann die Korrelationen der partiellen Ableitungen, daraus wird dann die entsprechende Submatrix gebildet und diese in die Korrelationsmatrix eingefügt.

Das folgende Programmlisting zeigt die Funktion, welche unter Benutzung der abstrakten Klasse CorrelationFunktion, eine Korrelationsmatrix mit Werten befüllt. Der gezeigte Code ist der Originalcode in C++, auf Pseudocode wird hier verzichtet da einige Besonderheiten von C++ eine recht große Rolle in der Programmierung spielen. Einige Zeilenumbrüche wären in C++ in der Form nicht zulässig, aus Platzgründen ließen sich diese allerdings nicht vermeiden.

```
1  template <class T, class B>
2  void createCorrelationMatrix(
3                      T &correlationMatrix,
4                      vector<Point<T,B> > &points,
5                      map<string,map<string,CorrelationFunction<T,B>*>> correlationMap){
6
7      T tmp(config::numSamplesDerivatives, config::numSamplesDerivatives);
8      correlationMatrix = tmp;
9      #pragma omp parallel for schedule(dynamic)
10     for(size_t i=0; i<points.size(); i++){
11         for(size_t j=i; j<points.size(); j++){
12             if(i==j){
13                 correlationMap[points[i].getType()][points[j].getType()]->getAllCorrelation(
14                         points[i],
15                         points[j],
16                         correlationMatrix,
17                         0.0,
18                         config::matrixPositions[i],
19                         config::matrixPositions[j],
20                         true );
21                 correlationMatrix[config::matrixPositions[i]][config::matrixPositions[j]]
22                                 += fmath::expd(config::diagonalAddition);
23             }
24             else{
25                 correlationMap[points[i].getType()][points[j].getType()]->getAllCorrelation(
26                     points[i],
```

```
27                      points[j],
28                      correlationMatrix,
29                      0.0,
30                      config::matrixPositions[i],
31                      config::matrixPositions[j],
32                      false);
33              }
34          }
35      }
36
37      #pragma omp parallel for
38      for(int i=correlationMatrix.getColumnSize()-1; i>0; i--){
39          for(int j=i-1; j>=0; j--){
40              correlationMatrix[i][j] = correlationMatrix[j][i];
41          }
42      }
43  }
```

Die Funktion createCorrelationMatrix hat genau drei Parameter. Der erste Parameter ist eine Referenz auf die eigentliche Korrelationsmatrix. Um unnötiges Kopieren zu vermeiden, wird eine Referenz verwendet, weil die Daten direkt in die Matrix geschrieben werden sollen. Da die Matrizen sehr groß werden können (20000x20000 sind keine Seltenheit) und während des Trainings sehr häufig gebildet werden müssen, können solche Überlegungen erhebliche Unterschiede in der Geschwindigkeit ausmachen.

Der Parameter Points vom Typ Point (Abbildung 5.3) ist ein eindimensionaler Vektor, welcher alle Member/Stützstellen beinhaltet.

CorrelationMap ist eine zweidimensionale Hashtabelle, die beiden Indizes sind Integer und beschreiben die Typen (vgl. Attribut type aus Abbildung 5.3) der entsprechenden Stützstellen. Durch diese Hashtabelle ist es möglich, jedem Paar von Stützstellen verschiedene Korrelationsfunktionen zuzuordnen. Dies ist bei der weiteren Entwicklung des Verfahrens von großer Bedeutung, insbesondere für Variable Fidelity Models (siehe Kapitel **??**). Der Wert der Hashtabelle ist vom Typ CorrelationFunction (siehe 5.4), dieser kann also durch jeden Subtyp der abstrakten Klasse CorrelationFunction überladen werden. Eine solche correlationMap könnte z.B. folgendermaßen aufgebaut sein:

Typ1	Typ2	CorrelationFunction
0	0	correlationFunctionGauss
0	1	correlationFunctionSpline
1	1	correlationFunctionGauss

Würde man mit dieser correlationMap z.B. folgenden Aufruf machen, so würde man die Methode getAllCorrelation des SubTyps CorrelationFunctionGauss (siehe 5.4) aufrufen.

```
correlationMap[0][0]->getAllCorrelation(...);
```

Genau diesen Aufruf findet man in den Zeilen 13 und 25.

In Zeile 7 und 8 wird eine neue Matrix vom Template Typ T allokiert und der Parameter correlationMatrix damit überschrieben. Der Wert config::numSamplesDerivatives beschreibt hier die Anzahl der Stützstellen plus die Anzahl der gegebenen partiellen Ableitungen.

In den Zeilen 9 bis 11 wird eine doppelte for Schleife über alle Stützstellen gestartet, um die Korrelationen von allen Stützstellen zu allen Stützstellen zu berechnen. Das "#pragma omp parallel for" ist eine einfache Schleifenparallelisierung von OpenMP. Die Indizes werden automatisch in Bereiche eingeteilt und dann in einzelnen Threads abgearbeitet. Die Anzahl der maximalen Threads wird direkt zu Anfang über ein Parameterfile festgelegt. Der zusätzliche Befehl "schedule(dynamic)" gibt an, wie der OpenMP Scheduler die Arbeit auf die Threads verteilt, es gibt prinzipiell drei Varianten:

- *static*: Jede Teilschleife besitzt eine feste Anzahl von Durchläufen, diese Durchläufe werden dann reihum an die Threads verteilt. Dieses Vorgehen ist, bei gleicher Lastverteilung der Teilschleifen optimal. Im Normalfall gibt es soviele Teilschleifen wie Anzahl Threads.
- *dynamic*: Hier werden die Teilschleifen dynamisch an die Threads verteilt, um das zu erreichen, werden die Teilschleifen kleiner gewählt als z.B. bei static. Dies ist sinnvoll, wenn die Last stark variiert, allerdings ist der Verwaltungsaufwand für die OpenMP Laufzeitumgebung höher.
- *guided*: Bei diesem Fall werden die Teilschleifen während der Laufzeit exponentiell von groß zu klein verändert. Dies ist ein Spezialfall von *dynamic* und reduziert den Verwaltungsaufwand.

Welche der drei Varianten für einen Fall geeignet ist, lässt sich oftmals nur durch Testen gut bestimmen. Bei der Belegung der Korrelationsmatrix scheint zuerst die Option *static* sinnvoller zu sein, da der Aufwand für einen Schleifendurchlauf in etwa gleich bleibt. Allerdings war die Option *dynamic* in Tests ca. 10 % schneller. Das könnte durch interne Compiler Optimierungen und die Verwendung des If- Else Blocks innerhalb der Schleife zu erklären sein.

Der If- Else Block ist dazu da, um zwischen der Berechnung der Korrelation zwischen zwei gleichen Stützpunkten und der Berechnung der Korrelation zwischen zwei verschiedenen Punkten zu unterscheiden. Dies ist sinnvoll, da sich die Korrelation zwischen zwei gleichen Stützpunkten durch Vereinfachungen der mathematischen Formulierung deutlich schneller berechnen lässt. Eine solche Vereinfachung wird im nächsten Abschnitt genauer beschrieben. Zusätzlich kann auf die Diagonalelemente der Matrix, ein Diagonalaufschlag addiert werden (Zeile 21-22). Dies wird verwendet um die Matrix für die Invertierung numerisch stabiler zu machen, in Kapitel 5.3.2 wird der Diagonalaufschlag genauer erläutert.

Innerhalb des If- Else Blocks wird die Methode getAllCorrelation der entsprechenden Subklasse von CorrelationFunction (siehe Abbildung 5.4) aufgerufen. Zwei wichtige Parameter dieses Methodenaufrufs sind "*config::matrixPositions[i]*" und "*config::matrixPositions[j]*", diese Arrays enthalten eine Tabelle, welche zu einer gegebenen Membernummer die entsprechende Position der Member in der Korrelationsmatrix zurückgeben soll. Da ein Member in der Korrelationsmatrix durch eine kleinere Submatrix beschrieben wird, stellt die Position immer die erste Zeile bzw. Spalte der Submatrix in der Korrelationsmatrix dar. Die Methode getAllCorrelation wird im folgenden Abschnitt genauer erläutert.

Da die Korrelationsmatrix symmetrisch ist, wird nur die rechte obere Dreiecksmatrix belegt und in den Zeilen 37-43 wird diese dann auf die linke untere kopiert. Dadurch wird die Geschwindigkeit des Algorithmus nochmals erhöht.

Die Methode getAllCorrelation

Die Methode getAllCorrelation, welche in Subklassen des Typs CorrelationFunction (siehe Abbildung 5.4) definiert ist, soll alle Korrelationswerte zwischen zwei Stützstellen berechnen. Diese werden in die Korrelationsmatrix eingetragen und von der Funktion createCorrelationMatrix (siehe Kapitel 5.1.4) aufgerufen. Das folgende Listing zeigt den Originalcode in C++.

```
1  template <class T,class B>
2  inline void CorrelationFunctionGauss<T,B>::getAllCorrelation(
3          Point<T,B> &point1,
4          Point<T,B> &point2,
5          T &corrMatrix,
6          B diag,
7          size_t iMatrix,
8          size_t jMatrix,
9          bool equalMember){
10
11     typename map<size_t, B>::iterator derIt;
12     typename map<size_t, B>::iterator derItCol;
13     size_t freevarNr=0, freevarNr2=0;
14     if(equalMember){
15          corrMatrix[iMatrix][jMatrix]= 1.0;
16          for(size_t i=0; i<point2.getNumPartDerivatives();i++){
17             freevarNr=i;
18             corrMatrix[iMatrix+i+1][jMatrix+i+1] = fmath::expd(thetas[0][freevarNr]);
19          }
20     }else{
21          corrMatrix[iMatrix][jMatrix] = calcSimpleGauss(point1, point2);
22          size_t i=0;
23          size_t j=0;
24          for(derItCol=point2.getAllPartDervsRef().begin();
25                    derItCol!=point2.getAllPartDervsRef().end();derItCol++){
26
27             freevarNr=derItCol->first;
28             corrMatrix[iMatrix][jMatrix+i+1] = calcGEKPartialDerivative(point1, point2,
29                                                   corrMatrix[iMatrix][jMatrix], freevarNr);
30
31             i++;
32          }
33          if(corrMatrix.getRowSize()>1){
34             i=0;
35             for(derItCol=point1.getAllPartDervsRef().begin();
36                       derItCol!=point1.getAllPartDervsRef().end();derItCol++){
37
38                freevarNr=derItCol->first;
39                corrMatrix[iMatrix+i+1][jMatrix] = -calcGEKPartialDerivative(point1, point2,
40                                                      corrMatrix[iMatrix][jMatrix], freevarNr);
41                i++;
42             }
43             i=0;
44             for(derItCol=point1.getAllPartDervsRef().begin();
45                       derItCol!=point1.getAllPartDervsRef().end(); derItCol++){
46                freevarNr=derItCol->first;
47                j=0;
48                for(derIt=point2.getAllPartDervsRef().begin();
49                   derIt!=point2.getAllPartDervsRef().end();derIt++){
50
51                   freevarNr2 = derIt->first;
52                   if(i==j)
53                      corrMatrix[iMatrix+i+1][jMatrix+j+1]=calcGEKPartialDerivative2(point1,
54                                                            point2,
55                                                            corrMatrix[iMatrix][jMatrix],
56                                                            freevarNr);
57                   else
58                      corrMatrix[iMatrix+i+1][jMatrix+j+1] = calcGEKPartialDerivative2(point1,
59                                                               point2,
60                                                               corrMatrix[iMatrix][jMatrix],
```

```
61                                                                  freevarNr ,
62                                                                  freevarNr2 );
63                         j++;
64                     }
65                     i++;
66                 }
67             }
68         }
69 }
```

Die Methode wird mit sieben Parametern aufgerufen. Die ersten beiden Parameter sind Referenzen auf zwei Objekte vom Typ Point. Zwischen diesen beiden Stützstellen sollen alle entsprechenden Korrelationswerte berechnet werden. Der nächste Parameter corrMatrix ist eine Referenz auf die Korrelationsmatrix, in welche die entsprechenden Korrelationswerte geschrieben werden sollen. Wie bereits im vorherigen Abschnitt ist der Parameter diag, ein Diagonalaufschlag für die Korrelationsmatrix. Dieser wird auf die Hauptdiagonale der Matrix addiert und kann für die numerische Stabilität der Invertierung von Bedeutung sein, in Kapitel 5.3.2 wird der Diagonalaufschlag genauer erläutert. Die nächsten beiden Parameter iMatrix und jMatrix geben an, bei welchen Indizes in der Korrelationsmatrix die neuen Korrelationen eingefügt werden sollen.

Der letzte Parameter equalMember ist ein bool'scher Wert und gibt an, ob es sich bei den beiden Stützstellen um dieselben Punkte handelt. Ist dies der Fall, kann die Berechnung der Korrelationswerte stark vereinfacht werden.

In den Zeilen 11-13 werden einige Variablen deklariert. Insbesondere zwei Iteratoren, welche zum iterieren über die partiellen Ableitungen der beiden Stützstellen dienen. Dies ist nötig, da die partiellen Ableitungen als map gespeichert sind und eine map in C++ nur über Iteratoren durchlaufen werden kann.

Die Zeilen 14-19 werden ausgeführt, wenn die beiden Stützstellen identisch sind. Dann vereinfachen sich die Korrelationswerte für eine Gauss Korrelation wie sie in Kapitel 5.1.3 beschrieben wurde zu:

$$c\left(\vec{x}_1, \vec{x}_1\right) = 1$$

$$\frac{\partial c\left(\vec{x}_1, \vec{x}_1\right)}{\partial x_1^p} = 0$$

$$\frac{\partial c\left(\vec{x}_1, \vec{x}_1\right)}{\partial x_1^p \partial x_1^j} = \begin{cases} 0 & p \neq j \\ e^{\theta_k} & p = j \end{cases}$$

Übertragen auf das Beispiel aus Gleichung 5.1, vereinfacht sich die entsprechende Korrelationsmatrix dann zu:

$$R = \left[\begin{array}{c|cccc} & Z(\vec{x}_1) & \frac{\partial Z(\vec{x}_1)}{\partial x_1^1} & Z(\vec{x}_2) & \frac{\partial Z(\vec{x}_2)}{\partial x_1^1} \\ \hline Z(\vec{x}_1) & 1 & 0 & c(\vec{x}_1, \vec{x}_2) & \frac{\partial c(\vec{x}_1, \vec{x}_2)}{\partial x_2^1} \\ \frac{\partial Z(\vec{x}_1)}{\partial x_1^1} & 0 & e^{\theta_k} & \frac{\partial c(\vec{x}_1, \vec{x}_2)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1, \vec{x}_2)}{\partial x_1^1 \partial x_2^1} \\ Z(\vec{x}_2) & c(\vec{x}_2, \vec{x}_1) & \frac{\partial c(\vec{x}_2, \vec{x}_1)}{\partial x_1^1} & 1 & 0 \\ \frac{\partial Z(\vec{x}_2)}{\partial x_1^1} & \frac{\partial c(\vec{x}_2, \vec{x}_1)}{\partial x_2^1} & \frac{\partial c(\vec{x}_2, \vec{x}_1)}{\partial x_2^1 \partial x_1^1} & 0 & e^{\theta_k} \end{array}\right]$$

Diese Vereinfachung gilt nur für die Gauss Korrelationsfunktion, für andere Korrelationsfunktionen ergeben sich aber ähnliche Vereinfachungen.

Handelt es sich bei den beiden Stützstellen allerdings nicht um dieselben, wird der else-Fall ab Zeile 21 aufgerufen. In dieser Zeile wird dann der einfache Korrelationswert der Matrix gebildet und in die gesamte Korrelationsmatrix eingetragen. Der Code für die private Methode calcSimpleGauss ist identisch mit dem Code für die Gauss Korrelationsfunktion aus Kapitel 5.1.3.

In den Zeilen 24-32 wird die Ableitung der Korrelationsfunktion zwischen den Stützstellen gebildet, abgeleitet wird nach den Parametern der zweiten Stützstelle. Die Schleife geht alle freien Variablen durch, an denen es eine partielle Ableitung gibt. Die eigentlichen Korrelationswerte werden in der privaten Methode calcGEKPartialDerivative berechnet und danach in die gesamte Korrelationsmatrix geschrieben, in der oberen Beispielmatrix würde das dem Wert $\frac{\partial c(\vec{x}_1, \vec{x}_2)}{\partial x_2^1}$ aus der rechten oberen Ecke entsprechen, da es in dem Beispiel nur eine freie Variable gibt. Der Methode calcGEKPartialDerivative werden anschließend die beide Punkte übergeben. Der genaue Funktionsablauf soll hier aus Platzgründen nicht weiter aufgeführt werden.

Die Abfrage in Zeile 33 prüft, ob es sich bei der Korrelationsmatrix um einen Vektor handelt. Ist dies der Fall, sind nachfolgenden Berechnungen nicht notwendig.

In den Zeilen 35-42 wird wie bereits in den Zeilen 24-32 die Ableitung der Korrelationsfunktion berechnet. In diesem Fall allerdings für die erste Stützstelle, dies würde in der Beispielmatrix dem Wert $\frac{\partial c(\vec{x}_1, \vec{x}_2)}{\partial x_1^1}$ entsprechen.

In den Zeilen 44-66 werden die zweiten Ableitungen der Korrelationsfunktion gebildet. Dafür muss über die freien Variablen von beiden Stützstellen iteriert werden, an denen sich partielle Ableitungen befinden. Die Berechnung der Ableitungen findet in der privaten Methode calcGEKpartialDerivative2 statt. Es gibt zwei verschiedene Implementationen der Methode, einmal eine für den Fall, dass die freien Variablen, nach denen abgeleitet wird für beide Punkte gleich sind und einmal für den Fall, dass sich diese unterscheiden. In der Beispielmatrix würde die Ableitung dem Wert $\frac{\partial c(\vec{x}_1, \vec{x}_2)}{\partial x_1^1 \partial x_2^1}$ entsprechen.

Zudem besitzen die beiden Punkte jeweils nur eine freie Variable und auch nur jeweils eine partielle Ableitung. Daher ist die Nummer der freien Variablen nach denen abgeleitet wird, in beiden Fällen eins und damit würde die entsprechende Implementation aus den Zeilen 52-56 aufgerufen werden.

Diese Methode stellt einen recht allgemeingültigen Algorithmus auf, welcher alle notwendigen Korrelationswerte zwischen zwei Stützstellen in die korrekten Positionen einer Korrelationsmatrix einfügt. Im UML Diagramm 5.4 wurde zusätzlich zur Methode getAllCorrelation eine Methode getAllCorrelationPartialDer gelistet. Diese soll die Ableitungen der Korrelationsfunktion nach den Hyperparametern zurückgeben. Dies ist für das Training, welches in Kapitel 5.2 erklärt wird, wichtig. Die eigentliche Methode ist der Methode getAllCorrelation allerdings relativ ähnlich und soll daher nicht näher dargestellt werden.

5.1.5 Bestimmung der Korrelationen bei Verwendung der Kovarianzmatrix und Co-Kriging

Für einige Anwendungen ist es notwendig die reale Korrelation zwischen zwei beliebigen Samples zu kennen. Im Co-Kriging hat man natürlich das Problem, dass diese erst einmal unbekannt sind, da mit der Kovarianzmatrix

5.1.6 Bestimmung der Hyperparameter durch die Maximum Likelihood Methode

Wie in den vorherigen Kapiteln gezeigt wurde, hängen die einzelnen Korrelationswerte der Korrelationsmatrix maßgeblich von den verwendeten Hyperparametern ab und damit die Güte des Kriging Modells. Ziel eines Kriging Trainings ist es daher, die optimalen Hyperparameter zu finden. Um dies zu erreichen, wird die Maximum Likelihood Methode verwendet.

Im ersten Teil des Kapitels soll die Maximum Likelihood Methode anhand eines simplen Beispiels erklärt werden. Im Anschluss daran wird die Umsetzung dieser Methode für das hier verwendete Kriging Modell gezeigt.

Der letzte Abschnitt behandelt dann die softwaretechnische Umsetzung dieser Methode.

5.1.7 Likelihood

Um den Likelihood Term und seine partiellen Ableitungen zu bilden, wird eine eigene Klasse vorgesehen. Da mehrere Likelihood Funktionen denkbar wären, wird eine abstrakte Klasse namens DensityFunction eingeführt. Abbildung 5.5 zeigt die Umsetzung der Klasse als UML Diagramm, Getter und Setter Methoden wurden hier aus Platzgründen ausgelassen. Bisher ist nur die Likelihood Funktion umgesetzt (siehe Kapitel **??**). Der Likelihood Term (Gleichung **??**) wird in der Methode calcDensity() berechnet und die entsprechende Ableitung (Gleichung **??**) in calcDensityDerivative(). Denkbar wären allerdings auch andere Likelihood Funktionen, welche auf anderen Verteilungen basieren.

Ziel der Klassenstruktur ist es, die beiden Gleichungen **??** und **??** so effizient wie möglich zu lösen. Die dort verwendeten Matrizen sind in der Regel sehr groß und voll besetzt, was eine effiziente Berechnung sehr wichtig macht. Allerdings sind die Matrizen symmetrisch und positiv definit, was wiederum einige Optimierungen zulässt. Zur Vereinfachung werden einige Terme der Gleichungen zusammengefasst:

$$\vec{R}_f = \mathbf{R}^{-1}\overrightarrow{F} \tag{5.2}$$

$$f_{rf} = \vec{F}^T\vec{R}_f \tag{5.3}$$

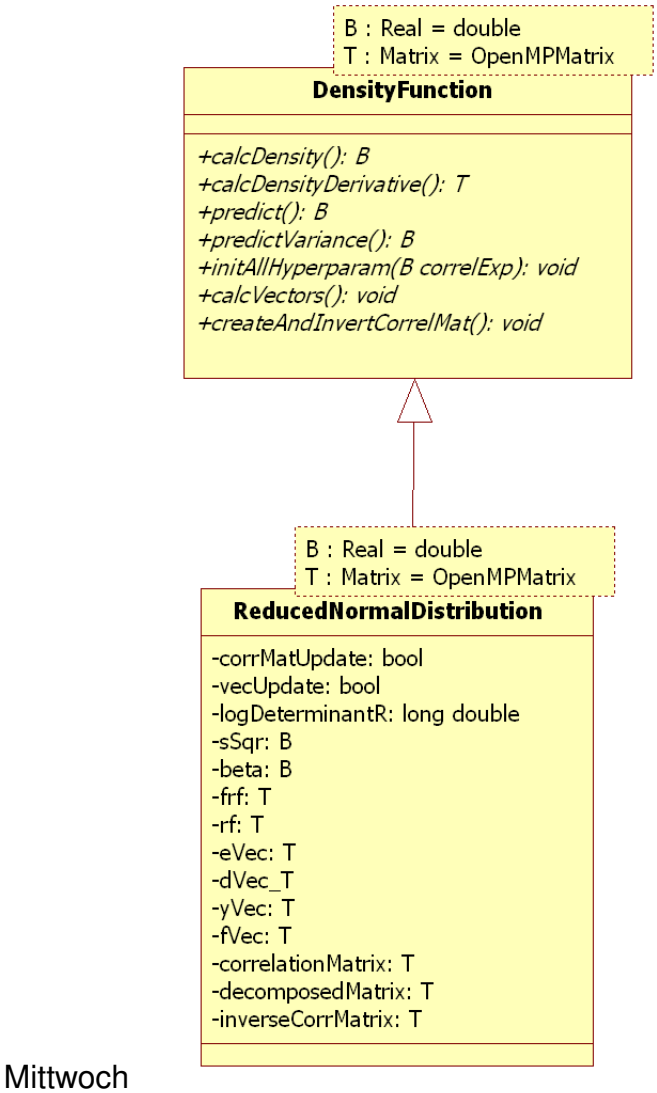
$$\overrightarrow{e} = (\vec{y}_s - \beta * \overrightarrow{F}) \tag{5.4}$$

$$\overrightarrow{d} = \mathbf{R}^{-1}\overrightarrow{e} \tag{5.5}$$

Die Terme werden in diese Form auch in der entsprechenden Klasse (ReducedNormalDistribution) einmal berechnet und dann gespeichert. Da diese sehr häufig wiederverwendet werden, muss man diese Termen nur einmal berechnen und eine Änderung ist nur notwendig, wenn die Hyperparameter verändert wurden. Über die Attribute corrMatUpdate und vecUpdate wird festgelegt, ob die Matrizen und Vektoren neu berechnet werden müssen oder nicht. Sie werden nur dann neu berechnet, wenn die Hyperparameter verändert wurden. Da die Hyperparameter nur über Getter und Setter Methoden zugänglich sind, kann man bei jedem Setter Zugriff auf die Hyperparameter die Attribute corrMatUpdate und vecUpdate auf true setzen.

Viele der Attributnamen entsprechen den hier verwendeten Bezeichnungen für die Vektoren/Matrizen, z.B. der Vektor $\vec{R}_f$ entspricht dem Attribut rf. Das Attribut logDeterminantR entspricht dem Logarithmus der Determinante der Korrelationsmatrix $\log(\det(\mathbf{R}))$. Aufgrund dieser Ähnlichkeit werden daher nicht alle einzeln aufgeführt.

Wie bereits in Kapitel 5.1.2 beschrieben, lässt sich die Determinante durch eine Cholesky Zerlegung sehr effizient berechnen. Insbesondere da die Zerlegung ebenfalls für die Invertierung der Matrix sinnvoll ist. Die Korrelationsmatrix wird in der Methode createAndInvertCorrelmat aufgestellt, zerlegt und dann invertiert. Die zerlegte Matrix wird in decomposedMatrix gespeichert, die invertierte Matrix in inverseCorrMatrix. Die benötigten Vektoren werden in der Methode calcVectors() berechnet und in den Attributen der Klasse gespeichert. Die Methoden predict() und predictVariance() berechnen dann unter Vorgabe eines Ortsvektors eine Schätzung der gesuchten Funktion $y^*(\vec{x}_0)$ (siehe Gleichung **??**) und der Varianz (Gleichung **??**).



Mittwoch

Abbildung 5.5: UML Diagramm der abstrakten Superklasse DensityFunction und der Subklasse ReducedNormalDistribution

Schritte für die Berechnung eines Likelihood Terms:

In diesem Abschnitt soll die Methode zur Berechnung des Likelihood Terms (calcDensity()) nochmals genauer erklärt werden. Das folgende Listing zeigt die Methode im Originalcode:

```
1  template <class T, class B>
2  B ReducedNormalDistribution<T,B>::calcDensity(){
3      try{
4          if(this->corrMatUpdate){
5              this->createAndInvertCorrelMat();
6              this->corrMatUpdate = false;
7          }
8          if(this->vecUpdate){
9              this->calcVectors();
10             this->vecUpdate = false;
```

```
11          }
12      }
13      catch(ChodecNotPosDef& e){
14          cout <<"calcDensity() MatrixExceptions:"<<e.what()<<endl;
15          config::diagonalAddition = log(2.0*exp(config::diagonalAddition));
16
17          this->getCorrelationMatrixRef().saveAsAscii("corrMatFailed");
18          this->getInverseCorrelationMatrixRef().saveAsAscii("corrMatInverseFailed");
19          return  (config::numSamples*1000.0);
20      }
21      catch(...){
22          cout <<"calcDensity() Exception"<<endl;
23          cout <<"-----------------------------------"<<endl;
24          return  (config::numSamples*1000.0);
25      }
26      return 0.5*(log(sSqr)*config::numSamples + logDeterminantR + config::numSamples );
27  }
```

In Zeile 4 wird zuerst überprüft, ob die Korrelationsmatrix neu aufgestellt werden muss oder nicht. Dies geschieht, wie bereits beschrieben, mit dem Attribut corrMatUpdate-Ist dieses true, dann wird die Korrelationsmatrix wie in Zeile 5 über die Methode createAndInvertCorrMat() erzeugt, eine Cholesky Zerlegung durchgeführt und dann invertiert. Nach erfolgreichem Aufruf der Methode wird das Attribut corrMatUpdate wieder auf false gesetzt. Ansonsten werden die Matrizen im Puffer verwendet, also correlationMatrix, decomposedMatrix und inverseCorrMat.

