Deutsches Zentrum
für Luft- und Raumfahrt e.V.

Dissertation

Multifidelity-Optimierungsverfahren für Turbomaschinen Dissertation

Andreas Schmitz

Institut für Antriebstechnik
Köln

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

Multifidelity-Optimierungsverfahren für Turbomaschinen Dissertation

Andreas Schmitz

Institut für Antriebstechnik
Köln

178 Seiten
27 Bilder
6 Tabellen
101 Literaturstellen

Multifidelity-Optimierungsverfahren für Turbomaschinen Dissertation

Deutsches Zentrum
für Luft- und Raumfahrt e.V.

Sicher:

Noch einbaue n: [1][2][3][4][5][6, 7][8][9][10] [11](10dimensions Samples minimum) [12](3dimensions Samples minimum in vie-len Fällen)

Kurzfassung

Abstract

Vorwort

Lalalaa

Nomenklatur

Abkürzungen

δ	Kronecker Delta
κ	Konditionszahl
λ	Diagonalaufschlag
$\mathbf{Cov}$	Kovarianzmatrix
$\mathbf{R}$	Korrelationsmatrix
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
$\vec{F}$	Der Vektor entspricht beim Ordinary-Kriging $\vec{1}$ und beim Gradient-Enhanced-Kriging sind die ersten $n - m$ Einträge Eins und für die restlichen m Einträge Null
$\vec{h}$	Euklidischer Abstandsvektor
$\vec{r}$	Korrelationsvektor
$\vec{x}$	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
$f(...)$	Entscheidungsfunktion
$F(\vec{x})$	Fehlerfunktion an der Stelle $\vec{x}$
h	Allgemeiner Hyperparameter
I	Informationsgehalt
L	Likelihood Funktion
m_f	Anzahl der verwendenten Fidelities bei Multifidelity Verfahren
N	Multivariate Normalverteilung
P	Wahrscheinlichkeit
t	Zeit
V	Kummulierte Volumenzugewinn einer Optimierung
w_i	Gewichte eines Kriging Modells
$y(\vec{x})$	Bekannter Funktionswert oder Stützstelle an der Stelle $\vec{x}$
$y^*(\vec{x})$	Geschätzter Funktionswert an der Stelle $\vec{x}$
β_i	Eintrag des Beta Vektors, entspricht einem Erwartungswert. Beim Ordinary Kriging gibt es nur einen Erwartungswert
AVX	Advanced Vector Extensions, eine Befehlssatzerweiterung für Prozessoren. Nachfolger der SSE Architektur.
HPC	High-Performance-Computing
k	Anzahl der freien Variablen eines Members
m	Anzahl der gegebenen partiellen Ableitungen
Member	Ein Satz freier Variablen mit dazugehörigen Funktionswerten. Beispielsweise ein Satz geometrischer Parameter mit berechneten mechanischen und aerodynamischen Größen.
n	Anzahl der beprobten Stützstellen, auch Member oder Samples genannt
o	Anzahl der Hyperparameter

s	Anzahl der verschiedenen Gütestufen bei einem Mutlifidelty Modell
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.

Inhaltsverzeichnis

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 [13]. 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. All diese Punkte machen die Auslegung von Turbomaschinen zu einer sehr großen Herausforderung für Forschung und die Industrie. Einerseits steht die wirtschaftliche Rentabilität im Vordergrund und andererseits umwelttechnische Aspekte. Um all diese Punkte erfüllen zu können, muss ein besonderes Augenmerk auf Emissionen, Effizienz, Betriebs- und Wartungskosten gelegt werden.

Ein sehr wichtiges Werkzeug um diesen Problemen zu begegnen sind moderne CFD- und Optimierungsverfahren. Diese 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 führten zu einer Verlagerung von sehr kostspieligen experimentellen Untersuchungen hin zu Untersuchungen am Computer durch CFD-Verfahren. Heutzutage werden experimentelle Untersuchungen häufig zur Validierung und Kalibrierung von CFD-Verfahren verwendet und finden zudem Anwendung im Bereich der Grundlagenforschung von Strömungsphänomenen.
Durch die neuen numerischen Verfahren wurde die Auslegungszyklen im Turbomaschinenbereich in den letzten Jahren massiv beschleunigt. Allerdings sind die verwendeten Optimierungsverfahren mit einem so hohen numerischen Aufwand verbunden, dass stark vereinfachte CFD-Modelle verwendet werden müssen. Durch diese Vereinfachungen können diese Modelle allerdings nicht alle Problemstellungen auflösen, sodass die Forschung auch hier noch sehr viele Fortschritte macht und machen kann.