Auf dieselbe Weise wird mit den Vektoren (Gleichungen 5.2-5.5) in den Zeilen 8-10 verfahren. Diese werden durch die Methode calcVectors erzeugt und dann in den Attributen der Klasse gespeichert. Durch die verwendete Matrix Klasse ist die Berechnung der einzelnen Vektoren/Matrizen sehr simpel. Das folgende Listing zeigt dies exemplarisch an der Methode zur Erzeugung von Vektor $\vec{d}$.

```
template <class T, class B>
void ReducedNormalDistribution<T,B>::calcDVec(){
    dVec = inverseCorrelationMatrix.matrixMultiplicationTranspose(eVec);
}
```

Die Methode ist Mitglied der Klasse ReducedNormalDistribution (siehe Abbildung 5.5) und wird durch die Methode calcVectors() aufgerufen. Das Attribut inverseCorrelation-Matrix ist vom Typ Matrix und beinhaltet die inverse Korrelationsmatrix. Diese wird mit dem transponierten Vektor $\overrightarrow{e} = (\vec{y}_s - \beta * \overrightarrow{F})$ multipliziert, die Transposition wird innerhalb der Multiplikation vorgenommen. Die anderen Methoden zur Berechnung der Dichtefunktionswerte usw. beinhalten prinzipiell nur andere Matrix Operationen und werden daher nicht alle aufgeführt.

Die Methode InitAllThetas() der Klasse DensityFunction und deren Subklassen soll alle Hyperparameter mit möglichst sinnvollen Werten initialisieren. Die verschiedenen Intialisierungsmöglichkeiten und deren Umsetzung werden in Abschnitt 5.3.2 noch genauer erläutert.

Da die Cholesky Zerlegung nur für positiv definite symmetrische Matrizen funktioniert, kann es bei der Zerlegung zu einer Exception vom Typ ChodecNotPosDef kommen, dies wird in Zeile 13 abgefangen. Ist diese Exception aufgetreten, wird der Diagonalaufschlag (siehe Kapitel 5.3.2) erhöht und ein sehr hoher Likelihood Wert zurückgegeben (Zeile 19), damit dieser in der Minimierung nicht mehr berücksichtigt wird.

Zusätzlich wird innerhalb der Matrix Klasse nach erfolgreicher Invertierung eine kurze Überprüfung der invertierten Matrix gemacht. Dies wird durch folgende Gleichung erreicht:

$$\text{Spur}\left(RR^{-1}\right) = n$$

Das Produkt der Inversen und der Korrelationsmatrix, ergibt die Einheitsmatrix. Da die Einheitsmatrix n Diagonalelemente besitzt, welche alle den Wert 1.0 haben, muss die Spur der multiplizierten Matrizen n ergeben. Gibt es numerische Ungenauigkeiten innerhalb der Invertierung, wird dieser Wert wahrscheinlich von n abweichen. Dies wird überprüft und bei Überschreitung eines Grenzwertes wird ebenfalls eine Exception geworfen. Diese wird mit allen anderen unbekannten Exceptions in Zeile 21 gefangen und als Reaktion ein sehr hoher Likelihood Wert zurückgegeben.

Nachdem alle Vektoren und Werte berechnet sind, wird in Zeile 26 die eigentliche Likelihood Funktion berechnet (siehe Gleichung **??**) und zurückgegeben.

5.2 Minimierungsverfahren/Training

Im Kapitel **??** wurde die Maximum Likelihood Methode vorgestellt. Als Ergebnis dieses Kapitels erhielt man zwei Gleichungen zur Berechnung des Likelihood Terms und der dazugehörigen Ableitung. Ziel ist es, für den Likelihood Term die optimalen Hyperparameter zu finden. Dieser Vorgang ist das eigentliche Training des Modells. Zu diesem Zweck werden zwei numerische Minimierungsverfahren vorgestellt. Beide Minimierungsverfahren waren bereits in einer institutseigenen Bibliothek vorhanden. Die entwickelte Minimierungsklasse sollte beide Verfahren nutzen können. Hierfür wurde ein spezielles Klassenmodell unter Benutzung von Boost Funktionsobjekten entwickelt.

Das Training bringt einige zusätzliche Probleme mit sich, z.B. müssen die Hyperparameter anfangs initialisiert werden. Zudem kann die Korrelationsmatrix schlecht konditioniert sein, für beide Probleme wurden Lösungsansätze entwickelt, welche hier vorgestellt werden.

5.2.1 Vermeidung negativer Hyperparameter

In diesem Abschnitt soll darauf eingegangen werden, welche Probleme negative Hyperparameter hervorrufen und wie man diese vermeiden kann.

Zudem wurde in vorherigen Kapiteln bereits ein Diagonalaufschlag erwähnt, dieser wird auf die Diagonalelemente der Korrelationsmatrix addiert und soll die Kondition der Matrix verbessern. Die genaue Vorgehensweise soll in diesem Abschnitt erläutert werden.

Negative Hyperparameter

Während der Minimierung der Likelihood Funktion kann es je nach gewähltem Minimierungsverfahren oftmals dazu kommen, dass die Hyperparameter negativ werden können. Bei der Gauss Korrelationsfunktion würde dies dazu führen, dass bei großen Abständen zwischen den Membern auch große Korrelationen vorhergesagt werden würden. Die Korrelationen könnten so auch einen Wert über Eins erhalten, was keinen Sinn macht. Daher sind negative Hyperparameter bei der Gauss Funktion zwingend zu vermeiden.

Eine einfache Möglichkeit dies zu tun, wäre während der Minimierung die Hyperparameter nach unten zu begrenzen. Dies kann bei einigen Minimierungsverfahren allerdings zu schwerwiegenden Problemen führen, sodass diese nicht mehr konvergieren würden.

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(\theta_l |x_{1l} - x_{2l}|^2\right)}$$

Wünschenswert wäre es also, dass der gesamte Bereich der reellen Zahlen verwendet werden könnte. Um dies zu erreichen, wurde als erstes versucht, das Quadrat der Hyperparameter zu verwenden:

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(\theta_l^2 |x_{1l} - x_{2l}|^2\right)}$$

Diese Formulierung führte allerdings zu einem unerwünschten Verhalten und zwar sieht die partielle Ableitung dieser Funktion wie folgt aus:

$$\frac{\partial c}{\partial \theta_l} = c\left(\vec{x}_1, \vec{x}_2\right)\left(-\theta_l \left|x_{1l} - x_{2l}\right|^2\right)$$

Das Problem hierbei ist, dass wenn der Hyperparameter während der Optimierung Null wird, auch die entsprechende partielle Ableitung Null wird. Während eines Minimierungsverfahrens kann es also passieren, dass die partiellen Ableitungen der Hyperparameter alle zu Null werden und die Minimierung als beendet angesehen wird. Um dieses Problem zu vermeiden, wurde die Exponentialfunktion verwendet:

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum\limits_{l=1}^{l<k}\left(e^{\theta_l}\left|x_{1l} - x_{2l}\right|^2\right)}$$

$$\frac{\partial c}{\partial \theta_l} = c\left(\vec{x}_1, \vec{x}_2\right)\left(-\frac{1}{2}e^{\theta_l}\left|x_{1l} - x_{2l}\right|^2\right)$$

Diese Formulierung der Gauss Korrelationsfunktion hat sich bisher als vorteilhaft herausgestellt, da der gesamte Raum der reellen Zahlen verwendet werden kann. Zudem ist die Funktion stetig und differenzierbar.

5.2.2 Konvergenzkriterium für das Kriging Training

5.3 Analytische Bestimmung der Kriging Varianz beim Ordinary Kriging

Durch die Umformulierung der Korrelationsmatrix zur Kovarianzmatrix, muss der Maximum Likelihood Schätzer der globalen Varianz σ^2 für eine multivariate Normalverteilung umformuliert werden:

$$\sigma_{k+1}^2 = \frac{1}{n}\left(\vec{y}_s - \beta\overrightarrow{F}\right)^T \sigma_k^2 \mathbf{Cov}^{-1}\left(\vec{y}_s - \beta\overrightarrow{F}\right)$$

$$\sigma_{k+1}^2 = \frac{1}{n}\left(\vec{y}_s - \beta\overrightarrow{F}\right)^T \sigma_k^2 \frac{1}{\sigma_k^2}\mathbf{Corr}^{-1}\left(\vec{y}_s - \beta\overrightarrow{F}\right)$$

Wobei k den aktuellen Iterationsschritt angibt.

Hierbei gibt es allerdings einen Nachteil. Es ist notwendig bei der ersten Iteration einen Startwert für σ_k^2 zu bestimmen. Mit diesem Startwert wird nun $\mathbf{Cov}^{-1}$ bestimmt. Nachdem $\mathbf{Cov}^{-1}$ bestimmt wurde, wird der Likelihood Schätzer für σ_{k+1}^2 berechnet und damit ist das korrekte σ_{k+1}^2 bekannt. Die Kovarianzmatrix wurde allerdings noch mit dem σ_k^2 aufgestellt und ist somit ungültig. Es wäre an dieser Stelle also nötig die Kovarianzmatrix neu aufzustellen und alle Schritte ein zweites Mal zu durchlaufen.

Bei einem Minimierungsverfahren ist das meist nicht notwendig, da die Schätzungen sich nicht sehr stark verändern. Bei einer zufälligen Initialisierung kann dies allerdings zu Problemen führen, die Update Schritte sollten in diesem Fall öfter wiederholt werden.

5.3.1 Regularisierungsterm und Nugget für alle Kriging Modelle

Diagonalaufschlag einstellen über eine maximale Konditionszahl

Während des Trainings kann es passieren, dass Hyperparameter eingestellt werden, die eine schlechte Konditionierung der Korrelationsmatrix hervorrufen. Dies kann bei der Cholesky Zerlegung problematisch werden. Um die Konditionszahl zu verbessern, wird ein Diagonalaufschlag auf die Hauptdiagonale der Matrix addiert. Die Gauss Korrelationsfunktion würde sich damit wie folgt ändern ($\delta_{i,j}$ beschreibt hier das Kronecker Delta und λ den Diagonalaufschlag):

$$c\left(\vec{x}_i, \vec{x}_j\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(e^{\theta_l}\left|x_{i,l}-x_{j,l}\right|^2\right)} + e^{\lambda}\delta_{i,j}$$

Es stellt sich allerdings die Frage, wie groß dieser Diagonalaufschlag im Einzelfall sein muss und inwiefern dieser das Ergebnis beeinflusst bzw. die Kondition verbessert. Grundsätzlich sollte dieser so klein wie möglich gewählt werden, um die Ursprungsmatrix so wenig wie möglich zu verändern. Um einen optimalen Wert zu ermitteln, gibt es zwei verschiedene Möglichkeiten.

Die erste Möglichkeit wäre, die Likelihood Funktion (siehe Gleichung **??**) nach dem Diagonalaufschlag zu differenzieren. Die Ableitung ist in diesem Fall sehr simpel, da alles außer dem Diagonalaufschlag wegfällt:

$$\frac{\partial c\left(\vec{x}_i, \vec{x}_j\right)}{\partial \lambda} = e^{\lambda}\delta_{i,j}$$

Die Ableitung der Korrelationsmatrix nach dem Diagonalaufschlag ergibt also eine Ein-

heitsmatrix multipliziert mit dem Diagonalaufschlag:

$$\frac{\partial \mathbf{R}}{\partial \lambda} = e^{\lambda}\mathbf{E}$$

Damit ergibt sich die Ableitung der Likelihood Funktion nach dem Diagonalaufschlag zu:

$$\frac{\partial L}{\partial \lambda} = \frac{e^{\lambda}}{2}\left[\text{Spur}\left(\mathbf{R}^{-1}\right) - \frac{1}{\sigma^2}(\vec{y}_s - \beta * \overrightarrow{F})^T\mathbf{R}^{-1}\mathbf{R}^{-1}(\vec{y}_s - \beta * \overrightarrow{F})\right]$$

Mit dieser Formel wäre es nun möglich, den Diagonalaufschlag einfach als zusätzlichen Hyperparameter zu minimieren. Allerdings bedeutet dies natürlich einen zusätzlicher Aufwand. Es soll daher nach einer einfacheren Methode gesucht werden.

Ein anderer möglicher Ansatz ist die Konditionszahl der Matrix. Diese ist definiert als Quotient aus maximalem und minimalem Eigenwert der Korrelationsmatrix:

$$\kappa = \left|\frac{\Xi_{max}\left(\mathbf{R}\right)}{\Xi_{min}\left(\mathbf{R}\right)}\right|$$

Überlegt man sich nun, welche Korrelationsmatrix am schlechtesten konditioniert wäre, kommt man auf die Einsmatrix:

$$\mathbf{R} = \begin{bmatrix} 1 & ... & 1 \\ ... & ... & ... \\ 1 & ... & 1 \end{bmatrix}$$

Die minimalen und maximalen Eigenwerte der Einsmatrix sind bekannt:

$$\Xi_{min}\left(\mathbf{R}\right) = 0$$

$$\Xi_{max}\left(\mathbf{R}\right) = n$$

Dies würde einer unendlichen Konditionszahl entsprechen:

$$\kappa = \infty$$

Addiert man nun den Diagonalaufschlag, bekommt man folgende Korrelationsmatrix:

$$\mathbf{R} = \begin{bmatrix} 1+e^{\lambda} & ... & 1 \\ ... & ... & ... \\ 1 & ... & 1+e^{\lambda} \end{bmatrix}$$

Daraus ergeben sich die folgenden neuen maximalen und minimalen Eigenwerte:

$$\Xi_{min}\left(\mathbf{R}\right) = e^{\lambda}$$

$$\Xi_{max}\left(\mathbf{R}\right) = e^{\lambda} + n$$

Und die entsprechende Konditionszahl verbessert sich zu:

$$\kappa = \left| \frac{e^{\lambda} + n}{e^{\lambda}} \right|$$

Da die Matrix positiv definit sein muss und damit nur positive Eigenwerte hat, kann der Betrag weggelassen werden:

$$\kappa = \frac{e^{\lambda} + n}{e^{\lambda}}$$

Wählt man nun für die Konditionszahl eine obere Grenze, bekommt man eine Untergrenze für den Diagonalaufschlag:

$$\kappa_{max} > \frac{e^{\lambda} + n}{e^{\lambda}}$$

$$e^{\lambda} > \frac{n}{(\kappa_{max} - 1)}$$

Eine geeignete Grenze für die obere Grenze der Konditionszahl ist aus der Erfahrung bekannt und liegt bei ca. 10^9, dieser Wert kann im Einzelfall natürlich angepasst werden. Der daraus resultierende Diagonalaufschlag würde bei einer üblichen Matrixgröße von 5000x5000, $5 * 10^{-6}$ betragen und es ist davon auszugehen, dass dieser so gut wie keinen Einfluss auf das Endergebnis haben sollte.

Diese Art der Berechnung des Diagonalaufschlags wird momentan im Code verwendet. Die entsprechende Ableitung der Likelihood Funktion ist im Code bereits umgesetzt, wird momentan allerdings nicht verwendet, da bisherige Tests dafür noch keine Notwendigkeit zeigten.

Diagonalaufschlag als Faktor

Multipliziert man nun den Diagonalaufschlag, bekommt man folgende Korrelationsmatrix:

$$\mathbf{R} = \begin{bmatrix} 1\lambda & ... & 1 \\ ... & ... & ... \\ 1 & ... & 1\lambda \end{bmatrix}$$

Daraus ergeben sich die folgenden neuen maximalen und minimalen Eigenwerte:

$$\Xi_{min}\left(\mathbf{R}\right) = 1$$

$$\Xi_{max}\left(\mathbf{R}\right) = \lambda + n - 1$$

Und die entsprechende Konditionszahl verbessert sich zu:

$$\kappa = \left| \frac{\lambda + n - 1}{\lambda - 1} \right|$$

Da die Matrix positiv definit sein muss und damit nur positive Eigenwerte hat, kann der Betrag weggelassen werden:

$$\kappa = \frac{\lambda + n - 1}{\lambda - 1}$$

Wählt man nun für die Konditionszahl eine obere Grenze, bekommt man eine Unter-

grenze für den Diagonalaufschlag:

$$\kappa_{max} > \frac{\lambda + n - 1}{\lambda - 1}$$

$$\lambda > \frac{n - 1 + \kappa_{max}}{(\kappa_{max} - 1)}$$

5.3.1.1 Likelihood Probleme durch multiplikativen Diagonalaufschlag

5.3.2 Initialisierung der Hyperparameter für alle Kriging Modelle

Um einen Minimierungsalgorithmus starten zu können, ist eine geeignete Initialisierung der Hyperparameter von großer Bedeutung. Diese kann die Konvergenz und auch die Stabilität der Minimierung stark beeinflussen. Innerhalb dieser Arbeit wurden mehrere Ansätze entwickelt, um eine geeignete Initialisierung zu finden.

Abschätzung konstanter Hyperparameter

Eine sehr einfache und schnelle Möglichkeit die Hyperparameter für eine Gauss Verteilung zu schätzen, wäre einen Erwartungswert für die Einträge in der Korrelationsmatrix zu wählen. Da die Korrelation grundlegend zwischen Eins und Null liegen sollte, ist dies recht einfach. Angenommen der Mittelwert der Korrelationsfunktion soll bei einem Wert von $c_{erw} = \{c_{erw} \in \mathbb{R} | 0 \leq c_{erw} \leq 1\}$ liegen.

$$c_{erw} = \frac{1}{n^2} \sum_{i=1}^{i<n} \sum_{j=1}^{j<n} e^{-\frac{1}{2} \sum_{l=1}^{l<k} \left(e^{\theta_l} \left| x_{i,l} - x_{j,l} \right|^2 \right)}$$

Nimmt man weiterhin an, dass die einzelnen Korrelationswerte nahezu identisch sind:

$$c_{erw} = e^{-\frac{1}{2} \sum_{l=1}^{l<k} \left(e^{\theta_l} \left| x_{i,l} - x_{j,l} \right|^2 \right)}$$

$$\log\left(c_{erw}\left(\vec{x}_i,\vec{x}_j\right)\right) = -\frac{1}{2}\sum_{l=1}^{l<k}\left(e^{\theta_l}\left|x_{i,l}-x_{j,l}\right|^2\right)$$

Nimmt man ferner an, dass x eine Realisierung einer Zufallsvariablen ist und bildet den Erwartungswert:

$$\log\left(c_{erw}\left(\vec{x}_i,\vec{x}_j\right)\right) = E\left[-\frac{1}{2}\sum_{l=1}^{l<k}\left(e^{\theta_l}\left|x_{i,l}-x_{j,l}\right|^2\right)\right]$$

Als weitere Vereinfachung sollen alle Hyperparameter den gleichen Wert haben:

$$\log\left(c_{erw}\left(\vec{x}_i,\vec{x}_j\right)\right) = -\frac{1}{2}e^{\theta}E\left[\sum_{l=1}^{l<k}\left(\left|x_{i,l}-x_{j,l}\right|^2\right)\right]$$

Die beiden Variablen werden als Zufallsvariablen angenommen und der Betrag wird aufgrund des Quadrats vernachlässigt:

$$\log\left(c_{erw}\right) = -\frac{1}{2}e^{\theta}E\left[\sum_{l=1}^{l<k}\left(x_{i,l}-x_{j,l}\right)^2\right]$$

$$\log\left(c_{erw}\right) = -\frac{1}{2}e^{\theta}\sum_{l=1}^{l<k}\left(E\left[x_{i,l}^2\right]-E\left[2x_{i,l}x_{j,l}\right]+E\left[x_{j,l}^2\right]\right)$$

Nimmt man nun an, dass die Zufallsvariablen unabhängig sind, gilt $E\left[2x_{i,l}x_{j,l}\right]=0$

$$\log\left(c_{erw}\right) = -\frac{1}{2}e^{\theta}\sum_{l=1}^{l<k}\left(E\left[x_{i,l}^2\right]+E\left[x_{j,l}^2\right]\right)$$

Die Varianz einer Zufallsvariable X ist definiert durch $\mathrm{var}\left(X\right)=E\left[X^2\right]-E\left[X\right]^2$. Die im Modell verwendeten Daten werden grundsätzlich auf einen Erwartungswert von Null und eine Standardabweichung von Eins normiert.

$$E[X] = 0$$

$$\mathbf{var}[X] = 1$$

Daraus ergibt sich folgende Formel für das verwendete Modell für die Varianz der Stützstellen:

$$\mathbf{var}[X] = E\left[X^2\right]$$

Also

$$E\left[x_{i,l}^2\right] = \mathbf{var}\left[x_{i,l}^2\right] = 1$$

und analog dazu:

$$E\left[x_{j,l}^2\right] = \mathbf{var}\left[x_{j,l}^2\right] = 1$$

Daraus folgt:

$$\log\left(c_{erw}\right) = -\frac{1}{2}e^{\theta}\sum_{l=1}^{l<k}(1+1)$$

$$\log\left(c_{erw}\right) = -\frac{1}{2}e^{\theta}2k$$

$$\log\left(-\frac{\log\left(c_{erw}\right)}{k}\right) = \theta \tag{5.6}$$

Mit dieser Formel hat man nun eine Möglichkeit die Hyperparameter zu schätzen. Aufgrund der vielen Annahmen und Vereinfachungen ist dieses Verfahren als heuristisch

einzustufen. Die Hyperparameter haben dann allerdings alle denselben Initialwert. Zudem ist der erwartete Korrelationswert c_{erw} unbekannt, dies kann leicht durch einfaches Ausprobieren gelöst werden, da der Wertebereich bekannt ist. Man würde also c_{erw} von 0 bis 1 variieren, damit einen Hyperparameter erhalten und mit diesem Hyperparameter die Likelihood Funktion berechnen. Letztlich wählt man den Hyperparameter, welcher den besten Likelihood Wert aufweist.

Zufällige Initialisierung der Hyperparameter

Eine weitere Möglichkeit eine Initialisierung für die Hyperparameter zu finden ist, diese zufällig zu Erzeugen und die entsprechende Likelihood Funktion zu berechnen. Es würden die Hyperparameter gewählt, welche die beste Likelihood Funktion haben. Die zufällige Erzeugung ist extrem zeitaufwendig, da für jeden Satz zufälliger Hyperparameter die Likelihood Funktion ausgewertet werden muss. Um diesen Aufwand zu reduzieren, kann man einfach Stützstellen weglassen. Die Likelihood Funktion sollte sich im Vergleich zumindest ähnlich verhalten. Statt einer zufälligen Veränderung der Hyperparameter kann man auch ein Minimierungsverfahren mit reduzierter Stützstellenzahl verwenden. Dies wurde im Code auch umgesetzt. Die möglichen Minimierungsverfahren sind dieselben wie sie für das eigentliche Training verwendet werden und werden im nächsten Abschnitt beschrieben.

Reduziert man die Anzahl der Stützstellen wird die Korrelationsmatrix dementsprechend kleiner. Dadurch sinkt der Aufwand für die Invertierung und die Matrix Multiplikationen erheblich. Tests zeigten, dass die Initialisierung durch solch ein Verfahren zwar langsamer ist, allerdings konvergiert das Minimierungsverfahren durch die bessere Initialisierung deutlich schneller. Da bei dem Minimierungsverfahren wieder die volle Anzahl der Stützstellen notwendig ist und zusätzlich noch die Ableitungen der Korrelationsmatrix berechnet werden müssen, bietet dieses Verfahren durch die bessere Initialisierung für das gesamte Training betrachtet eine deutliche Beschleunigung.

Allerdings konnte auch beobachtet werden, dass die Initialisierung häufiger zu lokalen Minima führt. Eine Begründung für dieses Verhalten wurde noch nicht gefunden und sollte weitergehend untersucht werden.

Ein geeignetes Initialisierungverfahren für Ordinary-, Gradient Enhanced- und CO-Kriging zu finden ist keine leichte Aufgabe, dennoch kann eine Initialisierung das Trainingsergebnis enorm beeinflussen. Dies ist insbesondere von großer Bedeutung bei der Verwendung von gradientenbasierten Trainingsverfahren wie dem Quasi-Newton. Das CO-Kriging stellt hier die größte Herausforderung dar, da bei diesem Verfahren nicht nur die Hyperparameter der Korrelationsfunktionen bestimmt werden müssen,

sondern auch die Prozessvarianz für jede Fidelity des Krigingmodells und die Diagonalaufschläge ebenfalls für jede Fidelity.

Eine zufällige Initialisierung der Hyperparameter ist sicherlich ein gutes Verfahren zur Initialisierung. Der Vorteil bei diesem Verfahren liegt darin, dass während einer Optimierung Variation in die Modelle gelangt. Dies ist wünschenswert, daes bei einem konstanten Initialisierungsverfahren durchaus passieren kann, dass man durchgängig schlechte Modelle hat. Dies passiert insbesondere bei hochdimensionalen Parameterräumen. Für eine zufällige Initialisierung müssen allerdings Grenzen für die Hyperparameter gewählt werden, um den Suchraum zu verkleinern.

Für die minimale Grenze der Hyperparameter eignet sich das in Kapitel 5.3.2 vorgestellte Verfahren, für den Erwartungswert der Korrelationen sollte man hier einen hohen Wert wählen, bspw. 0.99

$$\log\left(-\frac{\log(0.99)}{k}\right) = \theta_{min} \tag{5.7}$$

Die maximale Grenze bedarf einer kleinen Änderung, da man in diesem Fall wissen möchte, wie groß ein dominierendes θ ist. Nimmt man einen geringen Erwartungswert für die Korrelation an von z.B. 0.01, so wäre der Schätzwert für ein dominierendes θ größer, als wenn man annimmt, alle θ seien gleich. Die Formel ändert sich dadurch zu:

$$\log(-\log(0.01)) = \theta_{max} \tag{5.8}$$

Genetisch/Zufällige Initialisierung der Hyperparameter

Random 2 beschreiben

Reduktion der zu bestimmenden Parameter auf 5 beim CO-Kriging 2 Thetas, 2 Varianzen und 1 Scale Faktor

Wiederverwendung des besten bisher gefundenen und normalverteilt um diesen varieren

Birgt die Gefahr in ein lokales Minimum zu kommen, durch die Zufälligkeit ist diese Gefahr allerdings eher klein

Initialisierung auf Basis bereits vorhandener Kriging Modelle

Das Kriging Modell wird in der Regel innerhalb einer Optimierung verwendet. In der Regel wird mit jedem neuen konvergierten Member ein neues Training gestartet und so die Hyperparameter neu bestimmt. Grundsätzlich wäre es natürlich äußerst sinnvoll die Hyperparameter aus den letzten trainierten Modellen zur Initialisierung zu verwenden. Als einfacher Ansatz wäre es z.B. möglich einfach immer die Hyperparameter aus dem letzten Training zu Initialisierung zu verwenden. In folgenden Fällen, kann dies allerdings zu Problemen führen:

1. Das letzte Modell befindet sich in einem lokalen Minimum der Likelihood Funktion
2. Die Hyperparameter der Kovarianz Funktion(en) sind noch nicht richtig eingestellt

Im ersten Fall besteht die Gefahr, dass das Training durch die ungünstige Initialisierung im lokalen Minimum bleibt und so nicht die optimalen Hyperparameter findet. Das wiederum führt zu schlechten Vorhersagen. Im Extremfall kann es sogar passieren, dass das lokale Minimum während der gesamten Optimierung nicht mehr verlassen wird.

Der zweite Fall ist insbesondere am Anfang der Optimierung interessant, denn am Anfang hat man in der Regel nur wenig Samples zur Verfügung und damit ist es dem Kriging Training noch nicht möglich die richtigen Hyperparameter für die Kovarianzfunktion zu schätzen. Diese Fälle treten erfahrungsgemäß leider sehr häufig auf, deshalb sollte man auf diese Art der Initialisierung verzichten.