Um diesem Problem zu begegnen, 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. Und 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 zeitlich aufgelöst am Rechner zu simulieren und zu optimieren noch sehr weit entfernt. Selbst ein vollständiges Triebwerk zu optimieren ist mit momentanen Mitteln nicht machbar.

In dieser Problemstellung liegt auch die Motivation dieser Arbeit. Es soll ein industriell nutzbares Optimierungsverfahren vorgestellt werden, welches es ermöglichen soll höherwertigere CFD-Verfahren zu verwenden. Hierfür wurde ein effizientes Optimierungsverfahren entwickelt und getestet. Wobei auf industrielle HPC-Hardwareumgebung und numerische Effizienz des Optimierungsverfahrens geachtet wurde. Durch dieses Verfahren soll es in Zukunft ermöglicht werden, deutlich komplexere Sachverhalte zu optimieren.

1.2 Stand der Technik

Bisher hauptsächlich nur einzelne Komponenten (Verdichter) simuliert. Instationär und kein RANS sind immer noch sehr schwierig in einer Optimierung zu verwenden. Die Rechnerkapazität ist auch noch in absehbarer Zeit nicht ausreichend um hochauflösende Modelle von vielen zusammengesetzten Komponenten zu optimieren.

Optimierungen jetzt, wie laufen diese meistens (Paper und beuspiele nennen) einige Turbomaschinenanwendungen nennen mit typischen Zielfunktions parameter und Nebenbeindungensaznahl, RANS z.B. typisch aber immer öfter auch instationäre Simulationen gewünscht. Auch mechanische Kriterien werden immer wichtiger, Flattern usw. Anzahl der Betriebspunkte. Beschleunigung über Ersatzmodelle in Optimierungen bereits sehr etabliert.

Bisherige Implementierungen des Krigings sind meist durch die Rechenkapazität begrenzt auf relativ kleine Parameteranzahlen und Sampleanzahl. Durch GEK und CO kriging wird das Kriging aber immer mehr zu einem Falschenhals.

- Mit zunehmenden Forschungsgrad und mit steigender Rechnerleistung werden auch immer komplexere Problemstellungen optimiert. Die Anforderungen an die Rechnerleistung sind im Turbomaschinenbereich allerdings so hoch, dass sie selbst mit sehr

großen HPC-Clustern nicht erfüllt werden können.

AG Turbo Abschlussbericht von Christian zitieren, besonders den Stand der Technik:

Kurzen Abriß über stochastische Optimierungsverfahren und warum sie hier verwendet werden

Kurzen Abriß über verschiedene Ersatzmodelle und deren Vor- und Nachteile

Kurzen Abriß über die Kriging Verfahren und deren Erweiterungen

Kurzen Abriß über den steigenden Aufwand der Kriging Verfahren und der bisher erreichten Größenordnungen

1.3 Zielsetzung

Fidelities nutzen Bei der Netzauswahl bspw. muss ein Kompromiss gefunden werden zwischen Genauigkeit und Geschwindigkeit. Das ist schwer bis nahezu unmöglich für den gesamten Optimierungsraum. Unterschiede zieltich in den Fidelities MF Optimierung zur Beschleunigung der Optimierung, durch Nutzung von mehreren Fidelities. Maximal möglicher Zeitgewinn ist der Faktor zwischen Low und High Fdeiltiy. Außerdem wird gehofft, dass die MF Optimierung durch eine breitere Samplem Abdeckung bessere Ergebnisse findet. (globales Minimum) Benutzerfreundlich, ohne viel Eingaben Da das Training sehr aufwendig werden kann, sollen zudem moderne Beschleunigungsverfahren entwickelt werden. Das Training soll asynchron parallelisiert werden und durch unterschiedliche Prozessoren auch im Mixbetrieb beschleunigt werden können. Zudem soll das Training über eine 1:1 Verbindung ausgelagert werden können auf einen leisteungsstarken Rechner außerhalb unseres Clusters, da dort keine GPUs vorhanden sind. Also GPU/CPU z.B. Die Software soll zudem Objektorientiert aufgebaut werden, so dass auch andere Ersatzmodelle integriert werden können. Bspw. Supporting Vector Machines oder GEK. GEK +Adjoint kurz erwähnen, wurde ebenfalls integriert, aber die Diss bezieht sich eher auf das MF.

Problematik: 1. Viele Hyperparameter -> hoher Aufwand bei der Aufstellung der Korrelationsmatrizen und deren Ableitungen -> 1:n Verbindung wünschenswert

2. Viele Samples: Sehr hoher Aufwand bei den Matrixoperationen welche für das Training verwendet werden. -> 1:1 Verbindung zu einem extrem leistungsstarken Rechner außerhalb des Clusters

Anwendung in der Turbomaschinen Optimierung

2 Optimierungsstrategie Turbomaschine

Optimierungsverfahren erläutern.

-> Flussdiagramm Ordinary Kriging (Master Slave Prozesse und Ablauf)

-> Expected Volume Gain + rechnende Slaves (Marcel zitieren)

-> Statische Grundlagen einstreuen

2.1 Einleitung

1. -> Besonderheiten Turbomaschine:
   (a) Multi Physics, Objects, Contraints, viele freie Variablen, lange Prozessketzeiten,
   (b) oftmals wird das Ziel/Constraints nicht bekannt, wird noch oft verändert während der Optimierung
   (c) Oftmals nicht differenzierbare Funktoinen, von einigen vielleicht aber schon, daher Gr
      i. Beides: Keine Gradientenverfahren sinnvoll

Daraus ergibt sich Evolutionsstrategie sinnvoll

Problem: Lange Prozesskettenzeiten daher Beschleunigung ersatzmodelle

2.1.1 Grundlegender Optimierungsprozess und Evolutionsstrategie

Da die aerodynamische Auslegung von Triebswerkskomponenten immer einen Kompromiss zwischen den unterschiedlichsten Anforderungen darstellt, wurde bei der Entwicklung von AutoOpti besonderer Wert auf die Möglichkeit zur simultanen Optimierung mehrerer Zielfunktionen gelegt. Ein typisches Optimierungsziel / Zielfunktion für

einen Verdichter wäre z.B. die simultane Maximierung des Wirkungsgrads (Verhältnis von nutzbarer Energie zu genutzter Energie) bei gleichzeitiger Maximierung des Druckverhältnisses (Verhältnis vom Druck am Eintritt des Verdichters zum Druck am Austritt des Verdichters). Um die Zielfunktion(en) zu optimieren, kann der Optimierungsalgorithmus eine bestimmte Anzahl von freien Variablen verändern. Bei einem Verdichter wären das typischerweise Parameter, welche die Schaufelgeometrien variieren, beispielsweise die Länge der Schaufel oder die Dicke. In einer realen Optimierung werden oftmals hunderte von freien Variablen verwendet. Ein Satz von freien Parametern mit der entsprechenden Zielfunktion bezeichnet man als Member. Eine automatisierte Optimierung, wie sie im Institut für Antriebstechnik in Köln ausgeführt wird, basiert grundsätzlich auf einer Evolutionsstrategie [14]. Da die Optimierung meist auf einem Rechencluster ausgeführt wird, ist es sinnvoll, die Arbeit in verschiedene Prozesse einzuteilen, damit diese auf unterschiedlichen Rechnern verteilt werden können. In diesem Fall gibt es zwei verschiedene Arten von Prozessen, einen Root Prozess und mehrere Slave Prozesse.

- Der Root Prozess ist für die Verwaltung und Steuerung der Optimierung zuständig und läuft auf nur einem Rechner.
- Die Slave Prozesse, werden am Anfang der Optimierung durch den Root Prozess gestartet. In der Regel läuft jeder Slave Prozess auf einem anderen Rechner, da diese Prozesse den größten Teil der Arbeit übernehmen. Die Slave Prozesse bekommen vom Root Prozess freie Variablen gesendet und berechnen die dazugehörige Zielfunktion. Wie diese Berechnung abläuft, ist von Optimierung zu Optimierung unterschiedlich und kann vom Benutzer vorher in einer Prozesskette festgelegt werden. Meist werden Strömungslöser oder Programme zur mechanischen Berechnung gestartet.

Die Optimierungen laufen i.d.R. auf einem Cluster, wobei der Root Prozess auf einem Knoten ausgeführt wird und die Slaves jeweils einen eigenen Knoten zur Berechnung der Prozesskette bekommen.

Im Folgenden soll der grundsätzliche Ablauf einer AutoOpti-Optimierung auf Basis der Evolutionsstrategie kurz erläutert werden, detaillierte Ausführungen folgen im Anschluss:

1. Vor der eigentlichen Optimierung müssen initial einige Member erzeugt werden. Hierfür werden oftmals zufällig die freien Variablen variiert und die Zielfunktionen berechnet und dann in die Datenbank eingetragen. Dies wird solange durchgeführt, bis eine gewisse Anzahl von Membern in der Datenbank steht.