Eine rein zufällige Initialisierung hat allerdings den Nachteil, dass die Trainingszeit enorm steigt und die Modelle im Laufe der Optimierung sehr unterschiedlich ausfallen können. Eine andere Möglichkeit der Initialisierung ist eine Mischform zwischen zufälliger Initialisierung und der Verwendung alter Modelle. In diesem Fall soll ein Kriterium darüber entscheiden, ob ein altes Kriging Modell verwendet werden soll oder eine zufällige Initialisierung durchgeführt werden soll. Zudem ist es sinnvoll nicht nur das letzte Kriging Modell zu betrachten, sondern noch weitere Modelle die während der Optimierung entstanden sind. Dies macht insbesondere Sinn, da das Ausprobieren eines vorhandenen Hyperparameter Satz im Vergleich zum Training nur einen Bruchteil der Zeit benötigt und man so einzelne “Ausreißer” in den Modellen nicht den weiteren Optimierungsverlauf gefährden. Der in AutoOpti verwendete Algorithmus sieht wie folgt aus:

```
1 bestKrigingFile = None
2 vector<string> krigingFiles = getLastKrigingFiles(20);
3 if(krigingFiles.size() < 20){
4     initType = random;
```

```
5       end()
6   }
7
8
9   for(i=0; i < krigingFiles.size() ; i++){
10      oldLikelihood = getLikelihood(krigingFiles[i])
11      newLikelihood = calculateLikelihood(krigingFiles[i])
12      if( (newLikelihood < bestLikelihood)
13      and (newLikelihood < oldLikelihood)
14      and (newLikelihood < -numberSamples/4.0) ){
15              bestLikelihood =newLikelihood
16              bestKrigingFile = krigingFiles[i]
17      }
18  }
19
20  if(bestKrigingFile==None)
21      initType = random;
```

Der Algorithmus startet mit dem Speichern der Dateinamen der letzten 20 Kriging Modelle aus der laufenden Optimierung (Zeile 2). Sind noch keine 20 Kriging Modelle erzeugt worden, soll die Initialisierung zufällig erfolgen (Zeile 3-6). In der darauffolgenden for Schleife erfolgt nun die Bewertung der einzelnen Kriging Modelle. Für die Bewertung muss zuerst der alte Likelihood Wert ausgelesen werden, dies geschieht in Zeile 10. Im nächsten Schritt muss der Likelihood mit der aktuellen Datenbasis neu berechnet werden, in der Regel sind an dieser Stelle einige Member zur Datenbasis hinzugekommen. Dieser Schritt ist numerisch auch der aufwendigste, allerdings muss keine Invertierung $\mathcal{O}(\mathrm{n}^3)$ durchgeführt werden, sondern jeweils nur ein Gleichungssystem gelöst werden $\mathcal{O}(\mathrm{n}^2)$. In der darauffolgenden If Abfrage geht es zum einen darum das beste Modell der 20 eingelesenen Kriging Modelle zu finden. Hierfür wird einfach der kleinste Likelihood verwendet (siehe Kapitel 5.1.6). Zudem ist eine weitere Bedingung, dass der neu berechnete Likelihood kleiner sein muss, als der bereits eingelesene aus dem vorhergehenden Modell. Die Überlegung hierbei ist, dass wenn ein neues Sample eingefügt wird und dieses nicht in die angenommene Verteilung passt, die Hyperparameter vollständig neu eingestellt werden müssen. Im umgekehrten Fall, sollte der Likelihood kleiner werden, da dieser linear mit der Sample Anzahl sinkt.

Als letzte Bedingung ist eine absolute Grenze für den Likelihood Wert angegeben, diese basiert rein auf Erfahrungswerten und soll sicherstellen, dass grundsätzlich zu schlechte Modelle zufällig initialisiert werden. Dies ist meistens am Anfang einer Optimierung der Fall, wenn noch nicht genügend Daten vorhanden sind, um die Kovarianzfunktion ausreichend gut zu schätzen. In diesem Fall ist eine zufällige Initialisierung

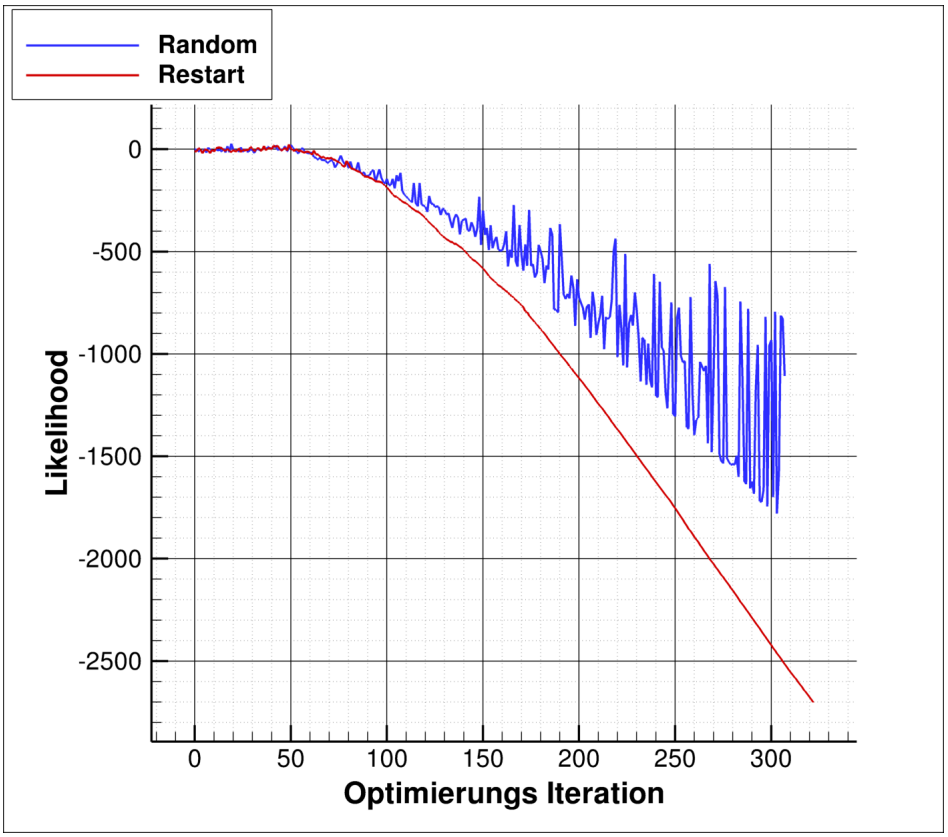


Abbildung 5.6: Vergleich verschiedener Initialisierungsverfahren und deren Auswirkung auf eine Testoptimierung

ebenfalls günstiger.

5.3.3 Minimierungsverfahren

Innerhalb des Kriging Modells wurden zwei verschiedene mehrdimensionale Minimierungsverfahren eingesetzt. Beide Verfahren waren bereits in einer institutseigenen Software Bibliothek verfügbar.

Minimierungsverfahren angelehnt an Resilient Backpropagation

Das erste hier verwendete Minimierungsverfahren ist angelehnt an ein Trainingsverfahren für Neuronale Netzwerke, genannt RPROP (Resilient Backpropagation) [Riedmiller & Braun, 1993, Helbig & Scherer, 2011] und ist ein Verfahren erster Ordnung. Besonderheit des Verfahrens ist, dass es nur das Vorzeichen der partiellen Ableitungen verwendet und nicht den Wert selbst.

Die Änderung der Hyperparameter θ_i für den nächsten Iterationsschritt $t+1$ ergibt sich aus der Schrittweite γ_i. Diese wird für jeden Hyperparameter einzeln bestimmt

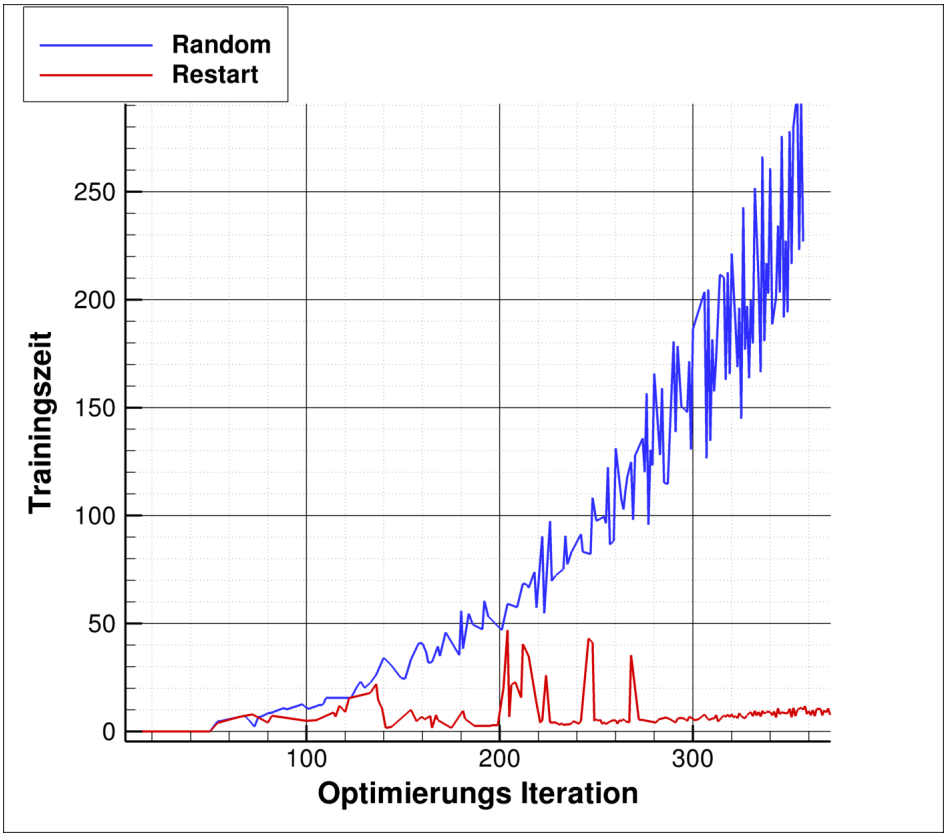


Abbildung 5.7: Vergleich verschiedener Initialisierungsverfahren und deren Auswirkung auf eine Testoptimierung

und in jeder Iteration geändert. Die Änderung hängt nur von dem Vorzeichen der entsprechenden partiellen Ableitung $\frac{\partial f}{\partial \theta_i}$ zum Zeitpunkt t der zu minimierenden Funktion f ab.

$$\theta_i^{t+1} = \theta_i^t - \gamma_i^t \text{sgn}\left(\left(\frac{\partial f}{\partial \theta_i}\right)^t\right)$$

Die Schrittweite wird in jedem Iterationsschritt für jeden Hyperparameter einzeln angepasst. Dies wird über zwei Multiplikatoren erzielt $\eta^+ = \{\eta^+ \in \mathbb{R} | 1 < \eta^+\}$ und $\eta^- = \{\eta^- \in \mathbb{R} | 1 > \eta^-\}$. Ist die entsprechende partielle Ableitung aus dem letzten Schritt multipliziert mit dem jetzigen Schritt größer als Null, wird die Schrittweite erhöht, indem die Schrittweite γ_i^t multipliziert wird mit η^+. Wenn die partielle Ableitung aus dem letzten Schritt multipliziert mit dem jetzigen Schritt kleiner als Null ist, dann wird die Schrittweite verkleinert durch Multiplikation mit η^-. Für die Schrittweite wird zudem eine Unter- und Obergrenze ($\gamma_{min}, \gamma_{max}$) festgelegt.

$$\gamma_i^{t+1} = \begin{cases} \min\left(\gamma_i^t \eta^+, \gamma_{max}\right) & wenn \left(\frac{\partial f}{\partial \theta_i}\right)^t \left(\frac{\partial f}{\partial \theta_i}\right)^{t-1} > 0 \\ \max\left(\gamma_i^t \eta^-, \gamma_{min}\right) & wenn \left(\frac{\partial f}{\partial \theta_i}\right)^t \left(\frac{\partial f}{\partial \theta_i}\right)^{t-1} < 0 \\ \gamma_i^t & sonst \end{cases}$$

Bei sehr flachen Bereichen der zu minimierenden Funktion, wo die partiellen Ableitungen nur sehr klein sind, würden andere Gradientenverfahren nur sehr langsam bis gar nicht mehr vorwärts kommen. Da dieses Verfahren allerdings die Größe der Gradienten überhaupt nicht berücksichtigt, kann dies nicht passieren. Das ist bei der Likelihood Funktion von besonderem Vorteil, da diese bereits durch Ihre Definition sehr viele flache Gebiete aufweist.

Verbessertes Minimierungsverfahren angelehnt an Resilient Backpropagation

Ein sehr großes Problem bei dem RPROP Verfahren ist, dass es relativ viele Iterationen benötigt bis es konvergiert. In jedem Iterationsschritt muss zum einen der Dichtefunktionswert der Likelihood Funktion berechnet werden und zum anderen die partiellen Ableitungen nach den Hyperparametern. Die Berechnung der Likelihood Funktion sowie die Berechnung einer partiellen Ableitung ist von der Komplexität $\mathcal{O}(n^2)$. Es müssen allerdings o partielle Ableitungen gebildet werden, aus diesem Grund kann der numerische Aufwand stark variieren.

Die Lernraten η^+, η^- sind im RPROP Verfahren konstant, insbesondere bei den anfänglichen Iterationsschritten führt dies zu einem relativ langsamen Anpassen der Deltas γ_i^t. Es wäre daher wünschenswert die Lernraten ebenfalls anzupassen. Eine gute Möglichkeit ist es verschiedene Lernraten einfach auszuprobieren. Der folgende Pseudo Programmcode zeigt die Umsetzung des neuen Verfahrens:

```
density = RPROPDensity(eta_plus, eta_minus)

// Teste kleinere und größere Lernraten für eta_plus
for(eta_plusFact=0.9; eta_plusFact<=1.1; eta_plusFact+=0.2){
        newEtaPlus = eta_plus*eta_plusFact

        if(newEtaPlus<1.2)
                                newEtaPlus=1.2
        if(newEtaPlus>2.0)
                                newEtaPlus=2.0
        if(eta_plus==newEtaPlus)
                                continue

        newDensity = RPROPDensity(newEtaPlus, eta_minus)
        if(newDensity<density)
                eta_plus=newEtaPlus
}

// Teste kleinere und größere Lernraten für eta_minus
for(eta_minusFact=0.9; eta_minusFact<=1.1; eta_minusFact+=0.2){
        newEtaMinus = eta_minus*eta_minusFact

        if(newEtaMinus<0.4)
                                newEtaMinus=0.4
        if(newEtaMinus>0.7)
                                newEtaMinus=0.7
        if(eta_minus==newEtaMinus)
                                continue

        newDensity = RPROPDensity(eta_plus, newEtaMinus)
        if(newDensity<density)
                eta_minus=newEtaMinus
}
```

In einem Iterationsschritt, wird dann zuerst der Dichtefunktionswert mit den bisherigen

Lernraten η^+, η^- berechnet.

Danach wird dann zuerst η^+ leicht erhöht oder verringert und überprüft, ob es im Bereich von $2.0 > \eta^+ > 1.2$ liegt (diese Werte sind reine Erfahrungswerte). Sollte sich das neue η^+ nicht geändert haben, so wird sich die Berechnung der Dichtefunktion gespart. Gewählt wird die Lernrate mit dem geringsten Dichtefunktionswert. Für die Lernrate η^- gilt im Prinzip dasselbe.

Für diese Art der Lernratenregelung sind maximal 4 neue Dichtefunktionsauswertungen notwendig, im Gegenzug hat man allerdings eine deutliche Verringerung der Iterationsanzahl und muss somit deutlich weniger partielle Ableitungen bestimmen. Diese sollte insbesondere für das CO-Kriging von großem Vorteil sein.

Um das Verfahren zu validieren, wurde eine Datenbasis aus einer aktuellen Optimierung für einen gegenläufigen Rotor verwendet (Referenz). Es gab ca. 113 freie Parameter und für das CO-Kriging somit 228 Hyperparameter (130*2 und 2x die Prozessvarianzen der Kovarianzfunktionen) und die Datenbasis enthielt zu diesem Zeitpunkt 373 Member. Das CO-Kriging wurde mit zufälligen Hyperparametern initialisiert und 10x mit dem RPROP Verfahren trainiert und 10x mit dem neuen RPROP2 Verfahren. Die Vorhersagen der fertig trainierten Ersatzmodelle wurden dann anhand einer Testdatenbasis validiert und ein mittlerer Vorhersagefehler bestimmt. Die folgenden Tabellen zeigen die Ergebnisse beider Verfahren:

RPROP	Mittelwert	Standardabweichung
Trainingszeit	346.3s	100.9s
Mittlerer Fehler	0.0201	0.00565
Trainingsiterationen	781	165

RPROP2	Mittelwert	Standardabweichung
Trainingszeit	186s	50.29s
Mittlerer Fehler	0.0149	0.0031
Trainingsiterationen	398	86

Man kann sehen, dass das RPROP Verfahren in diesem Beispiel die 1.86 fache Zeit benötigt um Konvergenz zu erreichen. Dies wird hauptsächlich durch die deutlich geringere Iterationsanzahl erreicht. Die mittleren Fehler sind in etwa vergleichbar, die etwas geringeren Fehler beim RPROP2 sind mit hoher Wahrscheinlichkeit zufälliger Natur.

Die folgende Abbildung zeigt nochmal den gemittelten Trainingsverlauf beider Verfahren. Die rote und schwarze Kurve stellt den mittleren Dichtefunktionswert über den Iterationsschritten dar. Die Fehlerbalken sind die Standardabweichungen der verschiedenen Trainings. Auch hier lässt sich gut erkennen, dass das RPROP2 Verfahren eine deutlich schnelleren Konvergenzverlauf hat, insbesondere am Anfang. Dies wird durch

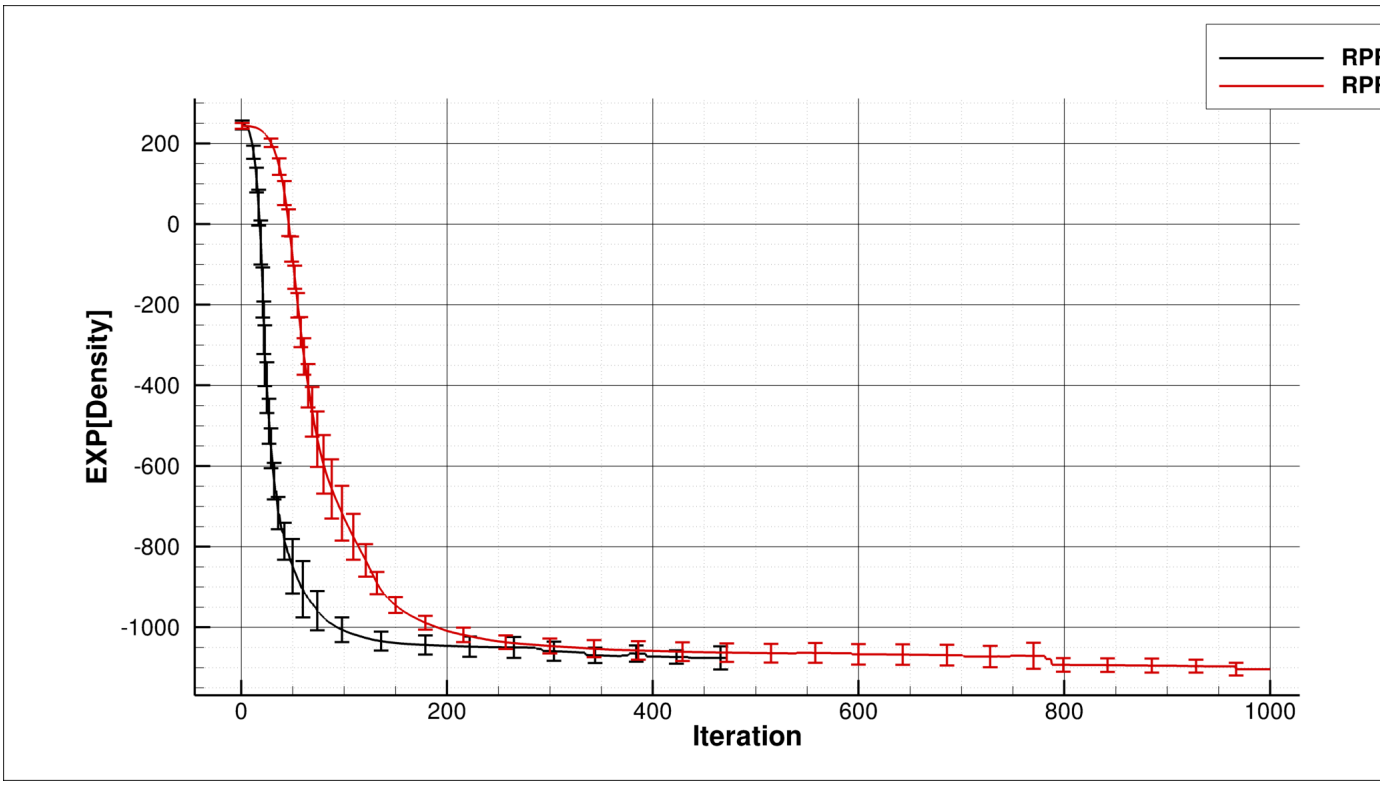


Abbildung 5.8:

die schneller eingestellten Deltas erreicht.

Natürlich bleibt zu beachten, dass das Verfahren bei einer sehr geringen Anzahl von Hyperparametern auch durchaus langsamer sein kann.

Quasi Newton

Das zweite implementierte Minimierungsverfahren ist ein Verfahren höherer Ordnung namens Quasi Newton. Basis für diese Art der mehrdimensionalen Minimierung ist eine Taylor Approximation zweiten Grades, wobei t der Iterationsschritt ist und $\mathbf{H}$ die Hesse Matrix:

$$f\left(\vec{\theta}\right) \approx f\left(\vec{\theta}_t\right) + \left(\vec{\theta} - \vec{\theta}_t\right)^T \nabla f\left(\vec{\theta}_k\right) + \frac{1}{2}\left(\vec{\theta} - \vec{\theta}_t\right)^T \mathbf{H}\left(\vec{\theta}_t\right)\left(\vec{\theta} - \vec{\theta}_t\right)$$

Die entsprechende Ableitung dieser Funktion muss im Minimum oder Maximum der Funktion Null ergeben:

$$\nabla f\left(\vec{\theta}\right) \approx \nabla f\left(\vec{\theta}_t\right) + \mathbf{H}\left(\vec{\theta}_t\right)\left(\vec{\theta} - \vec{\theta}_t\right) = 0$$

Besonderheit bei der Quasi Newton Methode ist, dass die Hesse Matrix $\mathbf{H}$, nicht direkt berechnet werden muss, sondern sukzessive über die Gradienten angenähert wird. Vorteil des Verfahrens ist, dass es deutlich schneller konvergiert als das bereits vorgestellte Verfahren erster Ordnung. Allerdings ist es weniger robust und kann in flachen Gebieten der Funktion langsam bis gar nicht konvergieren. Die exakte Umsetzung des Algorithmus und weitere Details können in [**?**, Gill et al., 1981, Gill, 2007] gefunden werden.

5.3.4 Softwaretechnische Umsetzung

Klasse für die Steuerung des Trainings

In diesem Abschnitt soll der Ablauf und die dazugehörige Klasse für ein Training eines Kriging Modells erklärt werden. Das UML Diagramm 5.9 zeigt die Klasse Trainer, welche das Training steuern und verwalten soll.

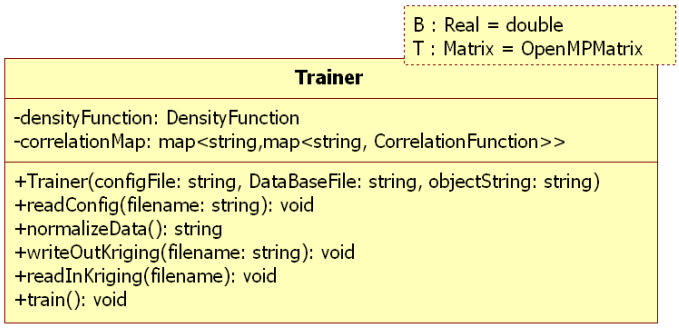


Abbildung 5.9: UML Diagramm der Trainer Klasse, welche das Training des Kriging Modells steuert

Die Methoden und auch Attribute der Trainer Klassen sollen am Ablauf des Trainings erläutert werden. Ein Training besteht im Wesentlichen aus den folgenden Schritten:

1. Der erste Schritt besteht aus der Erzeugung eines Trainer Objekts. Dem Konstruktor müssen drei Parameter übergeben werden: Der Name der Konfigurationsdatei (configFile), der Name der Datenbankdatei (DataBaseFile) und die zu trainierende Funktion. Der Parameter objectString gibt an, welcher der Funktionswerte aus der Datenbankdatei verwendet werden soll, diese kann zu einem Variablensatz mehrere verschiedene Funktionen beinhalten. Innerhalb des Konstruktors werden dann einige Schritte ausgeführt, um das Kriging Modell zu initialisieren.

   (a) Einlesen der Datenbankdatei.

(b) Einlesen der Konfigurationsdatei durch die Methode readConfig(), an dieser Stelle werden auch die zu verwendenden Korrelationsfunktionen gesetzt (Attribut correlationMap, siehe Kapitel 5.1.3). Zudem wird ein DensityFunktion Objekt erzeugt und in dem Attribut densityFunction gespeichert, siehe Kapitel **??**.

(c) Normalisierung der Stützstellen und der dazugehörigen Funktionswerte (mit der Methode normalizeData())

(d) Initialisierung der Hyperparameter, siehe Kapitel 5.3.2.

2. Starten der train() Methode des Trainer Objekts.

   (a) Erzeugung eines Minimierer Objekts, je nach gewähltem Minimierer Typ. Die entsprechende Klassenstruktur wird im nächsten Abschnitt behandelt

   (b) Starten des Minimierers

3. Nach erfolgreichem Training wird eine XML Datei geschrieben, in der im Wesentlichen alle Ergebnisse des Trainings stehen. Es werden die gefundenen Hyperparameter, die Korrelationsmatrix, einige Vektoren usw. gespeichert um bei einer späteren Vorhersage diese Werte nicht mehr berechnen zu müssen. Diese XML Datei beinhaltet also ein fertiges Kriging Modell.

Klassenstruktur zur Steuerung der Minimierungsverfahren

In Abbildung 5.10 wird das UML Diagramm der Klassenstruktur für die Minimierungsverfahren gezeigt. Es gibt eine abstrakte Superklasse MinimizerInterface, welche hier als Interface zu verstehen ist. Diese schreibt die notwendigen Methoden für die Subklassen vor. Die Subklassen sollen dann die konkreten Minimierungsverfahren realisieren. Die einzige öffentliche Methode callMinimizer ist dazu da, um von außen den entsprechenden Minimierer aufzurufen und die Minimierung zu starten. In der Methode function muss die zu minimierende Funktion berechnet werden. Die Rückgabe des berechneten Funktionswertes wird über eine Referenz des Parameters functionValue gemacht, eine Referenz wird aus Performancegründen verwendet. Der Parameter variables vom Typ vector, soll die entsprechenden Variablen beinhalten. Zu diesen Variablen wird dann der Funktionswert berechnet. Wie die Berechnung vor sich geht und was genau berechnet wird, ist den einzelnen Subklassen überlassen, diese müssen sich nur an die Interface Spezifikation halten.

Die Methode functionDerivative soll den Gradienten des Funktionswertes abgeleitet nach den Variablen berechnen. Der Gradient wird über die Referenz auf den Parameter derivatives zurückgegeben. Zusätzlich soll es möglich sein, Nebenbedingungen

für die Minimierung vorzugeben. Dies wird über die Methode constraintFunction umgesetzt. Die Nebenbedingung muss so formuliert werden, dass diese bei einem Wert größer oder gleich Null eingehalten wird und unter Null nicht eingehalten wird. Zudem muss der Gradient der einzelnen Nebenbedingungsfunktionen bereitgestellt werden und zwar über die Methode constraintFunctionDerivative.

Um die Konvergenz des Verfahrens festzustellen, wird die Methode convergenceCheck verwendet. Der Methode müssen drei Parameter übergeben werden, der erste Parameter ist ein vector mit den Funktionswerten der bisher durchgeführten Iterationen. Der zweite Parameter ist ein mehrdimensionaler vector, welcher alle Variablenwerte der bisherigen Iterationen beinhaltet und der letzte Parameter der Methode ist die Nummer der aktuellen Iteration. Das entsprechende Konvergenzkriterium muss dann innerhalb der Funktion umgesetzt werden. Beispielsweise könnte man das Verfahren als konvergiert ansehen, wenn die Funktions- und Parameterwerte sich seit einigen Iterationen nicht mehr verändert haben oder die Veränderung unterhalb einer bestimmten Schwelle liegt.

Die Methode saveFunction stellt eine Art von Rettungsfunktion dar. Diese soll aufgerufen werden, wenn das Verfahren in irgendeiner Form numerisch instabil wird. Beispielsweise könnte eine einfache Maßnahme sein, die Funktionswerte zufällig zu verändern, in der Hoffnung auf ein anderes (globaleres) Minimum zu treffen.

Die jeweiligen Subklassen implementieren in diesem Fall noch ein Attribut vom Typ DensityFunction, da diese die LikeliHood Funktion minimieren sollen und die Klasse DensityFunction alle nötigen Methoden für die Berechnung dieser liefert, siehe Kapitel 5.1.6.