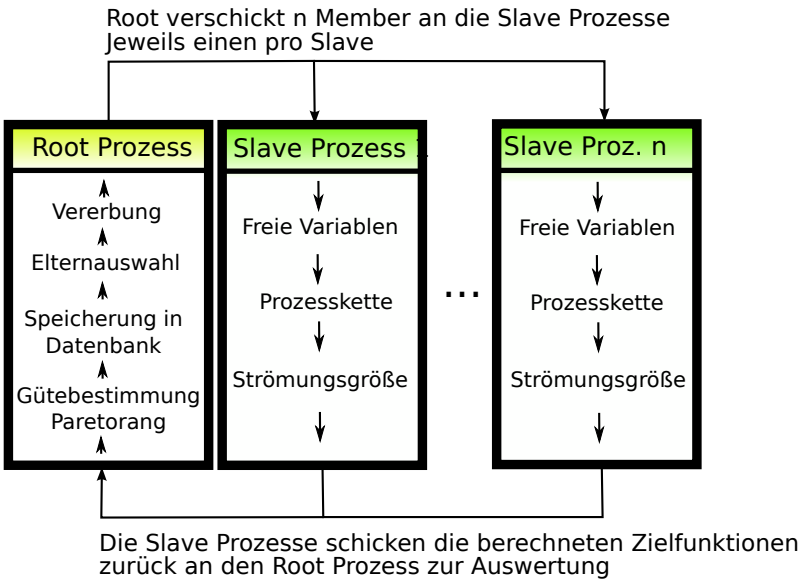


Abbildung 2.1: Prozesskette des genetischen Algorithmus in AutoOpti

2. Der Root Prozess wählt aus der vorhandenen Datenbank eine gewisse Anzahl von Membern als Eltern aus.

3. Der Root Prozess wendet Vererbungsoperatoren auf die Eltern an (Mutation, Kreuzungen usw.) und erzeugt so einen neuen Satz freier Variablen also einen neuen Member.

4. Der Root Prozess verschickt den neuen Member zur Berechnung an einen freien Slave X. Falls kein Slave frei ist, wartet der Root Prozess bis dies der Fall ist.

5. Der Slave Prozess X startet mit den empfangenen freien Variablen die Prozesskette und wartet bis diese beendet ist.

6. Ist die Prozesskette durchlaufen, wertet der Slave Prozess X die Ergebnisse der Prozesskette aus und berechnet daraus die Zielfunktion. Meist werden die Ergebnisse der Berechnung einfach direkt als Zielfunktion übernommen. Manchmal sind aber Normierungen oder andere Transformationen notwendig, daher der Zwischenschritt der Zielfunktionsberechnung.

7. Der Slave Prozess schickt die berechneten Zielfunktionen und weitere physikalische Parameter an den Root Prozess zurück und ist im Status bereit. Die weiteren physikalischen Parameter können beispielsweise für eine Nebenbedingung oder aber für den Benutzer zur Überwachung der Optimierung verwendet werden.

8. Die berechneten Zielfunktionen gehen zurück an den Root Prozess. Dieser bestimmt anhand der bestehenden Datenbasis und der Zielfunktion einen Gütewert. Danach wird der Member, gemäß seiner Güte, in die Datenbank einsortiert.

Vorteile dieser Strategie:

- Die Suche nach dem Optimum wird global durchgeführt.
- Die Zielfunktion muss nicht differenzierbar sein, was die Umsetzung dieser Strategie stark vereinfacht.
- Die Zielfunktion muss nicht immer bewertbar sein, beispielsweise bei nicht Konvergenz.
- Die Strategie ist sehr gut parallelisierbar, z.B. die oben genannte Einteilung in Root und Slave Prozesse.

Nachteile:

- Der größte Nachteil ist die Geschwindigkeit. Da die Prozesskette in der Regel sehr aufwendig zu berechnen ist, möchte man mit möglichst wenig erzeugten Membern zum Optimum der Zielfunktion gelangen. Allerdings müssen bei der Erzeugung über Vererbungsoperatoren sehr viele Member generiert werden, bis man zu einem zufriedenstellenden Ergebnis gelangt. Insbesondere aerodynamische Berechnungen sind extrem rechenintensiv. Diese nehmen trotz hoher Parallelisierung mehrere Stunden bis Tage auf mehreren Knoten in Anspruch.

2.2 Automatisierte Optimierung im DLR

Da der weltweite Flugverkehr sich seit dem Jahr 2000 fast verdreifacht hat, ergeben sich besondere Problemstellungen z.B. die Brennstoffknappheit, die Kapazitäten der Flughäfen und -routen sowie die steigende Umweltbelastung durch Lärm- und Schadstoffemissionen. Diese stehen immer mehr im Interesse der Öffentlichkeit. Um den genannten Problem gewachsen zu sein, sind weitreichende Forschungen in den verschiedensten Gebieten notwendig. Immer häufiger werden automatisierte Optimierer eingesetzt, um Triebswerkskomponenten hinsichtlich der genannten Ziele zu optimieren. Im Institut für Antriebstechnik des DLR wurde ein solcher automatisierter Optimierer namens AutoOpti [15, 16] entwickelt, welcher mittlerweile auch für Optimierungen außerhalb des Verdichterdesigns eingesetzt wird.