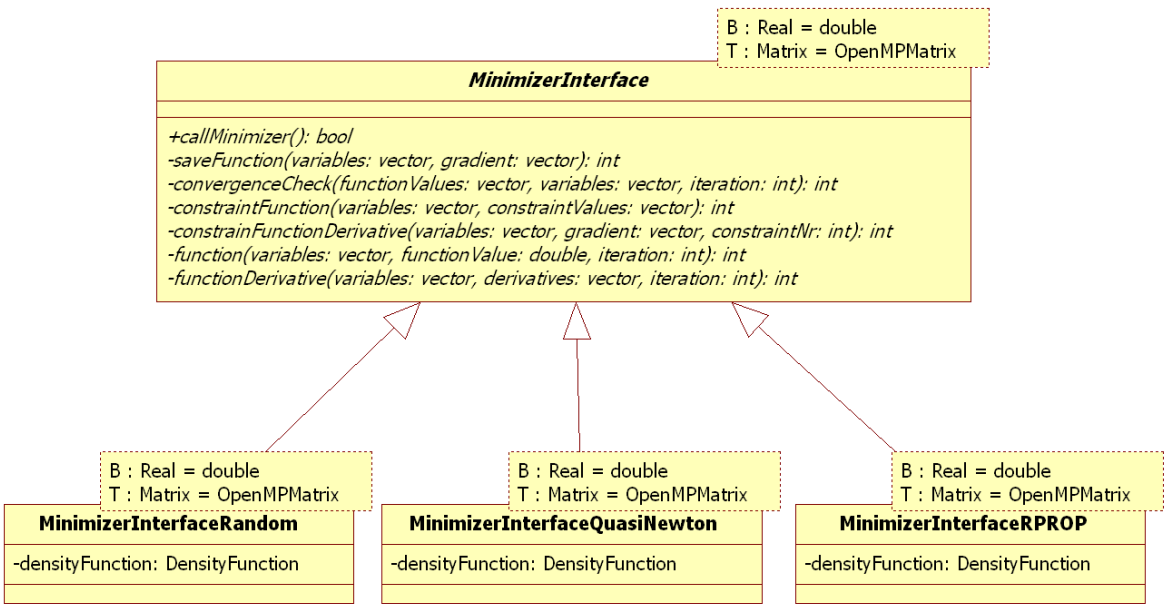


Abbildung 5.10: UML Diagramm der Klassenstruktur der Minimierungsalgorithmen

Das folgende Listing zeigt die Umsetzung der Methode function in der Subklasse MinimizerInterfaceRPROP. Die Methode ist in diesem Fall vereinfacht dargstellt, einige Ausgabefunktionen wurden aus Platzgründen entfernt. Die wichtigen Teile der Methode sind allerdings unverändert. Der Methodenkopf entspricht dem aus dem UML Diagramm. In Zeile 5 werden die aktuellen Variablen (in diesem Fall die Hyperparameter) dem Objekt densityFunction zu Berechnung der Likelihood Funktion übergeben. In den Zeilen 7-18 wird dann die eigentliche Likelihood Funktion innerhalb eines try-catch Blocks berechnet, um eventuelle Exceptions fangen zu können. Wird eine Exception geworfen, so wird eine Fehlermeldung ausgegeben und eine -1 als zurückgegeben. Zusätzlich kann wie in Zeile 15 die entsprechende Korrelationsmatrix bei Auftreten einer Exceptions ausgegeben werden, in diesem Fall wird die Matrix in eine Datei geschrieben.

```
1  int MinimizerInterfaceRPROP<T,B>::function( vector<double> &hyperparameter,
2                                              double &likelihood,
3                                              size_t *iteration){
4
5      densityFunction->setAllHyperparameter(hyperparameter);
6
7      try{
8          likelihood = densityFunction->calcDensity();
9      }
10     catch(InvCholNotIdentity &e){
11         cout <<"RPROP Func (InvCholNotIdentity):"<<endl;
12         return -1;
13     }
14     catch(ChodecNotPosDef &Exception){
15         densityFunction->getCorrelationMatrixRef().saveAsAscii("CorrMatFailed_sav");
16         cout <<"RPROP Func (ChodecNotPosDef):"<<endl;
17         return -1;
18     }
19
20 }
```

Ein zusätzliches Problem bei der Implementierung der Minimierungsverfahren war, dass diese in einer externen Bibliothek in Form von C Funktionen vorlagen und diese Funktionen Funktionspointer als Parameter erwarten. Die zu übergebenden Funktionen entsprechen den Funktionen aus der MinimizerInterface Klasse, also z.B. function oder functionDerivative.

```
quasiNewton(nrHyperparam,
            nrConstraints,
            variables,
            function(),
            functionDerivatives(),
            constraintFunction(),
            constrainFunctionDerivative(),
            convergenceCheck())

rprop(      nrHyperparam,
            nrConstraints,
            variables,
            variablesLowerLimit,
```

```
variablesUpperLimit,
function(),
functionDerivatives(),
constraintFunction(),
constrainFunctionDerivative(),
convergenceCheck())
```

Die Schwierigkeit ergibt sich in diesem Fall dadurch, dass die Funktionspointer in C++ eine Zuordnung zu dem entsprechenden Objekt zu dem die Funktionen gehören, benötigen. Das folgende Listing soll das Problem verdeutlichen:

```
1 MinimizerInterfaceRPROP testObject;
2 int (MinimizerInterfaceRPROP::*ptr2)(vector<double> &, double &, size_t *) =
3                                          &MinimizerInterfaceRPROP::function;
4 (testObject.*ptr2)(variables, functionValue, iteration);
```

In Zeile 1 wird ein Test Objekt vom Typ MinimizerInterfaceRPROP erzeugt (der Konstruktoraufruf wurde hier absichtlich vereinfacht). In den Zeilen 2-3 wird ein Funktionspointer namens ptr2 erzeugt, dieser zeigt auf eine Methode der Klasse MinimizerInterfaceRPROP mit den entprechenden Parametern der Methode function. Zusätzlich erfolgt in diesen Zeilen eine Zuweisung des Pointers der Methode durch *"&MinimizerInterfaceRPROP::function*".

In Zeile 4 wird ein beispielhafter Aufruf des Funktionspointers auf dem Objekt testObject gemacht. Dieses Beispiel würde so funktionieren. Das eigentliche Problem besteht aber nun darin, dass man in Zeile 2 statt der Subklasse MinimizerInterfaceRPROP die abstrakte Klasse MinimizerInterface verwenden möchte. Da diese Funktionspointer Parameter einer Funktion darstellen, wäre dieses Verhalten sehr wichtig, weil so alle Methodenpointer der Subtypen von MinimizerInterface angenommen werden würden. Andernfalls müsste man die Funktion rprop oder quasiNewton für jeden Subtypen von MinimizerInterface neu implementieren. Leider sind die Möglichkeiten polymorpher Programmierung in C++ stark begrenzt und solch ein Konstrukt wird von der Sprache nicht unterstützt.

Um dieses Problem zu umgehen, werden Funktionsobjekte [Douglas, 2004] der Boost Bibliothek verwendet. Mit dieser Bibliothek ist es möglich, die entsprechenden Methoden als Objekt an die entsprechenden externen Funktionen (z.B. rprop und quasiNewton) zu übergeben. Das folgende Listing soll die prinzipielle Funktionsweise von Boost Funktionsobjekten erklären:

```
1  class X {
2  public:
3   int foo(int);
4  };
5
6  boost::function<int (X*, int)> f;
7
8  X x;
9
10 f = &X::foo;
11 f(&x, 10);
```

In diesem Beispiel soll ein Funktionsobjekt der Methode foo der Klasse X erzeugt werden. Zu diesem Zweck wird ein Funktionsobjekt in Zeile 6 initialisiert, wobei innerhalb der eckigen Klammern zuerst der Rückgabewert int und danach die Parameter der Funktion (X*, int) übergeben werden. Der Parameter X* muss vorhanden sein, da innerhalb C++ der erste Parameter einer Methode immer das Objekt selbst ist. Im Normalfall wird dies jedoch automatisch umgesetzt und ist daher unsichtbar für den Programmierer. Die Zuweisung der Methode auf das Funktionsobjekt erfolgt dann in Zeile 10. Der Aufruf des Funktionsobjekts erfolgt nach normaler C++ Syntax, siehe Zeile 11.

Für die Klasse MinimizerInterfaceRPROP würde ein solches Funktionsobjekt wie folgt aussehen:

```
boost::function<int (MinimizerInterfaceRPROP*, vector<double> &, double &, size_t *)> fPointer;
```

Das Problem dass das Funktionsobjekt unabhängig vom Subtyp der Klasse MinimizerInterface sein soll, bleibt allerdings bestehen. Um dies nun zu umgehen, kann man boost::bind verwenden. Mit dieser Funktionalität ist es möglich, Parameter von Funktionen zu verändern.

Das nächste Listing soll dies verdeutlichen, es handelt sich hier um ein stark vereinfachtes Beispiel um die grundlegende Funktionalität zu erklären.

```
1  void external_rprop(boost::function<int (vector<double> &, double &, size_t *)> fPointer){...}
2
3  class MinimizerInterfaceRPROP: public MinimizerInterface{
4  public:
5      void callMinimizer(){
6          boost::function<int (vector<double> &, double &, size_t *)> fPointer;
7          fPointer = boost::bind(&MinimizerInterfaceRPROP::function, (*this), _1, _2, _3);
8          external_rprop(fPointer);
9      }
10 private:
11     int function(vector<double> &vars, double &f, size_t *it) {...}
12 }
```

In Zeile 1 ist eine Funktion definiert, welche einer externen Bibliotheksfunktion entspricht, beispielsweise einem externen RPROP Algorithmus. Dieser Algorithmus benötigt nun ein Funktionsobjekt, mit dem er die zu minimierende Funktion berechnen kann. Wie man sehen kann, benötigt das hier definierte Funktionsobjekt als ersten Parameter nicht mehr das aufrufende Objekt selbst.

Analog zum Originalcode wird als nächstes ist die Subklasse MinimizerInterfaceRPROP definiert, welche Subklasse der abstrakten Klasse MinimizerInterface ist. Diese Klasse besitzt nun eine öffentliche Methode namens callMinimizer. Diese Methode soll von irgendeinem Clienten ausgeführt werden können, um den Minimierungsalgorithmus starten.

Der erste Schritt innerhalb der Methode callMinimizer ist die Erzeugung eines boost Funktionsobjekts, auch hier ist der erste Parameter nicht mehr das aufrufende Objekt selbst (also MinimizerInterfaceRPROP*). In Zeile 7, wird nun ein Funktionsobjekt mit boost bind erzeugt. Mit bind ist es Möglich, die Methodenparameter zu verändern. Dies wird dazu verwendet das Funktionsobjekt quasi unabhängig von der aufgerufenen Klasse zu machen. Dem ersten Parameter für bind wird der Funktionspointer übergeben, der zweite Parameter ist das Objekt selbst. Mit boost bind ist es nun möglich, das Objekt einfach standardmäßig über den this Zeiger fest zu binden. Dieser taucht im kreierten Funktionsobjekt nicht mehr auf und man hat die gewünschte Unabhängigkeit erreicht. Die nächsten drei Parameter _1,_2,_3 sind Platzhalter für die später nötigen Parameter des Funktionsobjekts (also vector<double> &, double &, size_t *).

In der nächsten Zeile wird das Funktionsobjekt an die externe Bibliotheksfunktion übergeben und diese kann die Funktion nun nach belieben verwenden.

5.3.5 Renormalisierung der Hyperparameter

Möchte man mit bestehenden Hyperparametern, aber einer neuen oder erweiterten Datenbasis Vorhersagen treffen, so ändern sich die Erwartungswerte der zu trainierenden Funktion und deren Parametern. Da das alte Training und damit auch die Daten des Trainings mit den alten Erwartungswerten und Standartabweichungen normalisiert worden sind, müssen die Hyperparameter ebenfalls renormalisiert werden. Ansonsten würde man für die Kovarianz zwischen 2 lMembern unterschiedliche Werte bekommen.

Daraus ergibt sich folgende notwendige Bedingung:

$$cov\left(\vec{x}_{1alt}, \vec{x}_{2alt}\right) = cov\left(\vec{x}_{1neu}, \vec{x}_{2neu}\right) \tag{5.9}$$

Wobei x_{real} den unnormierten Parameter darstellt, μ_{alt} und σ_{alt} stellen den alten Erwartungswert sowie die Standartabweichung der Parameter dar.

$$\vec{x}_{1alt} = \begin{bmatrix} \frac{x_{1,1real}-\mu_{1alt}}{\sigma_{1alt}} \\ \vdots \\ \frac{x_{1,nreal}-\mu_{nalt}}{\sigma_{nalt}} \end{bmatrix}$$

Für $\vec{x}_{2alt}$, sowies $\vec{x}_{1neu}$ gilt analoges.

Die Bedingung 5.9 soll anhand eines Beispiels erläutert werden, es wird hierfür eine Gauss Korrelationsfunktion mit einem Hyperparameter verwendet:

$$\sigma^2_{KriAlt} e^{-\frac{1}{2}e^{\theta_{alt}}\left|\frac{x_{1real}-\mu_{alt}}{\sigma_{alt}}-\frac{x_{2real}-\mu_{alt}}{\sigma_{alt}}\right|^2} = \sigma^2_{KriNeu} e^{-\frac{1}{2}e^{\theta_{neu}}\left|\frac{x_{1real}-\mu_{neu}}{\sigma_{neu}}-\frac{x_{2real}-\mu_{neu}}{\sigma_{neu}}\right|^2}$$

$$\sigma^2_{KriAlt} e^{-\frac{1}{2}e^{\theta_{alt}}\frac{1}{\sigma^2_{alt}}|x_{1real}-x_{2real}|^2} = \sigma^2_{KriNeu} e^{-\frac{1}{2}e^{\theta_{neu}}\frac{1}{\sigma^2_{neu}}|x_{1real}-x_{2real}|^2}$$

Geht man nun davon aus, dass die Kriging Varianz sich nicht ändert:

$$\sigma^2_{KriAlt} = \sigma^2_{KriNeu}$$

Wobei diese Bedingung im Code unbedingt erfüllt sein muss. Im Code ist es so umgesetzt, dass zuerst die Kovarianzmatrix mit der alten Krigingvarianz erzeugt wird und danach der Likelihood Schätzer für die Krigingvarianz aufgerufen wird, siehe 5.3.

Dieser Schätzer sollte im Normalfall allerdings auf eine neue Krigingvarianz kommen, wodurch es im weiteren Verlauf zu großen Problemen kommen kann. Insbesondere bei der Vorhersage, wo für den Kovarianzvektor $\vec{c}$ (siehe **??**) dann die neue Krigingvarianz verwendet werden würde. Kovarianzmatrix und Vektor würden dann nicht mehr zusammen passen.

Die Formel lässt sich damit weiterhin vereinfachen:

$$e^{-\frac{1}{2}e^{\theta_{alt}}\frac{1}{\sigma^2_{alt}}|x_{1real}-x_{2real}|^2} = e^{-\frac{1}{2}e^{\theta_{neu}}\frac{1}{\sigma^2_{neu}}|x_{1real}-x_{2real}|^2}$$

$$e^{\theta_{alt}}\frac{1}{\sigma^2_{alt}}|x_{1real}-x_{2real}|^2 = e^{\theta_{neu}}\frac{1}{\sigma^2_{neu}}|x_{1real}-x_{2real}|^2$$

$$e^{\theta_{alt}} \frac{1}{\sigma_{alt}^2} = e^{\theta_{neu}} \frac{1}{\sigma_{neu}^2}$$

$$e^{\theta_{alt}} \frac{\sigma_{neu}^2}{\sigma_{alt}^2} = e^{\theta_{neu}}$$

$$\log\left(e^{\theta_{alt}} \frac{\sigma_{neu}^2}{\sigma_{alt}^2}\right) = \theta_{neu}$$

$$\theta_{alt} + \log\left(\frac{\sigma_{neu}^2}{\sigma_{alt}^2}\right) = \theta_{neu}$$

5.3.6 Konvergenz von globalen und lokalen Hyperparametern im Bezug auf die Entscheidungsfunktionen

In diesem Abschnitt soll versucht werden das Konvergenzverhalten von lokalen und globalen Hyperparametern für das CO-Kriging während einer Optimierung einzuschätzen. Dafür sollen einige analytische wie auch reale Testfälle herangezogen werden. Eine Entscheidungsfunktion wie in Kapitel **??** beschrieben, kann nur vernünftig funktionieren, wenn die dafür verwendeten Ersatzmodelle bereits gute Schätzungen für die Hyperparameter besitzen. Globale Hyperparameter sind σ, a, β und lokale sind die θ. Grundlegend müssen für das CO-Kriging in der hier beschriebenen Form zwei verschiedene Parametersätze bestimmt werden. Einer für die Low-Fidelity Kovarianzfunktion und ein Satz für die Fehler- oder Differenz-Kovarianzfunktion.

Eine grundlegende Annahme ist, dass dem Ersatzmodell deutlich mehr LF Samples als HF Samples zur Verfügung stehen. Das bedeutet, dass die LF Kovarianzfunktion schneller bestimmt werden kann als die Differenzfunktion.

Eine These ist es, dass die globalen Parameter der Differenzfunktion allerdings schon sehr früh gut geschätzt werden können, die lokalen aber mehr Samples benötigen. Um den Optimierungsfortschritt voranzutreiben, ist aber die Abschätzung der globalen Parameter die wichtigere. Beispiele finden.

Dennoch kann eine Entscheidungsfunktion erst dann sinnvolle Entscheidungen treffen, wenn die lokalen und globalen Hyperparameter der LF Funktion gut geschätzt worden sind und zumindest die globalen Hyperparameter der Differenzfunktion.

Eine große Schwierigkeit besteht darin einschätzen zu können, wann welche Hyperparameter ausreichend gut eingeschätzt worden sind. Es gibt Arbeiten, die eine Mindestanzahl an

5.4 Algorithmische Effizienz steigern

5.4.1 Filtern von unwichtigen Samples

Insbesondere beim Gradient-Enhanced-Kriging sind sehr hohe Matrixgrößen von über 10000x10000 schnell erreicht. Aus diesem Grund ist es sinnvoll nur Samples auszusuchen, welche der Optimierung einen wirklichen Zugewinn bringen. Den Einfluss eines Samples auf den Optimierungsverlauf zu berechnen ist sehr schwierig, aus diesem Grund wäre es auch aktzeptabel die Samples herauszufiltern, welche nur einen kleinen Einfluss auf das Ersatzmodell haben. Eine einfach Möglichkeit wäre die Korrelationsmatrix selbst, in dieser stehen die Korrelationen zwischen allen Samples untereinander. Samples mit sehr hohen Korrelationen haben folglich nur einen sehr kleinen Abstand zueinander. Der Abstand wird allerdings über die im Kriging verwendeten Modell-Korrelationsfunktionen bestimmt und diese sind sehr stark abhängig von den verwendeten Hyperparametern. Die Hyperparameter werden vom Training allerdings erst bei ausreichender Sample Anzahl vernünftig geschätzt. Dies kann im schlimmsten Fall dazu führen, dass man Samples aufgrund einer falsch geschätzten Metrik entfernt und den Optimierungsverlauf so empfindlich stört.

....

....

Wenn man die Thetas als Wirkweite der jeweiligen freien Variable interpretiert und zusätzlich davon ausgeht, dass das Training die korrekten Thetas bereits gefunden hat. So könnte man mehr partielle Ableitungen von freien Variablen mit hoher Wirkweite entfernen. Dies würde in etwa dem Gedanken entsprechen, dass bei einer komplexen bspw. hochfrequenten Funktion mehr partielle Ableitungen benötigt werden als bei einer sehr glatten simpleren Funktion. Allerdings würde man hier die Komplexität dann auf Parameterebene messen. Eine Möglichkeit wäre es, die Parameter nach Thetas zu sortieren, wobei kleine Thetas oben stehen sollten. Dann die partiellen Ableitungen mit dem kleinsten Abstand in der freien Variable aussortieren und jeweils einige entfernen.

Annahmen:

- Thetas bereits gut geschätzt
- Restart bereits im Gange

- Mit dem besten Restart Modell dann anfangen und folgende Schritte durchführen

Die niedrigsten Thetas raussortieren. Beim niedrigesten anfangen und dort die partiellen Ableitungen herauswerfen, welche den kürzesten Abstand in dieser freien Variable haben.

Dann das nächste niedrigste Theta und dort die partiellen Ableitungen rauswerfen. Dies sollte für ein stabileres Training sorgen ohne wichtige Informationen zu verlieren.

....

5.4.2 Inverse durch Gleichungssysteme ersetzen

Mit Hilfe der Cholesky Zerlegung können lineare Gleichungssysteme sehr effizient gelöst werden. Dies kann man sich zunutze machen, um bei der Likelihood Berechnung auf die Bestimmung der Inversen verzichten. Die Methodik ist weit verbreitet und soll in dieser Arbeit daher nur kurz erläutert werden.

Der Likelihoodterm (siehe **??**) sieht wie folgt aus:

$$\log(N) = -\log\left(\det(\mathbf{Cov})\right) - \left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)^T \mathbf{Cov}^{-1} \left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)$$

Die Determinante der Kovarianzmatrix wird aus der Cholesky Zerlegung gewonnen (siehe 5.5.1). Der quadratische Term $\left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)^T \mathbf{Cov}^{-1} \left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)$ beinhaltet allerdings noch die Inverse Kovarianzmatrix. Dieser kann mit Hilfe der Cholesky Zerlegung gewonnen werden, wir führen hierfür einen Hilfsvektor $\vec{e}$ ein:

$$\left(\vec{y}_s - \beta_1 \overrightarrow{F}\right) = \vec{e}$$

Zusätzlich noch einen Hilfsvektor $\vec{d}$

$$\mathbf{Cov}^{-1}\vec{e} = \vec{d}$$

Bei der Cholesky Zerlegung wird die Matrix $\mathbf{Cov}$ in ein Produkt aus einer unteren Dreicksmatrix und deren Transponierten zerlegt, die Dreiecksmatrix $\mathbf{L}$ gilt an dieser Stelle als bekannt:

$$\mathbf{L}\mathbf{L}^T = \mathbf{Cov}$$

Daraus folgt:

$$\left(\mathbf{L}\mathbf{L}^{\mathbf{T}}\right)^{-1}\vec{e} = \vec{d}$$

$$\vec{e} = \vec{d}\mathbf{L}\mathbf{L}^{\mathbf{T}}$$

Führt man nun folgende Substitution ein:

$$\vec{d_{tmp}} = \vec{d}\mathbf{L}$$

$$\vec{e} = \vec{d_{tmp}}\mathbf{L}^{\mathbf{T}}$$

So kann dieses Gleichungssystem durch eine einfache Rückwärtssubstitution $\vec{d}_{tmp}$gelöst werden. Danach kann direkt das folgende Gleichungssystem durch Vorwärtseinsetzen gelöst werden und der Vektor $\vec{d}$ ist hiermit bekannt.

$$\vec{d_{tmp}} = \vec{d}\mathbf{L}$$

Der Aufwand hierfür ist deutlich geringer als bei der Invertierung, da nur zwei Gleichungssysteme gelöst werden müssen anstatt n Gleichungssysteme für die gesamte Invertierung.

5.4.3 Vollständiger Verzicht auf die Inverse durch Likelihood Partielle Ableitungen durch Approximation der Spur

Bei der Bestimmung der partiellen Ableitungen nach den Hyperparametern des Likelihood Terms ist es deutlich schwieriger auf die Invertierung zu verzichten. Dies liegt an der Bestimmung der Ableitung der Determinante nach den Hyperparametern.

$$\frac{\partial L\left(\vec{h}\right)}{\partial h_l} = -\frac{\partial}{\partial h_l}\left(\log\left(\det(\mathbf{Cov})\right)\right) - \frac{\partial}{\partial h_l}\left(\left(\vec{y}_s - \beta\overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\left(\vec{y}_s - \beta\overrightarrow{F}\right)\right)$$

$$\frac{\partial L\left(\vec{h}\right)}{\partial h_l} = -\frac{1}{\det(\mathbf{Cov})}\frac{\partial}{\partial h_l}\left(\det(\mathbf{Cov})\right) + \left(\left(\vec{y}_s - \beta_1\overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\frac{\partial \mathbf{Cov}}{\partial h_l}\mathbf{Cov}^{-1}\left(\vec{y}_s - \beta_1\overrightarrow{F}\right)\right)$$

$$\frac{\partial L\left(\vec{h}\right)}{\partial h_l} = -\operatorname{Spur}\left(\mathbf{Cov}^{-1}\frac{\partial \mathbf{Cov}}{\partial h_l}\right) + \left(\left(\vec{y}_s - \beta_1\overrightarrow{F}\right)^T \mathbf{Cov}^{-1}\frac{\partial \mathbf{Cov}}{\partial h_l}\mathbf{Cov}^{-1}\left(\vec{y}_s - \beta_1\overrightarrow{F}\right)\right)$$

Spur schätzen [Avron & Toledo, 2011, Hutchinson, 1989, Mark Gibbs, 1997]

$$Spur\left(R\right) = E\left[\vec{d}^{T} R \vec{d}\right]$$

$$Spur\left(R\right) \approx \frac{1}{N} \sum \vec{d}^{T} R \vec{d}$$

$$\vec{d} = \begin{bmatrix} N\left(0,1\right) \\ \vdots \\ N\left(0,1\right) \end{bmatrix}$$

Die Approximation benötigt leider einen sehr großen Zufallsvektor $\vec{d}$ um eine ausreichende Genauigkeit zu erhalten. Dies macht die Methode letztlich wieder ineffizient. Zudem bleibt immer eine Restunsicherheit in den partiellen Ableitungen die sich sehr negativ auf das Training auswirken kann.

5.4.4 Vollständiger Verzicht auf die Inverse durch Rückwärtsdifferentiation der Cholesky Zerlegung

Wie in Kapitel 5.4.3 bereits erwähnt, ist es bei der Berechnung des Likelihood Terms möglich auf die Invertierung der Kovarianzmatrix zu verzichten. Bei der Bestimmung der partiellen Ableitungen des Likelihood Terms nach den Hyperparametern ist dies allerdings schwieriger. Die Bestimmung der partiellen Ableitungen folgt dem folgenden Berechnungsschema:

1. Über alle benötigen Hyperparameter h_l
   (a) Bestimmung der Ableitung der Kovarianzmatrix $\frac{\partial \mathbf{Cov}}{\partial h_l}$
   (b) Berechnung der Ableitung der quadratischen Form: $\left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)^T \mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l} \mathbf{Cov}^{-1} \left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)$
   (c) Berechnung der Ableitung der Determinante: $\mathrm{Spur}\left(\mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l}\right)$

Punkt a bedeutet vom Aufwand die Aufstellung der symmetrischen Matrix $\frac{\partial \mathbf{Cov}}{\partial h_l}$. Die Komplexität des Algorithmus liegt bei $\mathcal{O}\left(n^2\right)$ und kann zudem sehr gut parallelisiert werden. Die Bestimmung der einzelnen Ableitungen der Kovarianzmatrix hängt stark von dem verwendeten Kriging Modell und der verwendeten Korrelationsfunktion ab. Beim Gradient Enhanced Kriging können diese Einzelableitungen komplexer werden und damit auch vom numerischen Aufwand teurer. Dennoch können in der Regel sehr

viele Teile aus der Aufstellung der Kovarianzmatrix wiederverwendet werden, was den Aufwand erheblich reduziert und daher eher unerheblich macht.

Punkt b ist vom Aufwand her nahezu vernachlässigbar. In der Regel wurde der Vektor $\left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)^T \mathbf{Cov}^{-1}$ bereits in der Likelihood Berechnung bestimmt und es muss nur noch eine Vektor Matrix Multiplikation durchgeführt werden.

Punkt c ist der aufwendigste Teil, da nur für diesen Teil die Inverse bestimmt werden muss. Die Inverse wird natürlich außerhalb dieser Schleife nur einmal berechnet, dennoch könnte man ohne diesen Teil vollständig auf die direkte Berechnung der Inversen verzichten. Ist die Inverse bestimmt, liegt die Komplexität zur Berechnung der Spur bei $\mathcal{O}(n^2)$.

Der Hauptaufwand liegt also in der Berechnung der Inversen. Der genaue Ablauf zur Bestimmung der Inversen folgt dem Schema aus Kapitel 5.5.1. Grundlegend besteht dieses Schema aus zwei Schritten:

1. Cholesky Zerlegung der Kovarianzmatrix
2. Vorwärts- und Rückwärtssubstitution zur Bestimmung der Inversen

Der Aufwand beider Schritte liegt bei

1. Ungefähr $\frac{1}{6}n^3$Multiplikationen/Additionen, $\frac{1}{2}n^2$ Divisionen, n Wurzeloperationen
2. Vorwärts- und Rückwärtseinsetzen insgesamt: n^3 Multiplikationen/Additionen

Der Hauptaufwand liegt also bei der Vorwärts- und Rückwärtssubstitution. Wobei sich diese für den Fall einer Invertierung hervorragend parallelisieren lässt. Da man das Vorwärts- und Rückwärtseinsetzen bei der Invertierung über n Vektoren macht, kann man die Berechnung über die Vektoren parallelisieren. SIMD Routinen sind hier besonders effizient, da für jeden Vektor immer dieselbe Routine durchlaufen wird und nur die Daten sich ändern. Eine GPU, SSE oder AVX Beschleunigung ist hier also besonders anzustreben. Denkbar ist aber auch eine Parallelisierung auf Prozessebene, diese ließe sich nach demselben Schema aufteilen und die Teile dann natürlich auch über SIMD Befehle beschleunigen. Im Kapitel **??** wird eine mögliche Umsetzung der prozessweiten Parallelisierung aufgezeigt.

Ein kompletter Verzicht auf die Vorwärts Rückwärtssubstitution wäre dennoch erstrebenswert, da diese den größten Teil des Aufwands ausmacht. In Kapitel 5.4.3 wurde aufgezeigt, dass nur die Ableitung der Determinante der Kovarianzmatrix die Rückwärts- und Vorwärtssubstitution benötigt. Es gilt also eine andere Möglichkeit der analytischen Berechnung für diese Ableitung zu finden, die eine kürzere Laufzeit verspricht.

Einen interessanten Ansatz hierzu kann man in [Toal et al., 2009] finden. Dieser bedient sich der algorithmischen Differentiation im Rückwärtsmodus, der interessierte Leser sei auf [Mader et al., 2008, Griewank & Walther, 2008] verwiesen, welche einen sehr guten Überblick über die algorithmische Differentiation bieten. Die grundlegende Idee in diesen Ansätzen ist es den gesamte Likelihood Term rückwärts zu differenzieren. Dieser Ansatz bietet die Möglichkeit auf die Vor- und Rückwärtssubstitution zu verzichten, allerdings werden rückwärtsdifferenzierte Algorithmen des Cholesky Algorithmus und auch der Vor- und Rückwärtssubstitution benötigt. Für diese Algorithmen gibt es keine performante Implementation über Bibliotheken.

Aus diesem Grund in [Toal et al., 2011] ein Algorithmus vorgeschlagen, welcher die Invertierung zwar benötigt, aber keine rückwärtsdifferenzierten Algorithmen. Ein Geschwindigkeitsvorteil wird hierbei bei der Aufstellung der partiellen Ableitungen der Kovarianzmatrix $\frac{\partial \mathbf{Cov}}{\partial h_l}$ erreicht. Diese muss bei dem verwendeten Algorithmus nicht mehr direkt erzeugt werden. Die Vorwärts- und Rückwärtssubstitution muss dennoch durchgeführt werden. Diese stellt grundlegend auch den Hauptanteil bei der Berechnung des Likelihoods und der Ableitungen dar.

Eine andere Möglichkeit bietet die alleinige Rückwärtsdifferentiation der Determinante der Kovarianzmatrix. Dieser Ansatz bietet einen grundsätzliches Geschwindigkeitsvorteil,wie sich im Weiteren herausstellen wird. Zudem bietet dieser Ansatz den Vorteil, dass der Quellcode sich nur minimal ändert und für alle Kriging-Verfahren und auch Korrelationsfunktionen gilt. Es wird allerdings eine rückwärtsdifferenzierte Version des Cholesky Algorithmus benötigt. Für diese bestehen mittlerweile aber sehr effiziente BLAS-Implementierungen, diese werden im weiteren Verlauf vorgestellt. Der grundlegende Ansatz ist die Bestimmung der formellen Ableitung der gesuchten Ableitung:

$$\frac{\partial \left(ln \left(det \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l}$$

Wobei die Determinante das Produkt über alle quadrierten Diagonalelemente der Cholesky zerlegten Dreiecksmatrix LL^T ist:

$$ln \left(det \left(\mathbf{Cov} \left(h_l \right) \right) \right) = ln \left(\prod_i L_{i,i}^2 \right)$$

Wobei folgende Form in der Regel bevorzugt wird, da diese numerisch stabiler ist:

$$= 2 \sum ln \left(L_{i,i} \right)$$

Daraus ergibt sich die folgende Verkettung, welche die Cholesky Zerlegung berücksichtigt f_{chol}

$$\frac{\partial \left(ln \left(det \left(f_{chol} \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right) \right)}{\partial h_l}$$

Die bestehenden Abbildungen sehen wie folgt aus;

$$\mathbf{Cov} : \mathbb{R} \longmapsto \mathbb{R}^{n^2}$$

$$f_{chol} : \mathbb{R}^{n^2} \longmapsto \mathbb{R}^{n^2}$$

$$det : \mathbb{R}^{n^2} \longmapsto \mathbb{R}$$

$$ln : \mathbb{R} \longmapsto \mathbb{R}$$

Der Logarithmus der Determinante wird als f_{lndet} zusammengefasst

$$\frac{\partial \left(f_{lndet} \left(f_{chol} \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l}$$

$$f_{lndet} : \mathbb{R}^{n^2} \longmapsto \mathbb{R}$$

es gilt also

$$f_{lndet} \circ f_{chol} \circ \mathbf{Cov} : \mathbb{R} \longmapsto \mathbb{R}$$

daraus resultieren die Jacobi Matrizen D_{cov} der Größe $n^2 \times 1$, D_{chol} der Größe $n^2 \times n^2$ und D_{flndet} der Größe $1 \times n^2$ wobei $\mathbf{C_{i,j}}$ einen Eintrag der Matrix $\mathbf{Cov}$ bedeutet.

Damit wird die gesamte Ableitung zu:

$$\frac{\partial \left(f_{lndet} \left(f_{chol} \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l} = D_{flndet} D_{chol} D_{cov}$$

$$\frac{\partial \left(f_{lndet} \left(f_{chol} \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l} = \sum_i \sum_{j<=i} \sum_k \sum_{m<=k} \frac{\partial f_{lndet}}{\partial \mathbf{L_{i,j}}} \frac{\partial \mathbf{L_{i,j}}}{\partial \mathbf{C_{k,m}}} \frac{\partial \mathbf{C_{k,m}}}{\partial h_l} \tag{5.10}$$

Es gilt nun sich zu überlegen, wie man die hier gezeigte mehrdimensionale Kettenregel möglichst effizient bestimmen kann. Berechnet man die vollständige Summe, dann wäre die Komplexität für die Berechnung der Ableitung bei $\mathcal{O}\left(n^4\right)$. Der Aufwand wäre also deutlich größer als über die Bestimmung der Inversen. Benötigt wird also ein Algorithmus, welcher die notwendigen Terme der mehrdimensionalen Kettenregel ohne große Matrizen berechnen kann. Als erste Vereinfachung kann man sich versuchen die einzelnen Jacobi Vektoren zu bestimmen. Der Vektor D_{cov} entspricht einfach nur allen Einträgen der Matrix $\frac{\partial \mathbf{Cov}}{\partial h_l}$ und ist im Kriging daher bekannt. Als nächstes kann man den Vektor D_{flndet} bestimmen, da man die Funktion f_{lndet} kennt:

$$D_{flndet}^T = \begin{pmatrix} \frac{\partial f_{lndet}}{\partial \mathbf{L_{1,1}}} \\ \frac{\partial f_{lndet}}{\partial \mathbf{L_{2,1}}} \\ \vdots \\ \frac{\partial f_{lndet}}{\partial \mathbf{L_{n-1,n}}} \\ \frac{\partial f_{lndet}}{\partial \mathbf{L_{n,n}}} \end{pmatrix} = \begin{pmatrix} \frac{\partial 2 \sum ln(L_{i,i})}{\partial \mathbf{L_{1,1}}} \\ \frac{\partial 2 \sum ln(L_{i,i})}{\partial \mathbf{L_{2,1}}} \\ \vdots \\ \frac{\partial 2 \sum ln(L_{i,i})}{\partial \mathbf{L_{n-1,n}}} \\ \frac{\partial 2 \sum ln(L_{i,i})}{\partial \mathbf{L_{n,n}}} \end{pmatrix} = \begin{pmatrix} \frac{2}{L_{1,1}} \\ 0 \\ \vdots \\ 0 \\ \frac{2}{L_{n,n}} \end{pmatrix}$$

Die Bestimmung dieses Vektors kann auch als Diagonalmatrix interpretiert werden welche sehr schnell aus der Cholesky zerlegten Matrix bestimmt werden kann.

$$\bar{D}_{flndet} = \begin{bmatrix} \frac{2}{L_{1,1}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{2}{L_{n,n}} \end{bmatrix}$$

Auffällig ist hierbei, das der Großteil des Jacobi Vektors aus 0 Einträgen besteht, mit dieser Information lässt sich Gleichung 5.10 stark vereinfachen, da nur noch die Einträge $\frac{\partial f_{lndet}}{\partial \mathbf{L_{i,j}}} \neq 0 \,|i = j$ sind:

$$\frac{\partial \left(f_{lndet}\left(f_{chol}\left(\mathbf{Cov}\left(h_l\right)\right)\right)\right)}{\partial h_l} = \sum_i \sum_k \sum_{m<=k} \frac{\partial f_{lndet}}{\partial \mathbf{L_{i,i}}} \frac{\partial \mathbf{L_{i,i}}}{\partial \mathbf{C_{k,m}}} \frac{\partial \mathbf{C_{k,m}}}{\partial h_l} \tag{5.11}$$

Die Komplexität liegt somit nur noch bei $\mathcal{O}\left(n^3\right)$.

Als nächsten Schritt muss man sich nun überlegen, wie man den mittleren Teil der Kettenregel ($\frac{\partial \mathbf{L_{i,i}}}{\partial \mathbf{C_{k,m}}}$) bestimmen kann. Hierfür gibt es prinzipiell zwei Möglichkeiten:

1. Man geht von den Werten $\frac{\partial \mathbf{C_{k,m}}}{\partial h_l}$ aus und bestimmt ausgehend von diesen die Einträge von $\frac{\partial \mathbf{L_{i,i}}}{\partial h_l}$ über eine vorwärts differenzierte Cholesky Zerlegung
2. Man geht von den Werten $\frac{\partial f_{lndet}}{\partial \mathbf{L_{i,i}}}$ aus und bestimmt ausgehend von diesen die Einträge von $\frac{\partial f_{lndet}}{\partial \mathbf{C_{k,m}}}$über eine rückwärts differenzierte Cholesky Zerlegung

Grundsätzlich sollten beide Vorgehensweise vom numerischen Aufwand gleichwertig sein. Da man allerdings für jeden der o-Hyperparameter h_l die Matrix $\frac{\partial \mathbf{C_{k,m}}}{\partial h_l}$ bestimmen muss und damit wiederum $\frac{\partial \mathbf{L_{i,i}}}{\partial \mathbf{C_{k,m}}}$, muss man also die vorwärts differenzierte Cholesky Zerlegung o-mal aufrufen. In Fall 2 ist dies nicht so, denn die Berechnung ist unabhängig von der Anzahl der Hyperparameter. Aus diesem Grund verspricht der rückwärtsdifferenzierte Fall einen Vorteil bei der Bestimmung der partiellen Ableitungen der Hyperparameter.

Der numerische Aufwand für einen solchen rückwärtsdifferenzierten Cholesky Algorithmus liegt bei dem ungefähr doppelten Aufwand einer normalen Cholesky Zerlegung [Smith, 1995]. Das ist immer noch deutlich schneller als eine Vor- und Rückwärtssubstitution. Der in [Smith, 1995] beschriebene Algorithmus ist allerdings nur schwer parallelisierbar und auch für SIMD Architekturen nur schlecht geeignet. In Anhang A.9 wird eine parallelisierte und SIMD-beschleunigte Variante des Algorithmus nach Smith-Algorithmus präsentiert.

In der Arbeit von [Murray, 2016, Särkkä, 2013] werden einige moderne Varianten präsentiert, welche es ermöglichen Standard Level 2-3 BLAS Routinen zu verwenden.

Diese stellen die effizientesten Methoden dar und werden folgend auch verwendet.

5.4.5 Vergleich zwischen der Invertierung und der Rückwärtsdifferenziereu

Bei dem hier vorgestelltem Algorithmus handelt es sich prinzipiell nicht um eine klassische Rückwärtsdifferenzierung wie z.B. in [Toal et al., 2009]. Diese differenzieren die gesamte Ableitung des Likelihoods rückwärts. Diese Algorithmen haben jedoch den Nachteil, dass der Quellcode deutlich komplexer und schwerer zu warten ist, insbesondere wenn man mehrere Korrelationsfunktionen und Verfahren anbietet. In beiden Fällen ist es aber nicht direkt ersichtlich, dass die Rückwärtsdifferenzierte Methode die schneller ist. Die Vermutung ist allerdings, dass die Algortihmen der Rückwärtsdifferenzierung weniger Operationen benötigen und sich besser beschleunigen lassen. Ein direkter Vergleich der benötigten Operationen innerhalb der Algorithmen ist allerdings so gut wie unmöglich, da meist sehr stark optimierte Bibliotheken wie die Intel MKL oder ATLAS Bibliothek verwendet werden. Diese bedienen sich oft komplexer Algorithmen wie z.B. den Strassen Algorithmus, welcher eine Matrix Multiplikation mit einer Komplexität von $\mathcal{O}\left(n^{log_2 7}\right)$ berechnen kann. Oft werden die Algorithmen je nach Matrix Größe umgeschaltet, sodass sich die echte Komplexität kaum noch bestimmen lässt. In der Regel wird diese auch nicht dokumentiert. Dennoch lässt sich davon ausgehen, dass diese Bibliotheken das aktuelle Maximum an Geschwindigkeit darstellen, sodass eine gute und auch realistische Vergleichbarkeit gewährleistet ist. Aus diesem Grund sollen in diesem Abschnitt die Unterschiede der Verfahren über Benchmarks ermittelt werden.

Betrachtet man den Teil der Berechnung zur Bestimmung der Ableitung des Likelihoods nach den Hyperparameter, in dem sich die beiden Verfahren unterscheiden, so handelt es sich prinzipiell nur um einen Term:

$$\frac{\partial \left(ln \left(det \left(\mathbf{Cov} \left(h_l\right)\right)\right)\right)}{\partial h_l}$$

Die analytische Ableitung ist wie folgt bestimmt:

1. $\mathrm{Spur}\left(\mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l}\right)$

Für beide Verfahren gilt die gleiche Ausgangssituation: Die nach den Hyperparametern abgeleitete Kovarianzmatrix ist bekannt $\frac{\partial \mathbf{Cov}}{\partial h_l}$ und auch die Cholesky-zerlegte Matrix $\mathbf{L}$ ist bekannt. Ziel ist es die Ableitung nach den Hyperparametern des Terms $ln \left(det \left(\mathbf{Cov} \left(h_l\right)\right)\right)$ zu bestimmen.

Die Bestimmung der vorwärtsdifferenzierten Ableitung benötigt folgende Schritte:

1. Bestimmung der differenzierten Kovarianzmatrix $\frac{\partial \mathbf{Cov}}{\partial h_l}$ für jeweils einen Hyperparameter h_l
2. Bestimmung der Cholesky Zerlegung $\mathbf{L}$
3. Bestimmung von $\mathbf{Cov}^{-1}$, da die zerlegte Matrix bekannt ist, muss noch eine Vor- und Rückwärtssubstitution durchgeführt werden diese benötigt relativ genau n^3 Operationen.
4. Die Bestimmung der quadratischen Form $\left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)^T \mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l} \mathbf{Cov}^{-1} \left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)$
5. Das Matrixprodukt $\mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l}$, wobei hiervon nur die Diagonale berechnet werden muss, daher liegt diese Berechnung bei ca. n^2 Operationen. Durch die Spur kann man sparen, es gibt keine BLAS Routine dafür.

Die Bestimmung der Rückwärtsdifferenzierten Ableitung benötigt folgende Schritte.

1. Bestimmung der differenzierten Kovarianzmatrix $\frac{\partial \mathbf{Cov}}{\partial h_l}$ für jeweils einen Hyperparameter h_l
2. Bestimmung der Cholesky Zerlegung $\mathbf{L}$
3. Die Bestimmung der Rückwärtsdifferenzierten Cholesky Zerlegung $\frac{\partial \mathbf{L_{i,i}}}{\partial \mathbf{C_{k,m}}}$ inlkusive Aufstellen der Seed Matrix $\bar{\mathbf{L}}$ (siehe $\frac{\partial f_{lndet}}{\partial \mathbf{L_{i,i}}}$) dieses benötigt n Operationen
4. Die Bestimmung der quadratischen Form $\left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)^T \mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l} \mathbf{Cov}^{-1} \left(\vec{y}_s - \beta_1 \overrightarrow{F}\right)$
5. Die Bestimmung des Produkts der aus Punkt 2 berechneten Matrix und den Einträgen der differenzierten $\frac{\partial \mathbf{Cov}}{\partial h_l} \sim \frac{n^2}{2} + \frac{n}{2}$

Letztlich zählt für den Vergelcih haupütsächlich der Unterschied zwischen der Vorwärts- und Rückwärtssubstitution und der Rückwärtsdifferenzierten Cholesky Zerlegung. da beides über Intel MKL Routinen umgesetzt wurde, ist nicht bekannt wieviele Operationen wirklich gemacht werden. Daher wird folgend ein zeitlicher Vergleich über ein Benchmark angestellt. Erwartet wird ein konstanter Faktor zugunsten der Rückwärtsdifferenziereung.

Als Benchmark dienten zufällig erzeugte Testdaten aus der ZDT3 Funktion (siehe Kapitel 6) mit jeweils 8 Parametern. Es wurden jeweils 10 Kriging Iterationen durchgeführt und die Zeiten für die Erzeugung der partiellen Ableitungen des Likelihoods gemessen. Berechnet wurde der Testfall dabei zwei Xeon E5 2640 v3 mit insgesamt 16 Threads. Für alle Berechnungen wurde die Intel MKL Bibliothek verwendet.

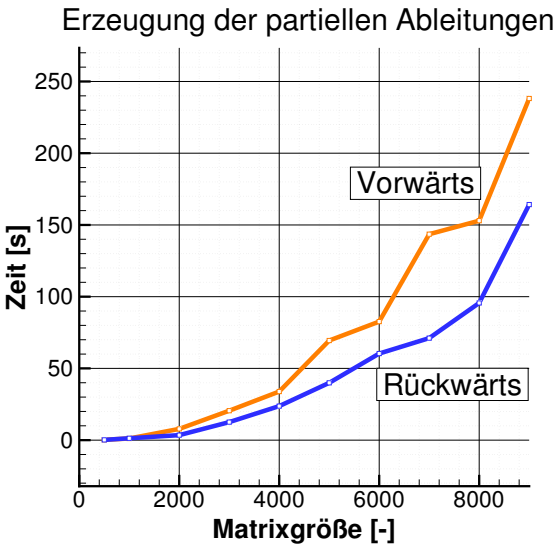


Abbildung 5.11: Vergleich der benötigten Gesamtzeit für die Erzeugung der partiellen Ableitungen im Vorwärts- und Rückwärtsmodus über verschiedene Matrixgrößen

Abbildung 5.11 zeigt die benötigte Zeit für die Erzeugung der partiellen Ableitungen. Der Rückwärtsmodus war in diesem Fall grundlegend schneller als der Vorwärtsmodus. Im Schnitt konnte ein konstanter Faktor zwischen den beiden Berechnungsmoden gefunden werden der bei ca. 0.68 lag.

Die folgenden Abbildungen zeigen die Zeiten der verschiedenen Schritte im Vergleich:

Wie erwartet sind die Zeiten vom ersten und vom vierten Schritt nahezu identisch, da die Berechnung letztlich auch dieselbe ist. Unterschiede können in Schritt 2-3 und Schritt 5 gesehen werden, in beiden Fällen ist der Rückwärtsmodus schneller.

5.5 Verwendung von GPGPU

In [Toal, 2016] wird die Verwendung von GPUs innerhalb des Kriging Verfahren bereits als gewinnbringend beschrieben. Innerhalb dieses Kapitels soll eine konkrete Umsetzung gezeigt werden und zudem auch ein Vergleich verschiedenster aktueller GPUs und CPUs angestellt werden.

Institutsvortrag Erklärungen nehmen !!!

General Purpose Computation on Graphics Processing Unit blablabla#

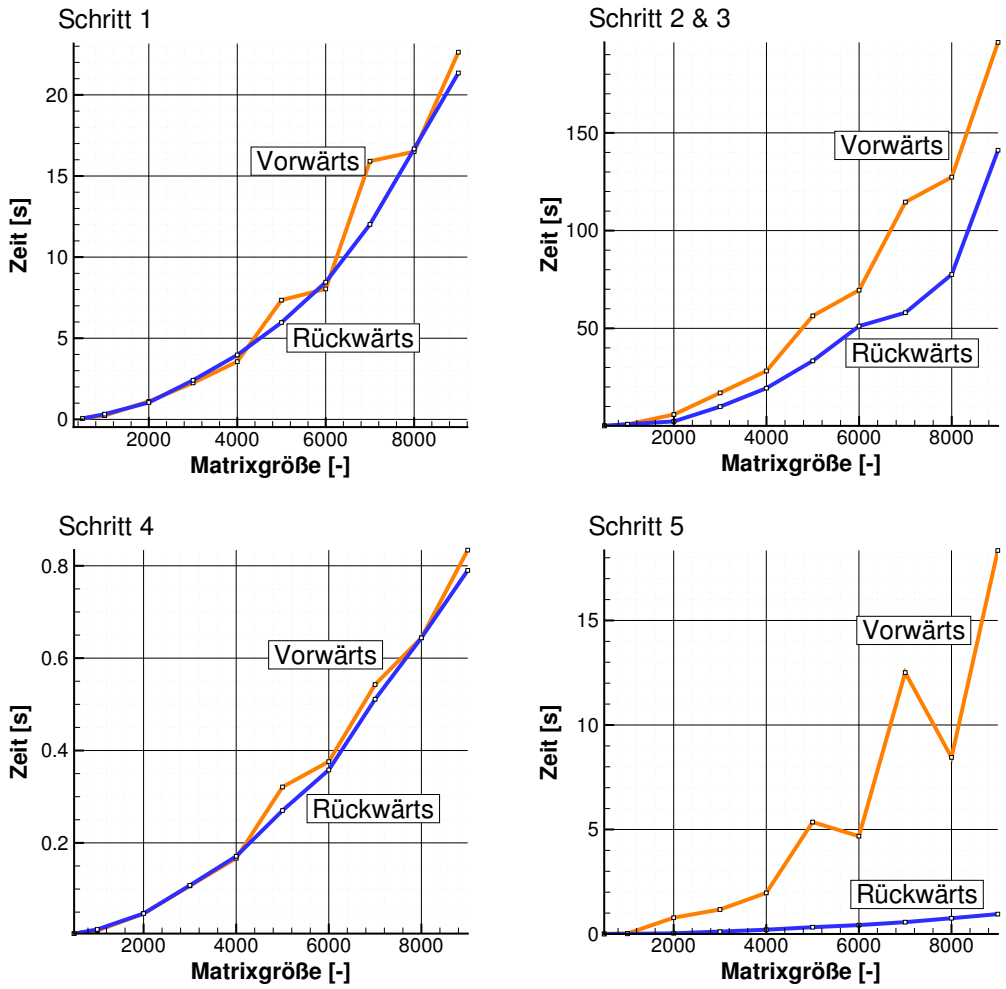


Abbildung 5.12:

5.5.1 Adjoint Matrix

Level 3 BLAS Umsetzung

Benchmarks Einzelne Rückwärts Cholesky

Komplettes GTraining mit verschiedenen Matrixgrößen

Mehrere Korrelationsfunktionen GEK und CO-Kriging sind hier sehr problematisch (SICHER?), da sich die Ableitung des Likelihood nach den Hyperparametern komplexer gestaltet und Softwaretechnisch nur noch schlecht zu lösen sind.

- Rverse Vorwärts und Rückwärts und Reverse Choelsky muss durchlaufen werden, ansonsten sehr elegant und schneller als die normale Implmentierung

In [Toal et al., 2011] wird dieser Ansatz nochmals verändert und mithilfe der linearen Algebra Ansätze von [Giles, 2008] auf eine Form gebracht, die auf die Rückwärtsdifferenzierung des Cholesky Algorithmus und der Vor- und Rückwärtssubstitution verzichtet. Allerdings wird hierfür die Inverse der Kovarianzmatrix benötigt, also auch die Vor- und Rückwärtssubstitution. Dieser Umstand wird in [Toal et al., 2011] als Vorteil angesehen, da masn Standardbibliotheken für lineare Algebra ohne Probleme weiterverwenden kann und die Bestimmung der Likelihood Ableitungen etwas .

Eine andere Möglichkeit ist die Bestimmung ....
-Trugschluss da Reverse Cholesky nur 2x Aufwand von Cholesky also immer nopch viel schneller als Vor und R+ückwärts

-Reverse Vor und Rück wird sowieso nicht benötigt, wenn nur die ableitung nach Determinante

- Reverse Cholesky wird nur eine effizienze Implementierung verwendet, hier wird eine aufgezeigt welche von 8sec auf 0.2sec bei 20 CPUs und SSE runter ist. in [Murray, 2016] wird auch eine Implementierung gezeugt, welche die Verwendung von Standardblibiotheksunftionen ermöglicht.

– - - - - - - - - - - - - - – - - - - - - - - - - - – - - - - - - - - - - - - - – - - - - - - - - - - -
- -

– - - - - - - - - - - - - - – - - - - - - - - - - - – - - - - - - - - - - - - - – - - - - - - - - - - -
- -

– - - - - - - - - - - - - - – - - - - - - - - - - - – - - - - - - - - - - - - - – - - - - - - - - - - -
- -

In wird ein Verfahren dargestellt, welches die gesamte Likelihood Berechnung rückwärts differenziert. Für dieses Verfahren ist es allerdings notwendig die komplette Vor- und Rückwärtssubstitution zu durchlaufen, zudem muss dieses Verfahren für jede

neue Korrelationsfunktion umgeschrieben werden, was es softwaretechnisch kompliziert macht. Der Aufwand der Vor- und Rückwärtssubstitution würde dadurch ebenfalls nicht wegfallen oder vermindert werden. Würde eine Beschleunigung in der Bestimmung der partiellen Ableitungen der Kovarianzmatrix bringen. Der Nachteil liegt jedoch darin, dass man eine Reverse Version der Cholesky Zerlegung und der Vorwärts- und Rückwärtssubstition benötigt. Dies würde eine Benutzung von Libraries wie CUDA oder der Intel MKL verhindern. Die Beschleunigung durch CUDA oder MKL wird von uns als deutlich höher eingeschätzt als der Zugewinn durch die adjungierte Kovarianzmatrix. Aus diesem Grund wird darauf verzichtet.

Grundlegend ist es nicht nötig die gesamte Likelihood Berechnung rückwärts zu differenzieren, denn der einzig wirklich problematische Term ist $\frac{\partial}{\partial h_l}\left(\log\left(\det(\mathbf{Cov})\right)\right)$. Für diesen Term ist es zwingend notwendig die Inverse der Kovarianzmatrix zu berechnen. Es ist also ausreichend, nur diesen Term rückwärts zu differenzieren.

In wird die Cholesky Zerlegung rückwärts differenziert, mithilfe dieses Algorithmus ist es möglich die Ableitung der Determinante ohne vorherige Invertierung der Kovarianzmatrix zu erhalten.

Code Optimierung vergleich, zwischen dem Originalcode und dem optimierten, Native & SSE !!!!!!