Online Änderungen möglich einbringen, was online änderbar ist auch

2.2.1 Ersatzmodellgestützte Optimierung

Um den Prozess zu beschleunigen, versucht man die Membererzeugung des Root Prozesses (Punkt 2 und 3) zu verbessern. Insbesondere versucht man Member zu erzeugen, die eine möglichst große Verbesserung der Zielfunktion aufweisen. Da die Erzeugung in der Evolutionsstrategie durch z.B. Mutation oder Kreuzung von Membern zufallsbasiert ist, sind die einzelnen Verbesserungen der Member eher klein und viele Iterationen und damit auch reale Funktionsauswertungen (z.B. Strömungslösungen) notwendig, um das gewünschte Ziel zu erreichen. Ersatzmodelle wie z.B. das Kriging Modell approximieren oder interpolieren anhand der vorhandenen Daten die Zielfunktion. Bei jeder Iteration muss ein solches Modell trainiert werden. Das Training ist in der Regel der zeitaufwendigste Teil der Verwendung eines Ersatzmodells, im Vergleich zu einer Strömungslösung ist der Aufwand allerdings zu vernachlässigen. Nach dem Training können Funktionswerte an beliebigen Stellen vorhergesagt werden. Die Vorhersage von Funktionswerten durch ein Ersatzmodell ist meist auch um einige Größenordnungen schneller als die echte Funktion, bspw. ein Strömungslöser. Die Ersatzmodelle werden dann herangezogen, um mit den Vorhersagen neue Member zu erzeugen. Die Erzeugung kann dann z.B. durch eine kleine Optimierung auf dem Ersatzmodell erfolgen. Der von dem Ersatzmodell beste vorhergesagte Member wird als neuer Member verwendet und anschließend mit der echten Funktion (bspw. ein Strömungslöser) nachgerechnet. Durch dieses Verfahren lassen sich die Anzahl der echten Funktionsauswertungen stark reduzieren, wie stark hängt natürlich von der Güte des verwendeten Modells ab.

Abbildung 2.2 zeigt den veränderten Root Prozess (die Slave Prozesse wurden aus Platzgründen ausgeblendet). Die Punkte 2 und 3 werden dabei durch folgenden Ablauf ersetzt:

1. Das Ersatzmodell wird mit der bereits vorhandenen Datenbank trainiert. Das Training stellt auch den aufwendigsten Teil der Nutzung des Modells dar.

2. Danach wird eine eigene Optimierung auf dem Ersatzmodell gestartet, wobei das Ersatzmodell hier den Teil der Prozesskette ersetzt und eine Approximation dieser darstellt. Da die Funktionsauswertungen auf dem Ersatzmodell erheblich schneller sind als die Prozesskette selbst, geht dieser Schritt relativ schnell.

3. Nach der Optimierung muss eine Auswahl der vom Ersatzmodell vorhergesagten Member vorgenommen werden. Hierfür gibt es diverse Ansätze, im einfachsten Fall würde man den Member mit der niedrigsten vorhergesagten Zielfunktion nehmen.

4. Der Root Prozess schickt die ausgewählten Member nun an einen Slave Prozess.

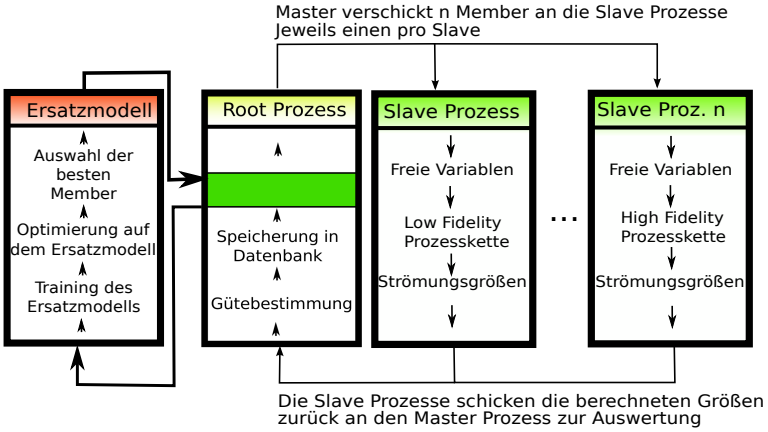


Abbildung 2.2: Nutzung eines Ersatzmodells im Optimierungsprozess

Der Slave Prozess berechnet dann die echte Zielfunktion, sodass diese dann auch mit der Vorhersage des Ersatzmodells verglichen werden kann.