Der Code wird deutlich unleserlicher und schwerer zu pflegen, deswegen nur für feste Code Bestandteile sinnvoll

_mm_load_pd(*adress) ist aligned und daher extrem schnell (wenn ich auf einer geraden Anzahl bin vom Array)

_mm_loadu_pd(double,double) is unaligned und daher sehr langsam(wenn ich auf einer ungeraden Anzahl bin vom Array)

Benchmark: alte Version bereits mit SSE und Parallelisierung, Gleichungssysteme stattt Inverse benutzt, nur für RPunkt die Inverse

Nur die chodecReverse und invertierung verlgiehcen, da der Rest prinzipiell gleich ist

asd

Erklären wie SSE durch den Speicher geht, also Speicher in 4er Blöcken aligned und wenn man außerhalb davon zugreift, liest die CPU die nächsten 2 Adressen und die 2te ist dann woanders. Das führt danndazu, mdass das Betriebssystem meckert und dem Prozess nicht erlaubt den Bereich zu lesen-> Speicherzugriffsfehöer

```
register int i;
int n = this->getColumnSize();
```

```
        if(this->getColumnSize() != this->getRowSize()){
cout <<"Matrix nicht quadratisch!\n";
return false;         }       if(this->getColumnSize() <=0){
cout <<"Matrix Größe <= 0!\n";          return false;         }
        adjointMat.setSymmetric(true);
        // Set Seed Matrix #pragma omp parallel for
for(int i=0; i<n; i++){             adjointMat[i][i] = 2.0/(*this)[i][i]
}                           for(int k=n-1; k>=0; k--){
for(int j=k+1; j<n;j++){                for(int i=j; i<n;i++){
adjointMat[i][k] -= adjointMat[i][j]*(*this)[j][k];
adjointMat[j][k] -= adjointMat[i][j]*(*this)[i][k];
}           }                           for(int j=k+1; j<n;j++){
adjointMat[j][k] /= (*this)[k][k];
adjointMat[k][k] -= adjointMat[j][k]*(*this)[j][k];
}           adjointMat[k][k]=0.5*adjointMat[k][k]/(*this)[k][k];
}

        adjointMat.setSymmetric(true);
        return true;
```

Blocked Algorithmus

Name	Max. GFlops (double FMA)	Straßenpreis (8.8.2016)	max. Leistung	Watt/GFlop
E7-8890v4	844.8	7600€	165W	0.195
E5-4669v4	774.4	7500€	135W	0.174
E5-2699v3	662.4	3830€	145W	0.219
E5-2698v3	588.8	3000€	135W	0.229
E5-2650v4	422.4	1200€	105W	0.249
E5-2650v3	368	1200€	105W	0.285
E5-2695v2	230.4	2100€	115W	0.5
E5-2620v2	100.8			

sdfgsdf

sdf

Name	Max. GFlops (double FMA)	Straßenpreis (
Tesla P100 NVLink	5300	Unbeka
Tesla P100 PCIe	4700	7300€
Tesla K80	1863(@BaseClock 560MHz), 2912@BoostClock(875MHz)	5800€
Tesla K40	1430	4400€
Quadro K6000	1732@901Mhz, 1152@600Mhz	5000€
GTX1080	277	800€

Intel

Name	Max. GFlops (double FMA)	Straßenpreis (8.8.2016)	GPU RAM	GB/s	max. Lei
7290*	3456	6800€	16GB	490	245V
7250	3046	5400€	16GB	490	215V
7120	1210	4000€	16GB (DDR4)	352	300V

72xx = (KnightsLanding)

71xx = KnightsCorner

K80 = 2 GPUs erwähnen

Cholesky Zerlegung

Die Methode chodec() stellt eine Cholesky Zerlegung für positiv definite und symmetrische Matrizen bereit, für die Matrix $\mathbf{R}$ muss also gelten:

$$\mathbf{R} = \mathbf{R}^T$$

und

$$\overrightarrow{v} * \mathbf{R} * \overrightarrow{v} > 0 \; für alle Vektoren \; \overrightarrow{v}$$

Bei der Cholesky Zerlegung wird die Matrix $\mathbf{R}$ in ein Produkt aus einer unteren Dreicksmatrix und deren Transponierten zerlegt:

$$\mathbf{L}\mathbf{L}^T = \mathbf{R}$$

Die Zerlegung kann nun verwendet werden, um durch Vor- und Rückwärtseinsetzen lineare Gleichungssysteme in der Form $\mathbf{R}\overrightarrow{x} = b$ zu lösen. Hierfür wird zuerst vorwärts eingesetzt:

$$\mathbf{L}\overrightarrow{y} = \overrightarrow{b}$$

und dann durch Rückwärtseinsetzen kann der gesuchte Vektor $\overrightarrow{x}$ erhalten werden:

$$\mathbf{L}^T\overrightarrow{x} = \overrightarrow{y}$$

$$\Longrightarrow \mathbf{R}\overrightarrow{x} = \mathbf{L}\mathbf{L}^T\overrightarrow{x} = \mathbf{L}\overrightarrow{y} = \overrightarrow{b}$$

ersetzt man nun $\overrightarrow{x}$ durch $\mathbf{R}^{-1}$ und $\overrightarrow{b}$ durch die Einheitsmatrix $\mathbf{E}$, kann man mit der Zerlegung die Inverse der Matrix $\mathbf{R}$ berechnen.

Nach diesem Schritt können die Vor- und Rückwärtssubstitutionen spaltenweise durchgeführt werden. Diese lassen sich sehr gut parallelisieren, da jede CPU einfach einen Vektor der Inversen berechnet.

Ein weiterer Vorteil der Zerlegung ist, dass die Determinante der Matrix $\mathbf{R}$ mit der Zerlegung einfach durch Multiplikation der Diagonalelemente der zerlegten Matrix gewonnen werden kann:

$$det\left(\mathbf{R}\right) = \prod_{i=1}^{n} L_{i,i}^{2}$$

Der verwendete Algorithmus ist in [**?**, Gill, 2007] nochmals detaillierter beschrieben.

Quadratische Form

Da bei dem Training eines Kriging Modells häufig quadratische Formen $\vec{v^T} R \vec{v}$ berechnet werden müssen, lohnt es sich diese zu beschleunigen. Dies wird dadurch gemacht, dass die Multiplikationen nicht nacheinander durchgeführt werden, sondern beide Multiplikationen in einer doppel-Schleife behandelt werden.

$$\sum_{i=1}^{n} \left(v_i \sum_{j=1}^{n} R_{i,j} * v_j \right) = \vec{v}^T R \vec{v}$$

Der Algorithmus ist im folgenden Listing gezeigt:

```
#pragma omp parallel for reduction(+:ret)
for(size_t row=0; row < matrix->getRowSize(); row++){
        double sum=0.;
        for(size_t col=0; col < matrix->getColumnSize(); col++){
                sum += matrix[row][col] * vec[col];
        }
        sum *= vec[row];
        result +=sum;
}
```

In der ersten Zeile wird eine Schleifenparallelisierung über OpenMP initialisiert, wobei das Aufsummieren von result nicht parallelisiert werden darf. Dies würde sonst zu unvorhersehbaren Fehlern führen, da mehrere Threads gleichzeitig in die Variable result schreiben möchten.

Trainings	Anteil GTX	Anteil K6000	Max. Time GTX	Max. Time K6000	Time
16	0	16	0s	156s	156s
	1	15	130s	143s	143s
	2	14	133s	132s	133s
	3	13	191s	128s	191s
12	0	12	0s	133s	133s
	1	11	113s	121s	121s
	2	10	122s	119s	122s
	3	9	187s	113s	187s
8	0	8	0s	96s	96s
	1	7	101s	85s	101s
	2	6	142s	83s	142s
	3	5	178s	82s	178s
4	0	4	0s	72s	72s
	1	3	98s	71s	98s
	2	2	123s	66s	123s

Tabelle 5.2: Benchmark der besten Verteilung mehrerer Trainings mit 5000 Samples auf 2 verschiedene GPUs

5.6 Resourcenverteilung bei parallelen Trainings

1. Nur GPUs aufteilen, da CPUs immer identisch sind und die GPUs auch CPU Anteile verwenden die immer gleich sind von der Berechnung her.
2. Speicher auf den GPUs checken und schauen wieviele Krigings maximal auf eine GPU gehen. Damit wird der jeweilige Anteil limitiert.
3. Anteil berechnen
   (a) Abhängig von der Matrixgröße (Aufteilung über GFlops sinnvoll)
   (b) Abhängig von der Anzahl der gleichzeitigen Trainings

2.

Der Speicherbedarf in Megabyte pro Matrix liegt bei $\frac{64Bit*n^2}{8*1024*2014} = r$. Während eines Trainings werden maximal 2 dieser Matrizen auf einer GPU allokiert.

Scheduler, Resourcenverteilung, verschiedene Resourcen, passt alles in den RAM

1000x1000

500x500

Trainings	Anteil GTX	Anteil K6000	Max. Time GTX	Max. Time K6000	Time
16	0	16	0s	18s	18s
	1	15	11s	16s	16s
	2	14	11s	16s	16s
	3	13	12s	16s	16s
	6	10	13s	15.5s	15.5s
	10	6	14.8s	14.8s	14.8s
8	0	8	0s	10s	10s
	1	7	6s	8s	8s
	2	6	6s	7s	7s
	3	5	7s	7s	7s
4	0	4	0s	5s	5s
	1	3	4s	5s	5s
	2	2	5s	4s	5s

Tabelle 5.3: Benchmark der besten Verteilung mehrerer Trainings mit 1000 Samples auf 2 verschiedene GPUs

Trainings	Anteil GTX	Anteil K6000	Max. Time GTX	Max. Time K6000	Time
16	0	16	0s	17.7s	
	1	15	10.4s	16.2s	
	2	14	10.2s	16s	
	3	13	10.2s	15.1s	
	9	7	10.4s	12.31s	
	12	5	10.2s	11.3s	
	14	2	10.2s	10.8s	
	15	1	10.2s	10.5s	
8	0	8	0s	7.7s	
	1	7	4.5s	6.5s	
	2	6	4.4s	5.9s	
	3	5	4.4s	5.7s	
	4	4	4.1s	5.2s	
	7	1	4.4s	4.4s	
4	0	4	0s	3.7s	
	1	3	2.4s	3.1s	
	2	2	2.2s	2.7s	
	1	3	2.4s	2.4s	

Tabelle 5.4: Benchmark der besten Verteilung mehrerer Trainings mit 1000 Samples auf 2 verschiedene GPUs

5.7 Analysesoftware von Krigingmodellen während der Laufzeit

Khon.py beschreiben

5.7.1 Hyperparameter aus zusammengesetzten Kovarianzfunktionen plotten

Angenommen alle Parameter außer dem betrachteten Parameter spielen keine Rolle, dann vereinfacht sich die Gauss-Korrelations-Formel zu:

$$cov\left(x_1, x_2\right) = a^2 * \sigma_l^2 * e^{-e^{\theta_l}|x_1 - x_2|^2} + \sigma_{err}^2 * e^{-e^{\theta_{err}}|x_1 - x_2|^2}$$

Nimmt man weiterhin an, dass der Abstand $|x_1 - x_2| = 1$ ist, vereinfacht sich die Formel weiter. Diese Annahme ist in der hier verwendeten Implementierung sinnvoll, da alle Koordinaten auf einen Erwartungswert von 0 und eine Standardabweichung von 1 normiert wurden.

$$cov\left(x_1, x_2\right) = a^2 * \sigma_l^2 * e^{-e^{\theta_l}} + \sigma_{err}^2 * e^{-e^{\theta_{err}}}$$

Die Gesamtvarianz des Modells liegt bei:

$$\sigma_{ges}^2 = a^2 * \sigma_l^2 + \sigma_{err}^2$$

Die Korrelation wird hier als sinnvolleres Maß als die Kovarianz angesehen, da der Wertebereich bekannt ist. Die Korrelation ergibt sich wie folgt:

$$corr_{ges}\left(x_1, x_2\right) = \frac{cov\left(x_1, x_2\right)}{\sigma_{ges}^2} = \frac{a^2 * \sigma_l^2 * e^{-e^{\theta_l}} + \sigma_{err}^2 * e^{-e^{\theta_{err}}}}{a^2 * \sigma_l^2 + \sigma_{err}^2}$$

Mit dieser Formel hat man eine Abschätzung für einen zusammengefassten Hyperparameter, dieser ermöglicht z.B. Diagramme usw.

5.8 Verteiltes Rechnen

Kein MPI, Cluster unklar wie es weiter geht (Xeon Phi oder Nvidia ....) daher extern auf NVIDIA, bisher die besten Bibliotheken und Dokumentation) Rechner lokal sollen mitgenutzt werden können Verschiedene Architekturen sollen gekoppelt werden können (GPU/CPU etc.) - Notwendigkeit von GPUs Prozessweite asynchrone Parallelisierung

ZeroMQ: CUDA und ZeroMQ gekoppel:[Cook, 2012], CERN Paper mit ZeroMQ Empfehlung:[Dworak et al., 2011]

5.9 Umsetzung der Entscheidungsfunktion

Die Annahmen klären Ersatzmodelle nicht neu trainieren jedesmal. bestimmung der Sigmas durch Bedingte Varianzvorhersage usw.

6 Benchmarks

- Analytische Tests - 2D Mises Tests - 3D TRACE

6.1 GPU Benchmarks

Nvidia PSG Cluster konnten K40/K80 und m40 GPUs getestet werden. 2x E5-2698 und jeweils 4 GPUs

Die M40 hat 24GB RAM

K40 12 GB

K80 24GB

6.2 Analytische Tests Ersatzmodelle

Beispiele zeigen, wo A,B und C geändert werden und die Faktoren/Summanden sich entsprechend anpassen. (my, sigma und beta)

Ordinary Kriging

GEK

COkriging

Trainingszeit, Vorhersagegenauigkeit

6.2.1 Initialisierungsverfahren im Vergleich

In diesem Abschnitt geht es darum, die verschiedenen Initialisierungsverfahren (siehe Kapitel 5.3.2) an einem analytischen Beispiel zu testen. Als Testfunktion wurde eine 7-dimensionale $\vec{x} \in \mathbb{R}^k; k = 5$ Sinusfunktion genommen. Die High-Fidelity Funktion besteht aus dem Low-Fidelity Ergebnis, welches skaliert und verschoben wird, zusätzlich bringt die erste freie Variable noch einen zusätzlichen funktionalen Zusammenhang in die Funktion.

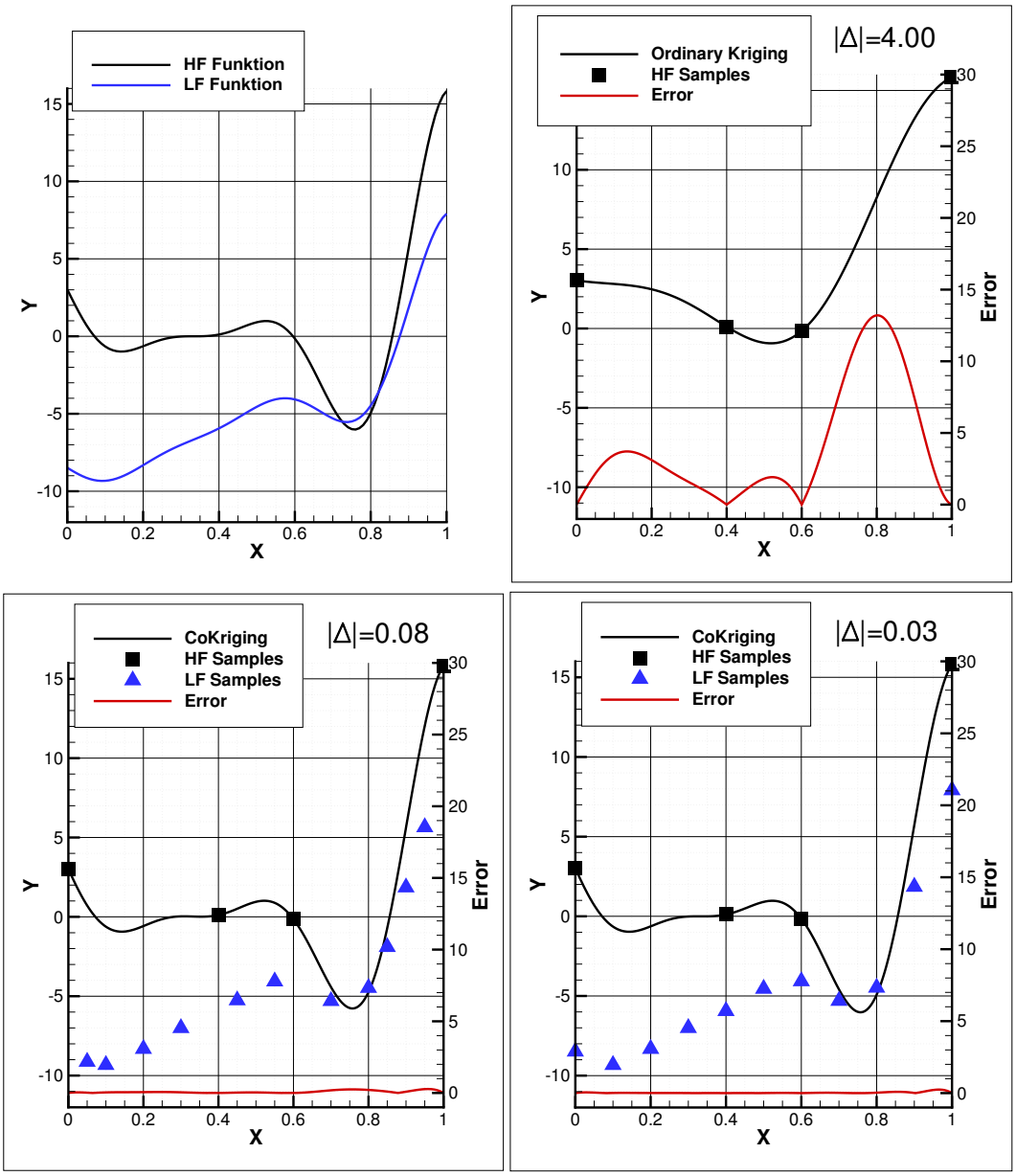


Abbildung 6.1:

$$y_{low}\left(\vec{x}\right) = \sum_{i=1}^{i<k} \sin\left(x_i * 2\pi\right)$$

$$y_{high}\left(\vec{x}\right) = 2y_{low}\left(\vec{x}\right) + 10 + \sin\left(x_0 * 2\pi\right)$$

Diese Funktionen bietet den Vorteil, dass man bereits ein erwartetes Trainingsergebnis hat und damit die Ergebnisse aus dem Kriging Training gut überprüfen kann. Erwartet werden:

- Ein Skalierungsfaktor zwischen Low-Fidelity und High-Fidelity von $a = 2$
- Globaler Erwartungswert Low-Fidelity von $\beta_{low} = 0$
- Globaler Erwartungswert High-Fidelity von $\beta_{high} = 10$
- Globale Standardabweichung Low-Fidelity Kovarianzfunktion von $\sigma_{low} = 1.58$
- Globale Standardabweichung Fehlerkovarianzfunktion von $\sigma_{err} = \frac{\sqrt{2}}{2}$
- Alle Low-Fidelity Korrelations-Hyperparameter sollten den gleichen Wert haben, $\forall \theta_{i,low} = const.$
- Korrelations-Hyperparameter der Fehlerfunktion sollten alle auf dem niedrigsten Wert stehen, bis auf den ersten

6.3 Analytische Tests Optimierung

Testoptimierung

Multifidelity

GEK

Ordinary

6.4 2 Mises Optimierung

Ordinray / Multifidelity

6.5 Trace Erich

asds

Nur die wichtigsten Sachen

Maximal

90-100 Seiten

7 Reale Turbomaschinen Optimierung

Lionels Anwendungn kurz zitieren. Lionel hätte mit einer reinen LOFI Optimierung sein Ziel nicht erreicht !!!

Reale Anwendu

- Mukoko Verdichter (ca. 215 Parameter)

- Benchmark sinnlos, da viele Einstellungen während der Optimierungs geänder werden müssen und daher ein Vergleich sehr schwierig ist

- Konvergenzprobleme sind extrem groß-> Nutzen immer das Maximum an Informationen für die ersatzmodelle, wenn nur ein Teil der Prozesskette konmvergiert ist, wird dieser verwendet

- Training asynchron, läuft permanent und die Slaves werden mit dem aktuellen Metamodel Satz erzteugt
  - 1 Knoten für das Training
  - 1 Knoten für die Optimierung auf dem Ersatzmodell
  - Slaves müssen nicht warten besser Auslastung, wenn TRACE nicht konvergiert ist es wichtig, dass die Ersatzmodelle neu trainiert werden, dadurch bleibt die Optimierung nicht stecken
- Timeline der Optimierung zeigen + Änderungen an genData oder Ersatzmodell kenntlich machen
- Zeiten zeigfen, Trainingszeit, Erzeugungszeit
  - Alles sehr hoch, nur durch die sehr lange Prozesskette sinnvoll
  - Ultra Restart hilft sehr stark
- Wie sieht die Entwicklung der Kriging Modelle über solch eine Optimierung aus?

– aßgeblich beeinfluss wird

	Strak	R1	S1	R2	S2
Variable	1-14	15-57	58-86	87-129	130-158
1-11 Aero	x	x	x	x	x
12		x			
13				x	
14-17		x			
18-21				x	
22		x			
23				x	
24		x			
25				x	
26		x			
27				x	
28-30		x			
31-32				x	
33		x			
34				x	
35-36		x			
37-38				x	

23.11.2016

Erzeugung von Low-Fidelity Membern, da diese schnell berechnet werden können und relativ gut konvergieren. Strategie ist hierbei den initialen Member zu mutieren, anders konnte keine gute Konvergenz erreicht werden. Problematisch durch die langsame Prozesskette und die vielen freien Variablen.

29.11.2016

- Die LF Member werden mit der HF Kette nachgerechnet

5.12.2016

- LF Init; ~350 LF Member erzeugt und 50 optimiert mit Ersatzmodellen und weicheren Nebenbedinungen.
- HF Init: ca. 1/6 kommen durch, erst 10 Samples konvergierte erzeugt

8.12.2016

- HF Init: Fertiggestellt ca. 25 Member konvergiert

9.12.2016

- MF Optimierung gestartet. Training läuft asynchron auf einem Knoten in der Dauerschleife und wird mit Random erstmal initialisiert.
- kriginghotornot.pdf vorhanden !!!
- Erste Strukturen in der KHON Matrix zu erkennen
- HighFidelityEachMember 2: Sehr wenig HF Startsamples, deswegen nur geringes Vertrauen in die ersten Ersatzmodelle. Dadurch kann der Zusammenhang zwischen LF/HF besser eingeschätzt werden als er ist.

11.12.2016 (VPN)

- Kein Member konvergiert
- ProbOfImprovement auf 0.9
- 6 Restriktionen temporär ausgeschaltet, um mehr Variation zu ermöglichen

12.12.2016

- Fehler in der Lofi Prozesskette, die Tests der Lofi Prozesskette liefen immer auf einem Knoten, bei 2 Knoten gab es MPI Probleme. Behoben während der Laufzeit, Lofi sollte jetzt konvergieren.
- Ersatzmodelle sehen ansonsten gut aus.
- Cluster Sehr voll, deswegen nur wenig HF Rechnungen bisher.

13.12.2016

- 7 HF Member konvergiert und 3 LF
- 2 Restriktionen wieder hinzugenommen: Abstände zwischen den Moden

- Hyperparameter ändern sich noch stark, besonders die Varianzen der Kovarianzfunktionen und auch der Scale Faktor. Das spricht dafür, dass immer noch nicht genügend HF Daten zur Verfügung stehen

14.12.2016

- Ersatzmodelltraining Konvergenz niedriger und Iterationen/Startiterationen hoch, da keine Knoten mehr verfügbar.
- kriging_flow_14_260

20.12.2016

- Massentrom(0.01->0.03%)/Totaldruckverhältnis(0.01->0.03%)/Wirkungsgrad(0.05->0.06%) Lofi Komvergenzkriterium TRACE höher gesetzt, damit mehr konvergieren.
- Restriktionen 7592-7594 eingeschaltet
- 500->1000 Optimierungen auf dem Ersatzmodell
- ProImpro: 0.9 -> 0.85
- Krigin 407
- Restriktionen 7428&29 ; 7460 7471

21.12.2016

- ProbOfImpro von 0.,85 wird nie erreicht und bis Iteration 1000 auch noch bessere Member gefunden. -> 2000 Iterationen und ProbImprMin auf 0.7 gesenkt, damit noch das EVG optimiert werden kann

23.12.2016

- Vom 23.12 bis 25.12 ist das Kriging Training ausgefallen. Libs wurden auf dem Cluster geändert, das führte zum Absturz und die Exec. mussten neu kompiliert werden

10.01.2017

- 1 Woche keine konvergenten LF Member mehr, scheinbar Probleme in der Netzerzeugung
- Ultra Restart angeschaltet

12.01.2017

- Netzerzeuger PyMesh musste gegen eine neuere Version ausgetauscht werde
- Cluster wurde gewartet und komplett heruntergefahren

13.01.2017

- Cluster wurde hochgefahren, kein Member konvergiert

16.01.2017

- Der Permas Lizenz Server wurde nicht mit hochgefahren und wurde nach mehreren Anfragen nun gestartet

17.01.2017

- Lizenzserver schon wieder weg, wurde nochmal neu gestartet

20.01.2017

- Permas 15 wurde von T-Systems entfernt und Permas 16 installiert, ohne richtige Absprache. Optimierung deswegen wieder gestoppt
- Permas 16 war zudem nicht lauffähig

23.01.2017

- 143 HF Samples
- HFEachMember auf 2 gestellt

25.01.2017

- HFEach auf 8 gestellt
- 154 konvergierte HF Samples 603 LF

1.2.2017

- LF konvergiert nur zu 10%, gehen alle im Pumpgrenznahen Punkt kaputt. Betriebpunkt im LF wahrscheinlich verschoben. Sollte bei zukünftiger Opti beachtet werden. LF Betriebspunkt könnte grundsätzlich verschoben werden.

9.2.2017

- Run 2 in Planung, daher nur noch kurze Zeit laufen
- Vorzeigbares Ergebnis ist Hauptprio.
- Einige Member erfüllen die Restriktionen
- Hauptsächlich Wirkungsgraderhöhung ab jetzt
- Wirkungsgradrestriktion auf min 0.76, um gezielter zu optimieren

7.1 un 1

Beschreiben

Probleme erklären

7.2 Run 2

Änderungen zu Run1 erklären und dann Ergebnisse zeigen

8 Fazit und Ausblick

1. Grundlegende Zielsetzung, Entwicklung eines an industriell einsetzbaren und erweiterbaren Multifidelity Optimierungsverfahrens
   (a) Optimierungsverfahren stark ausgerichtet auf den Turbomaschinenbereich, bedeutet:
      i. Viele Restriktionen
      ii. mehrere Zielfunktionen
      iii. hochdimensionale Räume
      iv. numerisch sehr rechenaufwendige Prozessketten
      v. Prozessketten in verschiedenen Gütestufen
      vi. HPC Rechnerarchitektur mit GPUs,
   (b) Zukünftige Problemstellungn müssen auch bearbeitet werden können
      i. GEK-CO Kriging
      ii. CO-Kriging fließénde Gütestufen
2. Erreicht wird dies durch
   (a) Eigen entwickeltes CO-Kriging als Ersatzmodell
   (b) Erweiterung des bisherigen Optimierungsverfahren auf mehrere Fidelities, Entscheidungsfunktionen, Datenbasen usw.
   (c) Offenes Softwaredesign
      i. SVM müssen umgesetzt werden können
      ii. Andere Kriging Verfahren müssen umgesetzt werden können
      iii. Danach noch Erweiterbar sein
      iv. Objektorientierung
      v. Allgemeine Schnitstellen vom Ersatzmodell zum Optimierer
      vi. Unterschiedliche Programmiersprachen
   (d) Kriging Verfahren Hardwaremäßig
      i. CPU-GPU Mixbetrieb muss möglich sein für das Optimierungsverfahren

ii. Client / Server System, um Berechnungunge asynchron und dynamisch zu verteilen. Auch auf verschiedenen Architekturen (GPU/CPU)

8.0.1 Gradient Enhanced CO-Kriging

Sehr sinnvoll, da oftmals nur wenige HF Member bekannt sind und damit die Differenzkovarianzfunktion noch nicht gut geschätzt wird. Bekommt man dort nun noch partielle Ableitungen, könnte diese Schätzung sehr schnell erfolgen. Zudem hätte man die globale Raumabdeckung durch die LF Samples erreicht und eine sehr hohe Genauigkeit durch die partiellen Ableitungen.

Vielleicht nur in den Ausblick!!!!

A Anhang

A.1 Aerodynamische Größen

Totaldruckverlust:

$$\omega = \frac{\frac{1}{\dot{m}_2}\int p_{t2}^{is} - p_{t2}\mathrm{d}\dot{m}}{\frac{1}{\dot{m}_1}\int p_{t1} - p_1\mathrm{d}\dot{m}}$$

Wobei die Indizes sich auf die in Abbildung A.1 gezeigten Auswerteebenen vor und hinter dem Verdichtergitter beziehen. p_{t2} beschreibt den Totaldruck an der Austrittsebene und p_{t2}^{is} den Totaldruck bei isentroper Zustandsänderung. Der Massenstrom wird beschrieben durch die Variable $\dot{m}$.