In AutoOpti verwendete Ersatzmodelle sind bayesisch trainierte Neuronale Netzwerke [17] und Kriging.

2.2.2 Expected Volume Gain

Aus [18][10] für eine Fitnessfunktion

$$EI\left(x\right) = E\left[I\right] P\left[I\right]$$

$$= \left(b - E\left[y*\left(x\right)\right]\right) * P\left(y*\left(x\right)\right) \left| y*\left(x\right) < b \right.$$

$$= \left(b - \frac{\int_{-\infty}^{b} p\left(y*\left(x\right)\right) y*\left(x\right) dy*}{\int_{-\infty}^{b} p\left(y*\left(x\right)\right) dy*}\right) \int_{-\infty}^{b} p\left(y*\left(x\right)\right) dy*$$

$$= \int_{-\infty}^{b} p\left(y*\left(x\right)\right) \left(b - y*\left(x\right)\right) dy*$$

Dies entspricht:

$$= [b-]$$

Theorie[9][11]

3 Multifidelity Optimierungsstrategie

COKRIGING:

3.1 TODO

- Änderungen der bisherigen Strategie - Datenbanken, Prozesskette, Training, Entscheidungsfunktion, Optimierung auf dem Ersatzmodell, LF nur über das EM gekoppelt - Trennen EM / AutoOpti über Interfaces - Kopplung C/C++
Entscheidungsfunktion über Varianz VG ohne Sampling überhaupt sinnvoll?

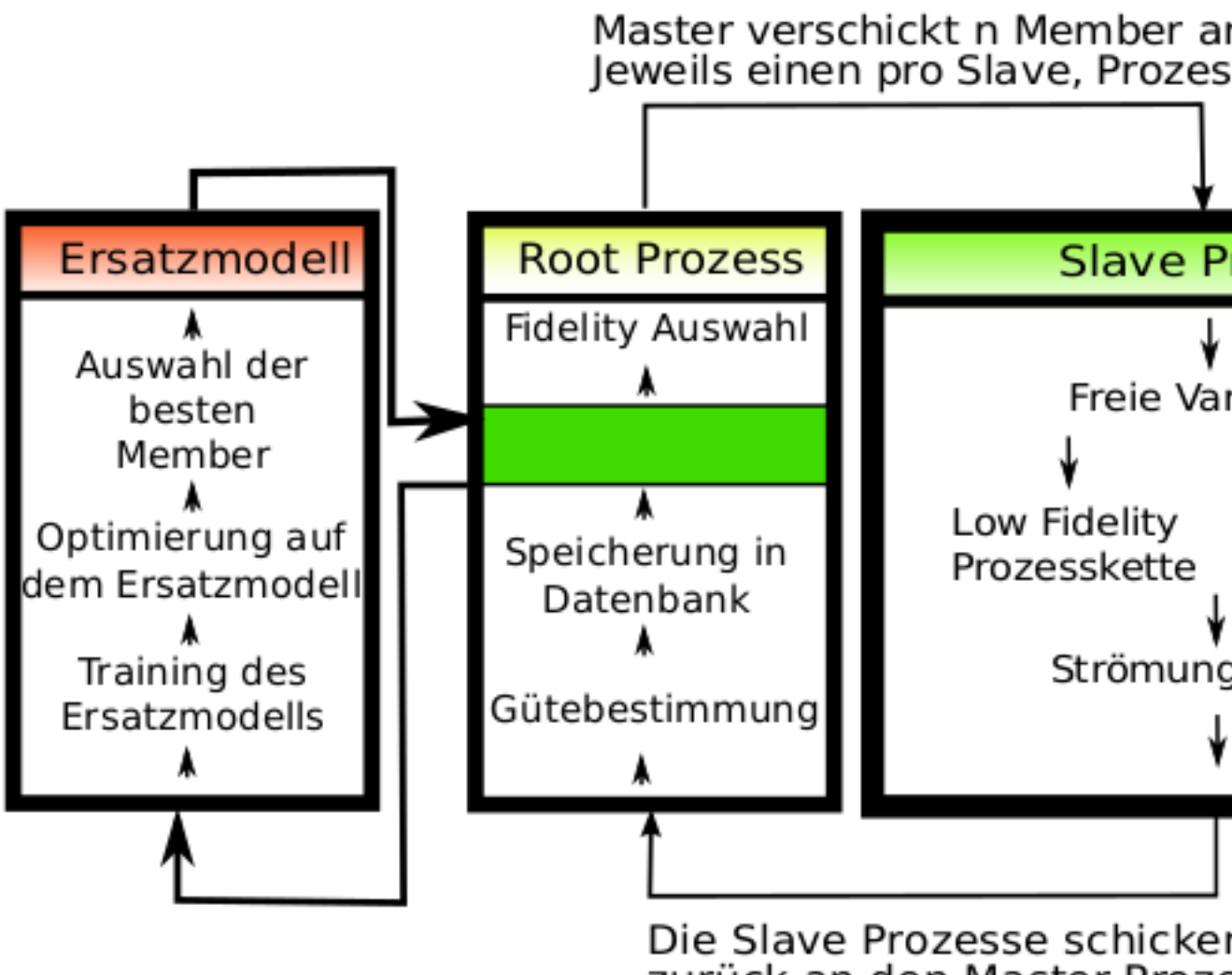
Hat man sich nach der Optimierung auf dem Ersatzmodel bereits für einen Ort entschieden und möchte nur noch entscheiden, ob dieser Ort mit der High Fidelity oder der Low-Fidelity Prozesskette durchlaufen werden soll, ist nicht klar, ob die Varianz des EVG eine geeignete Entscheidungsfunktion darstellt.