A.2 Varianz der Fehlerfunktion

$$\mathrm{var}\left[F\left(\vec{x}\right)\right] = \mathrm{var}\left[Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right]$$

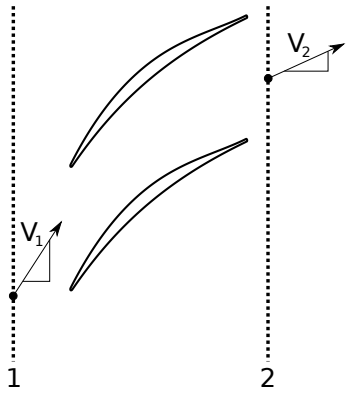


Abbildung A.1: Auswerteebenen eines exemplarischen Verdichtergitters

$$= E\left[\left(\left(Z_k(\vec{x}) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right) - E\left[Z_k(\vec{x}) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right]\right)^2\right]$$

$$= E\left[\left(Z_k(\vec{x}) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i - E\left[Z_k(\vec{x})\right] + E\left[\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right]\right)^2\right]$$

Im folgenden Schritt wird das Skalarprodukt als Summe formuliert:

$$= E\left[\left(\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right] - \left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} Z_i(\vec{x}_j)\, w_{i,j} - \sum_{i=1}^{s}\sum_{j=1}^{n_i} E\left[Z_i(\vec{x}_j)\right] w_{i,j}\right]\right)^2\right]$$

$$= E\left[\left(\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right] - \left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i(\vec{x}_j)\, w_{i,j} - E\left[Z_i(\vec{x}_j)\right] w_{i,j}\right)\right]\right)^2\right]$$

$$= E\left[\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right]^2 - 2\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i(\vec{x}_j)\, w_{i,j} - E\left[Z_i(\vec{x}_j)\right] w_{i,j}\right)\right]\right.$$
$$\left. + \left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i(\vec{x}_j)\, w_{i,j} - E\left[Z_i(\vec{x}_j)\right] w_{i,j}\right)\right]^2\right]$$

$$= E\left[\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right]^2\right] - 2E\left[\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i(\vec{x}_j)\, w_{i,j} - E\left[Z_i(\vec{x}_j)\, w_{i,j}\right]\right)\right]\right]$$
$$+ E\left[\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i(\vec{x}_j)\, w_{i,j} - E\left[Z_i(\vec{x}_j)\, w_{i,j}\right]\right)\right]^2\right]$$

$$= \mathrm{var}\left[Z_k(\vec{x})\right] - 2E\left[\left[Z_k(\vec{x}) - E\left[Z_k(\vec{x})\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i(\vec{x}_j)\, w_{i,j} - E\left[Z_i(\vec{x}_j)\, w_{i,j}\right]\right)\right]\right]$$
$$+ E\left[\left[\sum_{i=1}^{s} \left(\vec{Z}_i^T \vec{w}_i - E\left[Z_i\right] \vec{1^T} \vec{w}_i\right)\right]^2\right]$$

$$= \text{var}\left[Z_k\left(\vec{x}\right)\right] - 2E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]\right]$$
$$+ E\left[\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\left(Z_k\left(\vec{x}_l\right)w_{k,l} - E\left[Z_k\left(\vec{x}_l\right)w_{k,l}\right]\right)\right]\right]$$

$$= \text{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i}E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]$$
$$+ \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}E\left[\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\left(Z_k\left(\vec{x}_l\right)w_{k,l} - E\left[Z_k\left(\vec{x}_l\right)w_{k,l}\right]\right)\right]$$

$$= \text{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i}\text{cov}\left(Z_k\left(\vec{x}\right), Z_i\left(\vec{x}_j\right)w_{i,j}\right)$$
$$+ \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}\text{cov}\left(Z_i\left(\vec{x}_j\right)w_{i,j}, Z_k\left(\vec{x}_l\right)w_{k,l}\right)$$

$$= \text{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i}w_{i,j}\text{cov}\left(Z_k\left(\vec{x}\right), Z_i\left(\vec{x}_j\right)\right)$$
$$+ \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}w_{i,j}w_{k,l}\text{cov}\left(Z_i\left(\vec{x}_j\right), Z_k\left(\vec{x}_l\right)\right)$$

In Matrix-Schreibweise ergibt sich die folgende Formulierung, wobei $\vec{w} \in \mathbb{R}^{n_{all}}$ den Gewichtsvektor, $\mathbf{Cov} \in \mathbb{R}^{n_{all} \times n_{all}}$ die Kovarianzmatrix und $\overrightarrow{\text{cov}} \in \mathbb{R}^{n_{all}}$ den Kovarianzvektor darstellt:

$$\text{var}\left[F\left(\vec{x}_0\right)\right] = \text{var}\left[Z\left(\vec{x}_0\right)\right] - 2\overrightarrow{\text{cov}}\left(Z\left(\vec{x}_0\right), Z\left(\vec{x}\right)\right)^T * \vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \tag{A.1}$$

A.3 Kovarianz zwischen Vorhersagen

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{w}_1^T\mathbf{Cov}\vec{w}_2 \tag{A.2}$$

Die Gewichtsvektoren sehen folgendermaßen aus:

$$\vec{w_1} = \mathbf{Cov}^{-1}\vec{c_1} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_1} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

$$\vec{w_2} = \mathbf{Cov}^{-1}\vec{c_2} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_2} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

Eingesetzt in die ursprüngliche Formel A.2:

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{w_1}^T\mathbf{Cov}\vec{w_2}$$

$$\Leftrightarrow \left(\mathbf{Cov}^{-1}\vec{c_1} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_1} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)^T * \mathbf{Cov}\left(\mathbf{Cov}^{-1}\vec{c_2} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_2} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\Leftrightarrow \left(\mathbf{Cov}^{-1}\vec{c_1} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_1} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)^T \left(\vec{c_2} - \frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_2} + \frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\Leftrightarrow \left(\vec{c_1}^T\mathbf{Cov}^{-1} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} + \frac{\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right) \left(\vec{c_2} - \frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_2} + \frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\begin{aligned}\Leftrightarrow \vec{c_1}^T\mathbf{Cov}^{-1}\vec{c_2} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} + \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} \\ + \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k \\ + \frac{\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k - \frac{\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c_2}\beta_k + \frac{\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k^2\end{aligned}$$

$$\begin{aligned}\Leftrightarrow \vec{c_1}^T\mathbf{Cov}^{-1}\vec{c_2} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} + \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} \\ + \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k \\ + \frac{\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k - \frac{\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \frac{\beta_k^2}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\end{aligned}$$

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{c_1}^T\mathbf{Cov}^{-1}\vec{c_2} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} + \frac{\beta_k^2}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}$$

A.4 Varianz des Schätzfehlers

$$\vec{w} = \mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

$$\text{var}\left[F\left(\vec{x}\right)\right] = \text{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathbf{c}}^T\vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w}$$

$$\begin{aligned}\text{var}\left[F\left(\vec{x}_0\right)\right] = \text{var}\left[Z\left(\vec{x}_0\right)\right] - 2\vec{c}^T &\left(\mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right) \\ &+ \left(\mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)^T \\ &\mathbf{Cov}\left(\mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)\end{aligned}$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - 2\vec{c}^T\mathbf{Cov}^{-1}\vec{c} + 2\vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} - 2\vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(\vec{c}^T\mathbf{Cov}^{-1} - \vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\beta_k\right)\left(\vec{c} - \frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - 2\vec{c}^T\mathbf{Cov}^{-1}\vec{c} + 2\vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} - 2\vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \vec{c}^T\mathbf{Cov}^{-1}\vec{c} - \frac{\vec{c}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\vec{c}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(-\vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\beta_k\right)\left(\vec{c} - \frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} + \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} - \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(-\vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\beta_k\right)\left(\vec{c} - \frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} - \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} - \vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\beta_k\left(\vec{c} - \frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k\right)$$

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} - \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c}$$
$$-\vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\vec{c}\beta_k$$
$$-\left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c}\beta_k + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k^2$$

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} - \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\vec{c}\beta_k$$
$$+\vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} - \vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$
$$-\left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c}\beta_k + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k^2$$

Zwischenrechnung:

$$\vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k = \frac{\vec{c}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k = \frac{\vec{c}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

Da der Nenner und der Zähler ein Skalar ergeben, gilt:

$$\frac{\vec{c}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k = \left(\frac{\vec{c}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\beta_k = \frac{\vec{c}\mathbf{Cov}^{-1}\vec{F}^T}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k$$

Eingesetzt in die ursprüngliche Gleichung:

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} + \vec{c}^T\left(\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c}$$
$$-2\vec{c}\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \left(\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\right)^T\frac{\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k^2$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} + \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} \\ -2\vec{c}\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \frac{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k^2$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} + \vec{c}^T\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\vec{c} - 2\vec{c}\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\beta_k + \frac{\beta_k^2}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}$$

$$\Leftrightarrow \text{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} + \frac{1}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\left(\vec{c}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c} - 2\vec{c}\mathbf{Cov}^{-1}\vec{F}\beta_k + \beta_k^2\right)$$

Bei dem Term in der Klammer handelt es sich nur um Skalare und damit entspricht dies einer quadratischen Gleichung, also gilt:

$$\text{var}\left[F\left(\vec{x}_0\right)\right] = \text{var}\left[Z\left(\vec{x}_0\right)\right] - \vec{c}^T\mathbf{Cov}^{-1}\vec{c} + \frac{1}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}\left(\vec{c}^T\mathbf{Cov}^{-1}\vec{F} - \beta_k\right)^2$$

A.5 Zusammenhang Kovarianz und Fehler Varianz

$$cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right) = \vec{w_1}^T\mathbf{Cov}\vec{w}_2 = \vec{c_1}^T\mathbf{Cov}^{-1}\vec{c_2} - \frac{\vec{c_1}^T\mathbf{Cov}^{-1}\vec{F}\vec{F}^T\mathbf{Cov}^{-1}\vec{c_2}}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}} + \frac{\beta_k^2}{\vec{F}^T\mathbf{Cov}^{-1}\vec{F}}$$

$$\text{var}\left[Z_k^*\left(\vec{x}\right)\right] = cov\left(Z_k^*\left(\vec{x}\right), Z_k^*\left(\vec{x}\right)\right) = \vec{w}^T\mathbf{Cov}\vec{w}$$

Die Fehlervarianz setzt sich wie folgt zusammen:

$$\text{var}\left[F\left(\vec{x}\right)\right] = \text{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathbf{c}}^T\vec{w} + \vec{w}^T\mathbf{Cov}\vec{w}$$

Der letzte Term kann durch die Kovarianzformel $cov\left(Z_k^*\left(\vec{x_1}\right), Z_k^*\left(\vec{x_2}\right)\right)$ ersetzt werden:

$$\text{var}\left[F\left(\vec{x}\right)\right] = \text{var}\left[Z\left(\vec{x}\right)\right] - 2\overrightarrow{\mathbf{c}}^T\vec{w} + cov\left(Z_k^*\left(\vec{x}\right), Z_k^*\left(\vec{x}\right)\right)$$

Formuliert man die Gewichte und Kovarianz vollständig aus:

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}\right)\right] - 2\vec{c}^{T}\left(\mathbf{Cov}^{-1}\vec{c} - \frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^{T}\mathbf{Cov}^{-1}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\vec{c} + \frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\beta_k\right) + \vec{c}^{T}\mathbf{Cov}^{-1}\vec{c} - \frac{\vec{c}^{T}\mathbf{Cov}^{-1}\vec{F}\vec{F}^{T}\mathbf{Cov}^{-1}\vec{c}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}} + \frac{\beta_k^2}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}$$

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}\right)\right] - 2\vec{c}^{T}\mathbf{Cov}^{-1}\vec{c} + 2\vec{c}^{T}\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^{T}\mathbf{Cov}^{-1}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\vec{c} - 2\vec{c}^{T}\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\beta_k + \vec{c}^{T}\mathbf{Cov}^{-1}\vec{c} - \frac{\vec{c}^{T}\mathbf{Cov}^{-1}\vec{F}\vec{F}^{T}\mathbf{Cov}^{-1}\vec{c}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}} + \frac{\beta_k^2}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}$$

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}\right)\right] - \vec{c}^{T}\mathbf{Cov}^{-1}\vec{c} + \vec{c}^{T}\frac{\mathbf{Cov}^{-1}\vec{F}\vec{F}^{T}\mathbf{Cov}^{-1}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\vec{c} - 2\vec{c}^{T}\frac{\mathbf{Cov}^{-1}\vec{F}}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\beta_k + \frac{\beta_k^2}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}$$

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}\right)\right] - \vec{c}^{T}\mathbf{Cov}^{-1}\vec{c} + \frac{1}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\left(\vec{c}^{T}\mathbf{Cov}^{-1}\vec{F}\vec{F}^{T}\mathbf{Cov}^{-1}\vec{c} - 2\vec{c}^{T}\mathbf{Cov}^{-1}\vec{F}\beta_k + \beta_k^2\right)$$

$$\Leftrightarrow \operatorname{var}\left[Z\left(\vec{x}\right)\right] - \vec{c}^{T}\mathbf{Cov}^{-1}\vec{c} + \frac{1}{\vec{F}^{T}\mathbf{Cov}^{-1}\vec{F}}\left(\vec{c}^{T}\mathbf{Cov}^{-1}\vec{F} - \beta_k^2\right)$$

Dieses Ergebnis entspricht der Formel für $\operatorname{var}\left[F\left(\vec{x}\right)\right]$ vollständig und damit ist der Zusammenhang zwischen Kovarianz und Fehler Varianz hergestellt.

A.6 Likelihood Schätzer Varianzen im CO-Kriging

$$\mathbf{Cov} = \begin{bmatrix} a^2\sigma_{low}^2 corr_{low} + \sigma_{err}^2 corr_{err} & a\sigma_{low}^2 corr_{low} \\ a\sigma_{low}^2 corr_{low} & \sigma_{low}^2 corr_{low} \end{bmatrix}$$

$$\mathbf{Cov} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\mathbf{Cov} = \begin{bmatrix} a^2\sigma_{low}^2 C_{11A} + a^2\sigma_{err}^2 C_{11B} & a\sigma_{low}^2 C_{12} \\ a\sigma_{low}^2 C_{21} & \sigma_{low}^2 C_{22} \end{bmatrix}$$

$$\mathbf{Cov^{-1}} = \begin{bmatrix} \left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} & -\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1} \\ -C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} & C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1} \end{bmatrix}$$

Oben links:

$$\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} = \left(\left(a^2\sigma_{low}^2 C_{11A} + a^2\sigma_{err}^2 C_{11B}\right) - a\sigma_{low}^2 C_{12}\left(\sigma_{low}^2 C_{22}\right)^{-1} a\sigma_{low}^2 C_{12}\right)^{-1}$$

$$= \left(a^2\left(\sigma_{low}^2 C_{11A} + \sigma_{err}^2 C_{11B}\right) - a^2 C_{12}\left(C_{22}\right)^{-1}\sigma_{low}^2 C_{12}\right)^{-1}$$

$$= \left(a^2\left(\sigma_{low}^2 C_{11A} + \sigma_{err}^2 C_{11B} - \sigma_{low}^2 C_{12}\left(C_{22}\right)^{-1} C_{12}\right)\right)^{-1}$$

$$\mathbf{Cov^{-1}}\vec{G} = \begin{bmatrix} \left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} & -\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1} \\ -C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} & C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1} \end{bmatrix} \begin{bmatrix} \vec{1} & \vec{0} \\ \vec{0} & \vec{1} \end{bmatrix}$$

$$\mathbf{Cov^{-1}}\vec{G} = \begin{bmatrix} \left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & -\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\vec{1} \\ -C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & \left(C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\right)\vec{1} \end{bmatrix}$$

$$\vec{G}^T\mathbf{Cov^{-1}}\vec{G} = \begin{bmatrix} \vec{1}^T\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & -\vec{1}^T\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\vec{1} \\ -\vec{1}^T C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & \vec{1}^T\left(C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\right. \end{bmatrix}$$

Vorausgesetzt man kennt die Low Varianz bereits:

A.7 Korrelationsmatrix oder Kovarianzmatrix

In der einschlägigen Literatur wird die Kovarianzmatrix oftmals durch eine Korrelationsmatrix ersetzt,

Die normierte Kovarianz ist die Korrelation und der Zusammenhang zwischen Kovarianz und Korrelation sieht wie folgt aus:

$$c\left(X,Y\right)=\frac{\mathrm{cov}\left(X,Y\right)}{\sqrt{\mathrm{var}\left(X\right)}\sqrt{\mathrm{var}\left(Y\right)}} \tag{A.3}$$

Zudem soll der räumliche Zusammenhang zwischen zwei Zufallsvariablen nicht von seiner absoluten Lage im Raum abhängig sein, sondern nur von deren Abstandsvektor $h\in\mathbb{R}^{\mathrm{n}}=\{x_1-y_1,...,x_n-y_n\}$

Die Kovarianz zwischen zwei gleichen Zufallsvariablen ist per Definition die Varianz dieser Zufallsvariable:

$$\mathrm{cov}\left(Z,Z\right)=\mathrm{var}\left(Z\right)$$

Da die Kovarianz allerdings nur vom Abstandsvektor abhängig ist und der Abstandsvektor bei zwei gleichen Zufallsvariablen immer $\vec{0}$ ist, kann man daraus folgern, dass die Kovarianz zwischen zwei gleichen Zufallsvariablen immer konstant ist.

$$\mathrm{cov}\left(Z,Z\right)=\mathrm{cov}\left(h=\vec{0}\right)=const.$$

Und somit ist auch die Varianz konstant:

$$\Longrightarrow\mathrm{var}\left[Z\right]=\sigma^2=const. \tag{A.4}$$

Durch die Gleichungen A.3 und A.4 kann man die Kovarianzfunktion durch die Korrelationsfunktion ersetzen

$$c\left(X,Y\right)\sigma^2=\mathrm{cov}\left(X,Y\right)$$

Und somit auch die Kovarianzmatrix durch die Korrelationsmatrix $\mathbf{R}\in\mathbb{R}^{\mathrm{n}\times\mathrm{n}}$

$$\mathbf{Cov} = \sigma^2 \mathbf{R}$$

Damit lässt sich Gleichung **??** umformulieren, wobei der Korrelationsvektor mit $\vec{r} \in \mathbb{R}^{\mathrm{n}} = \left(c\left[Z\left(\vec{x}_0\right), Z\left(\vec{x}_1\right)\right], ..., c\left[Z\left(\vec{x}_0\right), Z\left(\vec{x}_n\right)\right]\right)^T, \vec{x} \in \mathbb{R}^{\mathrm{k}}$ bezeichnet wird. Dieser beschreibt die Korrelationen zwischen dem Funktionswert, der vorhersagt werden soll und dem Stützstellenvektor $\vec{y}_s$. Da der Funktionswert $y\left(x_0\right)$ nicht bekannt ist, hängt die Korrelationsfunktion nur von dem Abstandsvektor $\vec{h} \in \mathbb{R}^{\mathrm{k}} = \{x_0 - x_1, ..., x_k - x_k\}$ ab. Der Korrelationsvektor ergibt sich damit zu $\vec{r} \in \mathbb{R}^{\mathrm{n}} = \left(c\left[\vec{x}_0, \vec{x}_1\right], ..., c\left[\vec{x}_0, \vec{x}_n\right]\right)^T, \vec{x} \in \mathbb{R}^{\mathrm{k}}$. Wie eine solche Korrelationsfunktion konkret umgesetzt wird, folgt in Kaptitel **??**.

A.8 Vereinfachung der Vorhersagegleichung

An dieser Stelle stellt sich die Frage, inwiefern μ und der Likelihood Schätzer für $\vec{\tilde{\beta}} \in \mathbb{R}^{1 \times \mathrm{m_f}}$ zusammenhängen. Der Likelihood Schätzer für $\vec{\tilde{\beta}} \in \mathbb{R}^{1 \times \mathrm{m_f}}$ sieht wie folgt aus:

$$\left(G^T \mathbf{Cov}^{-1} G\right)^{-1} \left(G^T \mathbf{Cov}^{-1} \overrightarrow{y}\right) = \vec{\tilde{\beta}}$$

Der Vektor $\vec{F} \in \mathbb{R}^{\mathrm{n} \times 1}$ lässt sich auch umformulieren zu, wobei $G \in \mathbb{R}^{\mathrm{n} \times \mathrm{m_f}}$:

$$\vec{F} = \begin{bmatrix} \vec{\beta}_1 \\ \vec{\beta}_2 \end{bmatrix} = G\vec{\tilde{\beta}} = G\left(G^T \mathbf{Cov}^{-1} G\right)^{-1} \left(G^T \mathbf{Cov}^{-1} \overrightarrow{y}\right)$$

$$\mu = \frac{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} \vec{y}}{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} G\vec{\tilde{\beta}}}$$

$$\mu = \frac{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} \vec{y}}{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} G\vec{\tilde{\beta}}}$$

$$\mu = \frac{\left(G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)\right)^T\mathbf{Cov}^{-1}\vec{y}}{\left(G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)\right)^T\mathbf{Cov}^{-1}G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)}$$

$$\mu = \frac{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}G^T\mathbf{Cov}^{-1}\vec{y}}{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}G^T\mathbf{Cov}^{-1}G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)}$$

$$\mu = \frac{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}G^T\mathbf{Cov}^{-1}\vec{y}}{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)}$$

Daraus folgt letztlich:

$$\mu = 1$$

A.9 Beschleunigter Rückwärtsdifferenzierter-Cholesky-Algorithmus nach Smith

Innerhalb dieses Abschnittes soll eine Variante des Rückwärtsdifferenzierten-Cholesky-Algorithmus nach Smith [Smith, 1995] gezeigt werden. Diese Variante beinhaltet im Vergleich zum Original noch SIMD-Beschleunigungen und eine Parallelisierung auf Thread Ebene.

```
1  int num_threads;
2  #pragma omp parallel { num_threads= omp_get_num_threads(); }
3  int n = this->getColumnSize();
4  if(this->getColumnSize()!=adjointMat.getRowSize() or adjointMat.getRowSi
5      adjointMat = OpenMPMatrix<T>(n,n);
6  }
7  T* adjointArray = adjointMat.getArray();
8  T* thisArray = (*this).getArray();
9  T parallelTmp[num_threads*n];
10 T diag[n];
11 T inverseDiag[n];
12 T diagAdjoint[n];
```

```
13 T thisArrayLine[n];
14 __m128d diagSSE;
15 __m128d *diagAdjointSSE=(__m128d *)diagAdjoint;
16 __m128d *inverseDiagSSE=(__m128d *)inverseDiag;
17 __m128d bufSSE;
18 __m128d buf2SSE;
19 __m128d buf3SSE;
20 __m128d twoSSE = _mm_set_pd(2.0,2.0);
21 __m128d oneSSE = _mm_set_pd(1.0,1.0);
22 double tmpSSE;
23 for(int i=0; i<n/2; i++){
24     int iD = i*2;
25     diag[iD] = thisArray[iD*n+iD];
26     diag[iD+1] = thisArray[(iD+1)*n+iD+1];
27     diagSSE = _mm_load_pd(&(diag[iD]) );
28     inverseDiagSSE[i] = _mm_div_pd(oneSSE,diagSSE);
29     diagAdjointSSE[i] = _mm_mul_pd(inverseDiagSSE[i],twoSSE);
30     }
31 if(n%2>0){
32         int iD = n-1;
33         diag[iD] = thisArray[iD*n+iD];
34         inverseDiag[iD] = 1.0/diag[iD];
35         diagAdjoint[iD]= 2.0*inverseDiag[iD];
36     }
37 for(int k=n-1; k>=0; k--){
38         A_memmove(thisArrayLine, &(thisArray[k*n]),n*sizeof(T));
39         int jStart=k+1;
40         if(k*n%2==0){
41             // dafür sorgen, dass j in der Schleife immer gerade ist, um
42             if(jStart%2>0 and jStart < n){
43                 adjointArray[k*n+jStart] -=
   2.0*diagAdjoint[jStart] * thisArrayLine[jStart];
44                 jStart++;
45             }
46
47             for(int j=jStart; j<n-1;j+=2){
48                 bufSSE =  _mm_load_pd(&(diagAdjoint[j]));
49                 buf2SSE =  _mm_load_pd(&(thisArray[k*n+j]));
50                 buf3SSE =  _mm_load_pd(&(adjointArray[k*n+j]));
```

```
51                  _mm_store_pd( &(adjointArray[k*n+j]) , _mm_sub_pd(buf3SS
52              }
53              if( (n-(jStart)) % 2 > 0){
54                  adjointArray[k*n+(n-1)] -=
    2.0*diagAdjoint[(n-1)] * thisArrayLine[(n-1)];
55              }
56          }
57          else{
58              for(int j=jStart; j<n;j++){
59                  adjointArray[k*n+j] -=  2.0*diagAdjoint[j] * thisArrayLi
60              }
61          }
62          if((n-k)/num_threads > 10 ){
63              for(int i=0; i<num_threads;i++)
64                  A_memset(&(parallelTmp[i*n +k+2]), 0.0, (n-(k+2))*sizeof
65              #pragma omp parallel for
66              for(int j=k+1; j<n;j++){
67                  int num_thread = omp_get_thread_num();
68                  for(int i=j+1; i<n;i++){
69                      parallelTmp[num_thread*n+i] -=
    adjointArray[j*n+i]* thisArrayLine[j];
70                  }
71              }
72              #pragma omp parallel for
73              for(int j=k+2; j<n;j++){
74                      for(int i=0; i<num_threads;i++){
75                          adjointArray[k*n+j] += parallelTmp[i*n+j];
76                      }
77              }
78          }
79          else{
80              for(int j=k+1; j<n;j++){
81                  //int num_thread = omp_get_thread_num();
82                  for(int i=j+1; i<n;i++){
83                      adjointArray[k*n+i] -=
    adjointArray[j*n+i]* thisArrayLine[j];
84
85                  }
86              }
```

```
87          }
88          if((n-k)/num_threads > 4 ){
89              #pragma omp parallel for private(bufSSE, buf2SSE)
90              for(int j=k+1; j<n;j++){
91                  if(j*n%2>0){
92                      for(int i=j+1; i<n;i++){
93                          adjointArray[k*n+j] -=
    adjointArray[j*n+i] * thisArrayLine[i];
94                      }
95                  }
96                  else{//SSE
97                      __m128d ress = _mm_setzero_pd();
98                      int iStart=(j+1);
99                      double tmp=0.0;
100                     if(iStart%2 >0){
101                         adjointArray[k*n+j] -=
    adjointArray[j*n+iStart] * thisArrayLine[iStart];
102                         iStart++;
103                     }
104                     for(int i=iStart; i<n-1;i+=2){
105                         bufSSE =  _mm_load_pd(&(thisArrayLine[i]));
106                         buf2SSE =  _mm_load_pd(&(adjointArray[j*n+i]));
107                         ress = _mm_sub_pd(ress,_mm_mul_pd(bufSSE,buf2SSE
108                     }
109                     ress = _mm_hadd_pd(ress,ress);
110                     _mm_store_sd(&tmp, ress);
111                     adjointArray[k*n+j]=adjointArray[k*n+j]+tmp;
112                     if((n-iStart)%2 >0){
113                         adjointArray[k*n+j] -=
    adjointArray[j*n+(n-1)] * thisArrayLine[(n-1)];
114                     }
115                 }
116             }
117         }
118         else{
119             for(int j=k+1; j<n;j++){
120                 for(int i=j+1; i<n;i++){
121                     adjointArray[k*n+j] -=
    adjointArray[j*n+i] * thisArrayLine[i];
```

```
122                 }
123                             }
124         }
125         buf3SSE = _mm_set_pd1(inverseDiag[k]);
126         __m128d ress = _mm_setzero_pd();
127         jStart=(k*n)+(k+1);
128         if(jStart%2>0 and jStart < n+k*n -1){
129             adjointArray[jStart] *=  inverseDiag[k];
130             diagAdjoint[k] -=  adjointArray[jStart]*thisArray[jStart];
131                 jStart++;
132                         }
133         for(int j=jStart; j<n+k*n -1;j+=2){
134             bufSSE =  _mm_load_pd(&(thisArray[j]));
135             buf2SSE = _mm_load_pd(&(adjointArray[j]));
136             _mm_store_pd(&(adjointArray[j]), _mm_mul_pd(buf2SSE,buf3SSE)
137             buf2SSE = _mm_load_pd(&(adjointArray[j]));
138             ress = _mm_sub_pd(ress, _mm_mul_pd(buf2SSE,bufSSE) );
139         }
140         ress = _mm_hadd_pd(ress,ress);
141         _mm_store_sd(&tmpSSE, ress);
142         diagAdjoint[k] = diagAdjoint[k] + tmpSSE;
143         if(((n+k*n)-jStart)%2>0){
144             adjointArray[n+k*n -1] *= inverseDiag[k];
145             diagAdjoint[k] -=  adjointArray[n+k*n -1]*thisArray[n+k*n -
146         }       diagAdjoint[k]=0.5*diagAdjoint[k]*
inverseDiag[k];
147     }
148     for(int i=0; i<n; i++){
149         adjointArray[i*n+i] =diagAdjoint[i] ;
150     }
```

A.10 Alternative Herleitung der symbolischen Differenzierung

$$\frac{\partial\left(ln\left(det\left(\mathbf{Cov}\left(h_l\right)\right)\right)\right)}{\partial h_l} = \text{Spur}\left(\mathbf{Cov}^{-1}\frac{\partial \mathbf{Cov}}{\partial h_l}\right)$$

$$\phi_{ij}^{L}\left(A\right)=\begin{cases}A_{ij} & i>j\\ 0.5A_{ij} & i=j\\ 0 & i<j\end{cases}$$

$$\phi_{ij}^{U}\left(A\right)=\begin{cases}0 & i>j\\ 0.5A_{ij} & i=j\\ A_{ij} & i<j\end{cases}$$

$$\text{Spur}\left(\mathbf{Cov}^{-1}\frac{\partial\mathbf{Cov}}{\partial h_l}\right)=\text{Spur}\left(\left(\phi^{U}\left(\mathbf{Cov}^{-1}\right)+\phi^{L}\left(\mathbf{Cov}^{-1}\right)\right)\frac{\partial\mathbf{Cov}}{\partial h_l}\right)$$

$$=\text{Spur}\left(\phi^{U}\left(\mathbf{Cov}^{-1}\right)\frac{\partial\mathbf{Cov}}{\partial h_l}+\phi^{L}\left(\mathbf{Cov}^{-1}\right)\frac{\partial\mathbf{Cov}}{\partial h_l}\right)$$

$$=\text{Spur}\left(\phi^{U}\left(\mathbf{Cov}^{-1}\right)\frac{\partial\mathbf{Cov}}{\partial h_l}\right)+\text{Spur}\left(\phi^{L}\left(\mathbf{Cov}^{-1}\right)\frac{\partial\mathbf{Cov}}{\partial h_l}\right)$$

Da $\mathbf{Cov}^{-1}$ und $\frac{\partial\mathbf{Cov}}{\partial h_l}$ symmetrisch sind gilt:

$$=2\text{Spur}\left(\phi^{L}\left(\mathbf{Cov}^{-1}\right)\frac{\partial\mathbf{Cov}}{\partial h_l}\right)$$

Dies ist nun exakt dasselbe Ergebnis der symbolischen Differenzierung.