Grundlegend möchte man die Entscheidung in diesem Fall so treffen, dass sich das Ersatzmodell pro Zeit maximal verbessert. Die Frage die sich stellt ist, welches Maß für die Ersatzmodellverbesserung sinnvoll ist. Auf den ersten Blick, erscheint die Varianz des Expected Volume Gain ein geeignetes Maß darzustellen, da diese alle Fitnessfunktionen und auch Nebenbedingungen beinhaltet und diese sinnvoll in einem Maß miteinander kombiniert.

Betrachtet man folgenden Fall, dass ein neuer Member außerhalb der Restriktionen vorgeschlagen wird, dieser aber eine sehr hohe Standardabweichung in der Vorhersage hat. Die Wahrscheinlichkeit dafür, dass dieser Member innerhalb der Restriktionen liegt, ist aber aufgrund der hohen Standardabweichung ebenfalls relativ hoch. Fügt man dem Ersatzmodell nun einen Low-Fidelity Member hinzu, so sollte die Standardabweichung in der Vorhersage reduziert werden. Das Expected Volume Gain sollte in diesem Fall kleiner werden, vorausgesetzt man sampled die Low-Fidelity Vorhersage nicht. Tut man dies, ist die Frage wie sich die Varianz des EVG ändert,

- Verschiedenen Verfahren nennem z.B. gestuftes Verfahren also erst Low und dann mit den Ergebnissen eine High Fidelity Optimierung starten

Die Einbindung eines Multi-Fidelity Verfahrens in eine automatisierte Optimierung stellt


Master verschickt n Member a
Jeweils einen pro Slave, Prozes
Ersatzmodell
Auswahl der besten Member
Optimierung auf dem Ersatzmodell
Training des Ersatzmodells
Root Prozess
Fidelity Auswahl
Speicherung in Datenbank
Gütebestimmung
Slave P
Freie Va
Low Fidelity Prozesskette
Strömung
Die Slave Prozesse schicke
zurück an den Master Proze

einen vor zahlreiche Probleme. Die meisten dieser Probleme entstehen daraus, dass es mehrere Prozessketten gibt, um einen Member zu berechnen. In der Regel gibt es eine Low-Fidelity Prozesskette 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, mit dem Ziel 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 (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. Die größte Schwierigkeit innerhalb der Optimierung besteht allerdings darin zu entscheiden, wann welche Prozesskette verwendet wird. Im folgenden Kapitel sollen verschiedene denkbare Lösungen für dieses Problem beleuchtet werden.

- Annahmen bei unserem Verfahren: DIe Thetas ändern sich nicht, also das neue Sample passt in die Verteilung, und die Vertilung ist gut geschätzt. Lokale Varianzreduktion,

3.2 Vor- und Nachteile von Multifidelity Optimierungen auf Basis eines Multifidelity Ersatzmodells

Grundsätzlich stellt sich als erstes die Frage nach dem Nutzen einer Multifidelity Optimierung und insbesondere wo die Vorteile eines Multifidelity Ersatzmodells wie dem CO-Kriging Verfahren liegen. Es wäre ja auch durchaus denkbar zuerst eine reine Low-Fidelity Optimierung durchzuführen und mit den Ergebnissen eine High Fidelity Optimierung zu starten. Die Hoffnung hierbei wäre, dass die Low-Fidelity Funktion nur verschoben ist und das Minimum sich an derselben Stelle im Parameterraum befindet. Die folgende Abbildung zeigt ein solches Verhalten, in rot dargestellt ist die Low-Fidelity Funktion und in schwarz die High-Fidelity Funktion. Die Low-Fidelity Funktion ist um -1 verschoben und zusätzlich leicht gestört worden. Würde man in diesem Fall zuerst die

Low-Fidelity Funktion optimieren, so würde man mit hoher Wahrscheinlichkeit das richtige Minimum finden und bräuchte die High-Fidelity Funktion nicht mehr zu optimieren.