A.11 Wann würde eine globales Varianzkriterium zu einer anderen Entscheidung führen als das lokale?

Testfunktion, in der die Hauptinformation bereits im Low-Fidelity Modell enthalten ist:

$$f_{high}=2x+0.1\sin\left(20x\right)$$

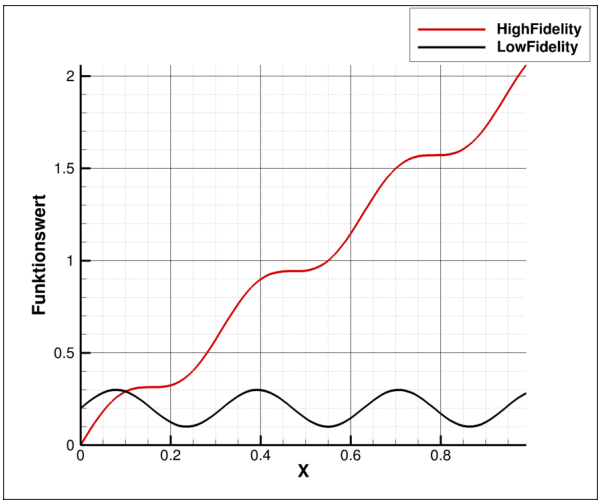


Abbildung A.2:

Samples	Fehlerintegral (x über 0-1)	Dev Integral	Dev lokal x=0.35	
Original	0.000183472470246	0.000295987939791	0.0002337432566266	
+HF	0.000115168572795	0.000159359003548	0.0000000298023224	
+LF	5.96393583479e-05	0.000101191138483	0.0001315242093663	

Tabelle A.1:

$$f_{low} = 0.1 \sin(20x) + 0.2$$

Die Vorhersage mit 11 High-Fidelity und 7 Low-Fidelity Samples:

Dieselbe Vorhersage mit einem zusätzlichen HF Sample an der Stelle x=0.35

Die Vorhersage mit 11 High-Fidelity und 7 Low-Fidelity Samples:

Die Vorhersage mit 11 High-Fidelity und 7 Low-Fidelity Samples:

Die Vorhersage mit 11 High-Fidelity und 7 Low-Fidelity Samples:

In diesem Fall würden das lokale und globale Kriterium zu einer anderen Entscheidung führen.

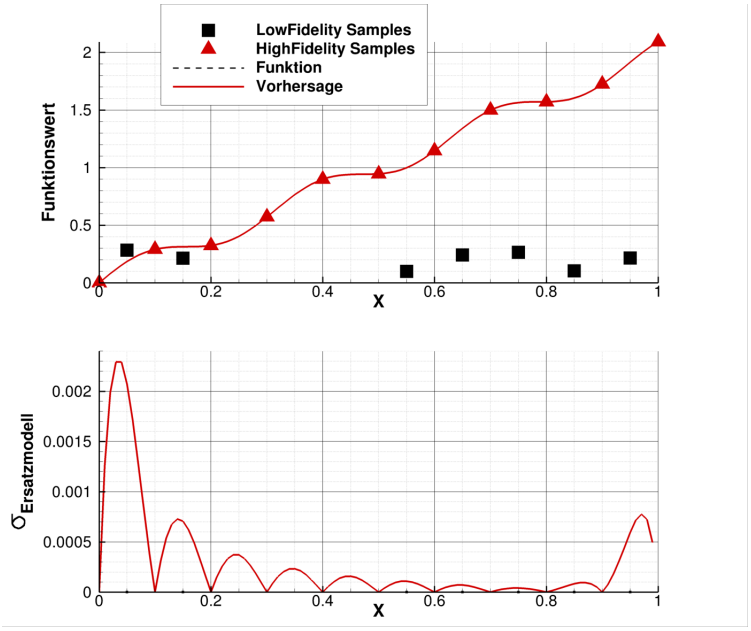


Abbildung A.3:

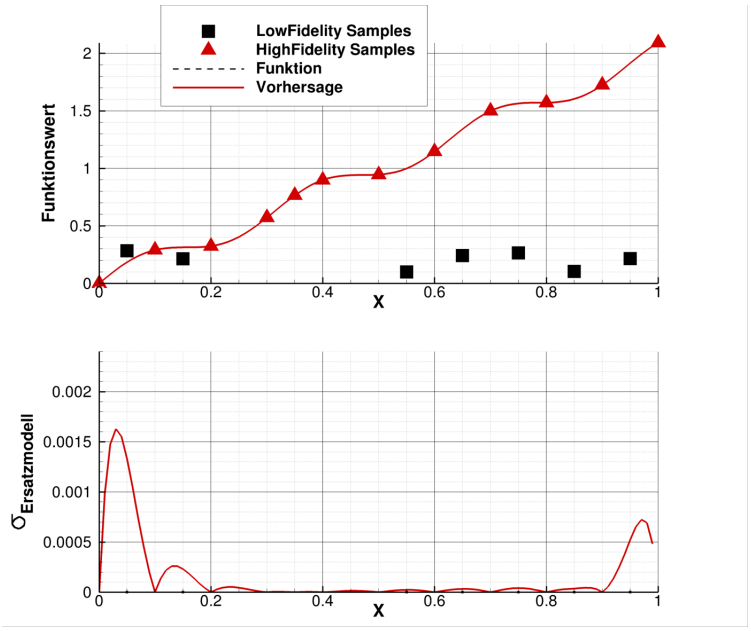


Abbildung A.4:

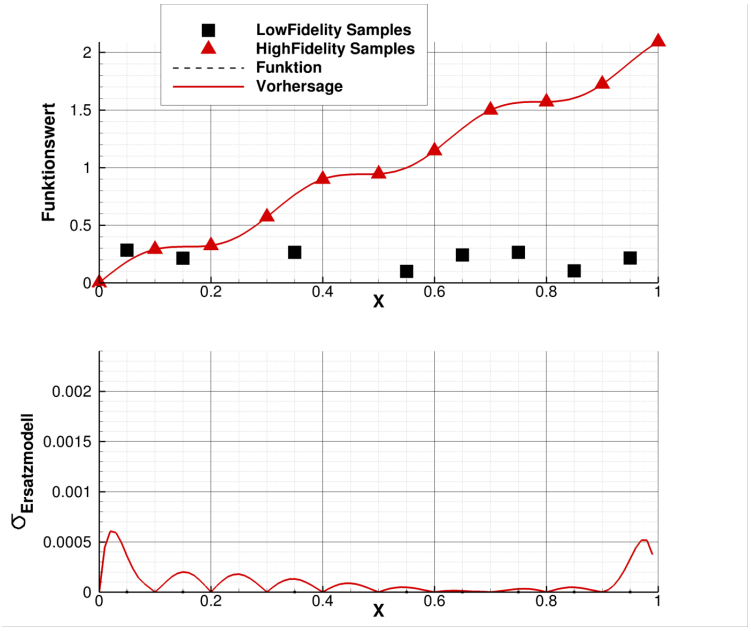


Abbildung A.5:

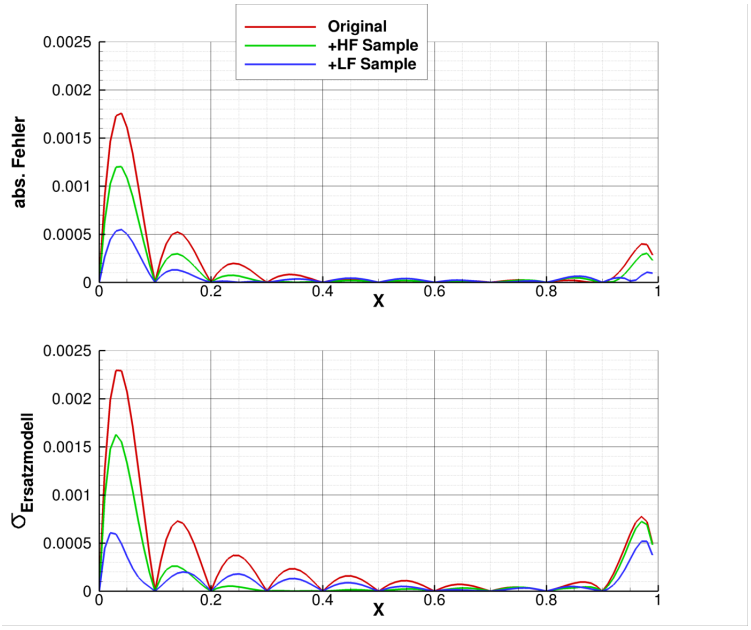


Abbildung A.6:

Literaturverzeichnis

[Adams et al., 2016] Adams, B. M., Mohamed S., E., Eldred, M. S., Geraci, G., Jakeman, J. D., Maupin, K. A., Monschke, J. A., Swiler, L. P., Stephens, J. A., Vigil, D. M., & Wildey, T. M. (2016). Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.5 Theory Manual.

[Aulich & Siller, 2011] Aulich, M. & Siller, U. (2011). High-Dimensional Constrained Multiobjective Optimization of a Fan Stage. In *ASME GT2011-45618*.

[Avron & Toledo, 2011] Avron, H. & Toledo, S. (2011). Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM*, 58(2), 1–34.

[Backhaus et al., 2012] Backhaus, J., Aulich, M., Frey, C., Lengyel, T., & Voß, C. (2012). Gradient Enhanced Surrogate Models Based on Adjoint CFD Methods for the Design of a Counter Rotating Turbofan. *ASME Turbo Expo*.

[Backhaus et al., 2017] Backhaus, J., Schmitz, A., Frey, C., Mann, S., Nagel, M., Sagebaum, M., & Gauger, N. R. (2017). Application of an Algorithmically Differentiated Turbomachinery Flow Solver to the Optimization of a Fan Stage. (pp. 1–20).

[Baert et al., 2015] Baert, L., Beauthier, C., Leborgne, M., & Lepot, I. (2015). Surrogate-Based Optimisation for a Mixed-Variable Design Space: Proof of Concept and Opportunities for Turbomachinery Applications. In *Volume 2C: Turbomachinery* (pp. V02CT45A015).: ASME.

[Bellman, 1972] Bellman, R. (1972). *Dynamic programming.* Princeton University Press.

[Box & Muller, 1958] Box, G. E. P. & Muller, M. E. (1958). A note on the generation of random normal deviates. *The annals of mathematical statistics*, (29), 610–611.

[Bronstejn & Semendjajew, 2008] Bronstejn, I. N. & Semendjajew, K. A. (2008). *Taschenbuch der Mathematik*. Harri Deutsch.

[Brooks et al., 2011] Brooks, C. J., Forrester, A. I. J., Keane, A. J., & Shahpar, S. (2011). MULTI-FIDELITY DESIGN OPTIMISATION OF A TRANSONIC COMPRESSOR ROTOR. *9th European Conf. Turbomachinery Fluid Dynamics and Thermodynamics, Istanbul, Turkey*.

[Chahine et al., 2012] Chahine, C., Seume, J. R., & Verstraete, T. (2012). The Influence of Metamodeling Techniques on the Multidisciplinary Design Optimization of a Radial Compressor Impeller. In *Volume 8: Turbomachinery, Parts A, B, and C* (pp. 1951).: ASME.

[Cook, 2012] Cook, S. (2012). *CUDA Programming*. Morgan Kaufmann.

[Domercq, 2006] Domercq, O. (2006). Advances in Axial Compressor Aerodynamics. *Lecture Series 2006-06*, (pp.̃38).

[Douglas, 2004] Douglas, G. (2004). Boost Function.

[Drela & Youngren, 2008] Drela, M. & Youngren, H. (2008). A User's Guide to MISES 2.63.

[Dworak et al., 2011] Dworak, A., Ehm, F., Sliwinski, W., & Sobczak, M. (2011). MIDDLEWARE TRENDS AND MARKET LEADERS 2011. *CERN*.

[Eifinger, 2013] Eifinger, P. (2013). Validierung eines Gradient Enhanced Kriging Verfahrens anhand einer Parameterstudie am Profilschnitt eines Fans. *Projektarbeit*.

[Elfert et al., 2016] Elfert, M., Weber, A., Wittrock, D., Peters, A., Voss, C., & Nicke, E. (2016). Experimental and Numerical Verification of An Optimization of a Fast Rotating High Performance Radial Compressor Impeller. *ASME Turbo Expo*, (pp. 1–12).

[Fog, 2013] Fog, A. (2013). Software optimization resources.

[Forrester & Keane, 2009] Forrester, A. I. J. & Keane, A. J. (2009). Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1), 50–79.

[Forrester et al., 2007] Forrester, A. I. J., Sóbester, A., & Keane, A. J. (2007). Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 463(2088), 3251–3269.

[Forrester et al., 2008] Forrester, A. I. J., Sobester, A., & Keane, A. J. (2008). *Engineering design via surrogate modelling : a practical guide*. J. Wiley.

[Gen & Cheng, 2000] Gen, M. & Cheng, R. (2000). *Genetic algorithms and engineering optimization*. Wiley.

[Gibbs & MacKay, 1997] Gibbs, M. & MacKay, D. J. C. (1997). Efficient implementation of Gaussian processes.

[Giles, 2008] Giles, M. B. (2008). Collected matrix derivative results for forward and reverse mode AD. *Lecture Notes in Computational Science and Engineering*, 64, 34–44.

[Gill, 2007] Gill, P. (2007). Numerical linear algebra and optimization.

[Gill et al., 1981] Gill, P. E., Murray, W., & Wright, M. H. (1981). Practical optimization.

[Griewank & Walther, 2008] Griewank, A. & Walther, A. (2008). *Evaluating derivatives : principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics.

[Han et al., 2012] Han, Z., Zimmerman, R., & Görtz, S. (2012). Alternative Cokriging Method for Variable-Fidelity Surrogate Modeling. *AIAA journal*, 50(5), 1205–1210.

[Han et al., 2009] Han, Z.-H., Görtz, S., & Zimmermann, R. (2009). On improving efficiency and accuracy of variable-fidelity surrogate modeling in aero-data for loads context. In *Proceeings of European Air and Space Conference*.

[Helbig & Scherer, 2011] Helbig, H. & Scherer, A. (2011). Neuronale Netze. *Vorlesungsskript*.

[Hodges & Pierce, 2011] Hodges, D. H. & Pierce, G. A. (2011). *Introduction to structural dynamics and aeroelasticity*. Cambridge University Press.

[Holland, 1975] Holland, J. H. (1975). Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence. *Ann Arbor: University of Michigan Press, 1975*.

[Howard & Gallimore, 1992] Howard, M. A. & Gallimore, S. J. (1992). Viscous Throughflow Modelling for Multi-Stage Compressor Design. In *Volume 1: Turbomachinery* (pp. V001T01A109).: ASME.

[Huang et al., 2006] Huang, D., Allen, T. T., Notz, W. I., & Miller, R. A. (2006). Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32(5), 369–382.

[Hutchinson, 1989] Hutchinson, M. F. (1989). A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3), 1059–1076.

[J. Forrester et al., 2006] J. Forrester, A. I., Keane, A. J., & Bressloff, N. W. (2006). Design and Analysis of "Noisy" Computer Experiments. *AIAA Journal*, 44(10), 2331–2339.

[J. Toal & Keane, 2011] J. Toal, D. J. & Keane, A. J. (2011). Efficient multipoint aerodynamic design optimization via cokriging. *Journal of Aircraft*, 48(5), 1685–1695.

[Jin et al., 2001] Jin, R., Chen, W., & Simpson, T. (2001). Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23(1), 1–13.

[Jones, 2001] Jones, D. R. (2001). A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization*, 21(4), 345–383.

[Jones et al., 1998] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13, 455–492.

[K.A., 2011] K.A. (2011). Processors - Define SSE2,SSE3 and SSE4.

[Keane, 2006] Keane, A. J. (2006). Statistical improvement criteria for use in multiobjective design optimization. *AIAA journal*, 44(4), 879–891.

[Kennedy & O'Hagan, 2000] Kennedy, M. C. & O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87, 1–13.

[Krige, 1953] Krige, D. G. (1953). A statistical approach to some basic mine valuation problems on the witwatersrand.

[Krüger, 2012] Krüger, F. (2012). *Entwicklung von parallelisierbaren Gradientenbasierten Verfahren zur automatisierten, Ersatzmodell-gest{ü}tzten Optimierung unter Nebenbedinungen f{ü}r CFD-FEM-Verdichterdesign*. PhD thesis.

[Kügeler, 2005] Kügeler, E. (2005). Numerisches Verfahren zur genauen Analyse der Kühleffektivität filmgekühlter Turbinenschaufeln.

[Lapworth & Shahpar, 2004] Lapworth, L. & Shahpar, S. (2004). DESIGN OF GAS TURBINE ENGINES USING CFD. *European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS 2004*.

[Le Gratiet, 2013] Le Gratiet, L. (2013). Bayesian Analysis of Hierarchical Multifidelity Codes *. 1, 244–269.

[Lengyel-Kampmann, 2015] Lengyel-Kampmann, T. (2015). *Vergleichende aerodynamische Untersuchungen von gegenläufigen und konventionellen Fanstufen für Flugtriebwerke*. PhD thesis.

[Lepot et al., 2011] Lepot, I., Leborgne, M., Schnell, R., Yin, J., Delattre, G., Falissard, F., & Talbotec, J. (2011). Aero-mechanical optimization of a contra-rotating open rotor and assessment of its aerodynamic and acoustic characteristics. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 225(7), 850–863.

[Lieblein & Seymour, 1955] Lieblein & Seymour (1955). Aerodynamic Design of Axial-flow Compressors. VI - Experimental Flow in Two-Dimensional Cascades.

[Mackkay, 1991] Mackkay, D. J. C. (1991). *Bayesian Methods for Adaptive Models*. PhD thesis, Citeseer.

[Mader et al., 2008] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., & Van Der Weide, E. (2008). ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers.

[Mark Gibbs, 1997] Mark Gibbs, D. J. M. (1997). Efficient Implementation of Gaussian Processes.

[Matheron, 1963] Matheron, G. (1963). Principles of geostatistics. *Economic geology*, 58(8), 1246–1266.

[McKay et al., 1979] McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2), 239.

[Mönig et al., 2000] Mönig, R., Mildner, F., & Röper, R. (2000). Viscous-Flow 2D-Analysis Including Secondary Flow Effects. In *Volume 1: Aircraft Engine; Marine; Turbomachinery; Microturbines and Small Turbomachinery* (pp. V001T03A104).: ASME.

[Müller-Töws, 2000] Müller-Töws, J. (2000). *Aerothermodynamische Auslegung der Meridianströmung mehrstufiger Axialverdichter mit Hilfe von Optimierungsstrategien*. PhD thesis.

[Murray, 2016] Murray, I. (2016). Differentiation of the Cholesky decomposition.

[Nürnberger, 2004] Nürnberger, D. (2004). *Implizite Zeitintegration für die Simulation von Turbomaschinenströmungen*. PhD thesis, Ruhr-Universität Bochum.

[Özkaya & Gauger, 2014] Özkaya, E. & Gauger, N. R. (2014). *One-shot methods for aerodynamic shape optimization*. PhD thesis.

[Peter & Dwight, 2010] Peter, J. E. & Dwight, R. P. (2010). Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches. *Computers & Fluids*, 39(3), 373–391.

[Pierret, 1999] Pierret, S. (1999). *Designing turbomachinery blades by means of the function approximation concept based on artificial neural network, genetic algorithm, and the Navier-Stokes equations*. PhD thesis, Faculté Polytechnique de Mons - Von Karman Institute for Fluid Dynamics.

[Pierret & Van den Braembussche, 1998] Pierret, S. & Van den Braembussche, R. A. (1998). Turbomachinery Blade Design Using a Navier-Stokes Solver and Artificial Neural Network. In *Volume 1: Turbomachinery* (pp. V001T01A002).: ASME.

[Plackett, 1950] Plackett, R. L. (1950). Some theorems in least squares. *Biometrika*, 37(1/2), 149–157.

[p.l.c., 2016] p.l.c., B. (2016). *BP Statistical Review ofWorld Energy June 2016*. Technical report.

[Queipo et al., 2005] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., & Tucker, P. K. (2005). Surrogate-based Analysis and Optimization.

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie; Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog.

[Reid & Moore, 1978] Reid, L. & Moore, R. D. (1978). Design and overall performance of four highly loaded, high speed inlet stages for an advanced high-pressure-ratio core compressor.

[Reimer, 2016] Reimer, E. (2016). *Vergleichende Optimierung eines Fans mit High- und Multifidelity Verfahren*. Bachelor thesis, Fachhochschule Aachen.

[Riedmiller & Braun, 1993] Riedmiller, M. & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on* (pp. 586–591).: IEEE.

[Särkkä, 2013] Särkkä, S. (2013). *BAYESIAN FILTERING AND SMOOTHING*. Cambridge University Press.

[Schnös & Nicke, 2017] Schnös, M. & Nicke, E. (2017). Exploring a Database of Optimal Airfoils for Axial Compressor Design. *AIAA/CEAS Aeroacoustics Conference*.

[Schönweitz et al., 2013] Schönweitz, D., Voges, M., Goinis, G., Enders, G., & Johann, E. (2013). Experimental and Numerical Examinations of a Transonic Compressor-Stage With Casing Treatment. In *Volume 6A: Turbomachinery* (pp. V06AT35A035).: ASME.

[Shahpar, 2000] Shahpar, S. (2000). A Comparative Study of Optimisation Methods for Aerodynamic Design of Turbomachinery Blades. In *Volume 1: Aircraft Engine; Marine; Turbomachinery; Microturbines and Small Turbomachinery* (pp. V001T03A087).: ASME.

[Shepard, 1968] Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference* (pp. 517–524).: ACM.

[Shyy et al., 2001] Shyy, W., Papila, N., Vaidyanathan, R., & Tucker, K. (2001). Global design optimization for aerodynamics and rocket propulsion components. 37, 59–118.

[Siller et al., 2009] Siller, U., Voß, C., & Nicke, E. (2009). Automated Multidisciplinary Optimization of a Transonic Axial Compressor. *AIAA Aerospace Sciences Meeting*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.

[Simpson et al., 2008] Simpson, T., Toropov, V., Balabanov, V., & Viana, F. (2008). Design and Analysis of Computer Experiments in Multidisciplinary Design Optimization: A Review of How Far We Have Come - Or Not. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* Reston, Virigina: American Institute of Aeronautics and Astronautics.

[Smith, 1995] Smith, S. P. (1995). Differentiation of the Cholesky Algorithm. *Journal of Computational and Graphical Statistics*.

[Sozio et al., 2013] Sozio, E., Verstraete, T., & Paniagua, G. (2013). Design-Optimization Approach to Multistage Axial Contra-Rotating Turbines. In *Volume 6B: Turbomachinery* (pp. V06BT37A016).: ASME.

[Steimann et al., 2012] Steimann, P., Frenkel, M., & Keller, D. (2012). Moderne Programmiertechniken und Methoden.

[Tang et al., 2016] Tang, X., Luo, J., & Liu, F. (2016). Adjoint-Response Surface Method in Aerodynamic Shape Optimization of Turbomachinery Blades. In *Volume 2C: Turbomachinery* (pp. V02CT39A004).: ASME.

[Thornburg, 2006] Thornburg, H. (2006). Autoregressive Modeling: Elementary Least-Squares Methods. *Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University Stanford, California 94305*.

[Toal et al., 2009] Toal, D., Forrester, A., Bressloff, N., Keane, A., & Holden, C. (2009). An adjoint for likelihood maximization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2111), 3267–3287.

[Toal et al., 2011] Toal, D. J., Bressloff, N., Keane, A., & Holden, C. (2011). The development of a hybridized particle swarm for kriginghyperparameter tuning.

[Toal, 2016] Toal, D. J. J. (2016). A study into the potential of GPUs for the efficient construction and evaluation of Kriging models. *Engineering with Computers*, 32, 377–404.

[Verstraete, 2008] Verstraete, T. (2008). *MULTIDISCIPLINARY TURBOMACHINERY COMPONENT OPTIMIZATION CONSIDERING PERFORMANCE, STRESS, AND INTERNAL HEAT TRANSFER*. PhD thesis, University of Ghent - Von Karman Institute.

[Verstraete et al., 2014] Verstraete, T., Prinsier, J., & Cosi, L. (2014). Design and Off-Design Optimization of a Low Pressure Steam Turbine Radial Diffuser Using an Evolutionary Algorithm and 3D CFD. In *Volume 1B: Marine; Microturbines, Turbochargers and Small Turbomachines; Steam Turbines* (pp. V01BT27A046).: ASME.

[Voß et al., 2014] Voß, C., Aulich, M., & Raitor, T. (2014). Metamodel Assisted Aeromechanical Optimization of a Transonic Centrifugal Compressor.

[Voß & Nicke, 2008] Voß, C. & Nicke, E. (2008). Automatische Optimierung von Verdichterstufen. *AG Turbo COOREFF 1.1.1*, (September), 1–71.

[Weicker, 2015] Weicker, K. (2015). *Evolutionäre Algorithmen*. Springer Vieweg.

[Willburger, 2011] Willburger, A. (2011). *Beitrag zur Berechnung der Meridianströmung in Axialverdichtern auf der Basis der umfangsgemittelten Navier-Stokes-Gleichungen unter Berücksichtigung dreidimensionaler Einflüsse*. Kassel University Press.

[Willeke & Verstraete, 2015] Willeke, S. & Verstraete, T. (2015). Adjoint Optimization of an Internal Cooling Channel U-Bend. In *Volume 5A: Heat Transfer* (pp. V05AT11A029).: ASME.

[Wu, 1952] Wu, C.-H. (1952). A general theory of three-dimensional flow in subsonic and supersonic turbomachines of axial-, radial-, and mixed-flow types.

[Z.-H. Han, R. Zimmermann, 2010] Z.-H. Han, R. Zimmermann, S. G. (2010). A New Cokriging Method for Variable-Fidelity Surrogate Modeling of Aerodynamic Data. *48th AIAA Aerospace Sciences Meeting*, (January), 1–22.

[Zimmermann & Han, 2009] Zimmermann, R. & Han, Z.-h. (2009). Simplified cross-correlation estimation for Cokriging predictors as multi-fidelity surrogate models. (1984), 1–24.