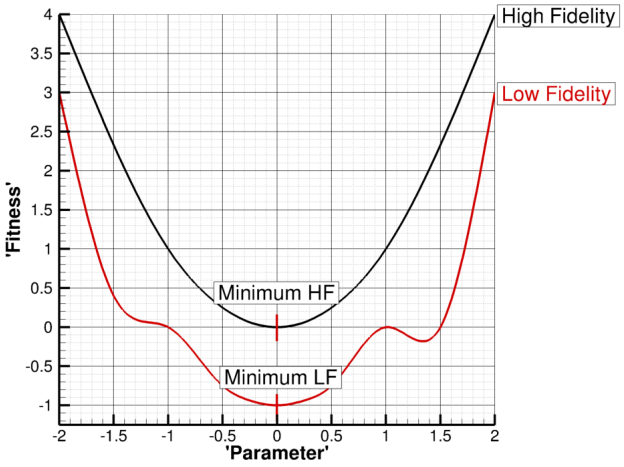


Abbildung 3.2:

In der Realität ist es allerdings möglich, dass die Low-Fidelity Funktion lokale Minima aufweist die in der High-Fidelity Funktion nicht auftreten müssen. Startet man nun eine Low-Fidelity Optimierung und findet ein solche lokales Minimum und verwendet dieses als Startpunkt in der High-Fidelity Optimierung, so hat man im schlimmsten Fall nichts gewonnen. Die gesamte vorangegangene Low-Fidelity Optimierung war somit umsonst oder hat sich zumindest zeitlich nicht rentiert. Man sollte zudem nicht vergessen, dass man es in realen Anwendungen mit hochdimensionalen Funktionen zu tun hat und die Wahrscheinlichkeit für lokale Minima eher steigen wird.

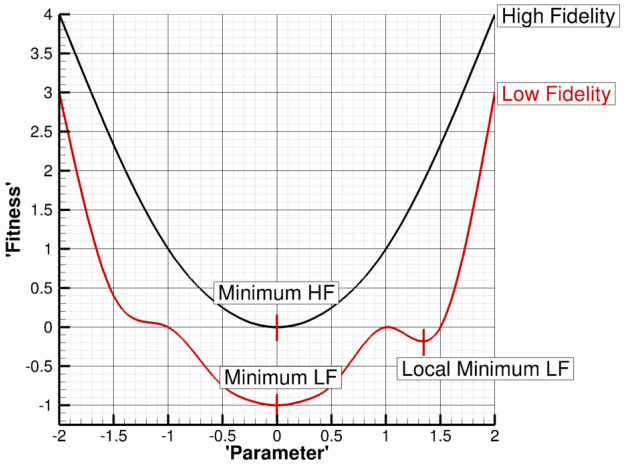


Abbildung 3.3:

Der umgekehrte Fall wäre ebenfalls möglich, in diesem wäre die High-Fidelity Funktion komplexer als die Low-Fidelity Funktion.

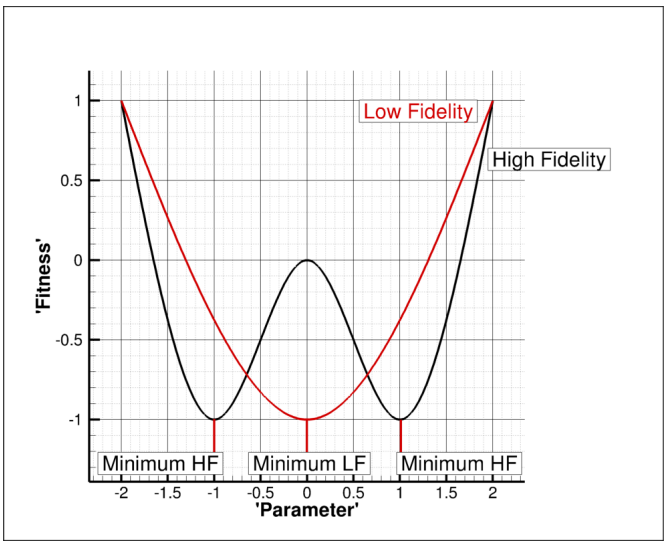


Abbildung 3.4:

Ein weiteres Problem bei der stufenweisen Optimierung ist die Verwendung von Restriktionen. In der folgenden Abbildung sieht man die bereits vorgestellte Funktion. Allerdings gibt es hier noch eine Restriktion welche durch die zwei blauen Linien gekennzeichnet ist. Ziel dieser Restriktion ist es, dass sich der Funktionwert innerhalb dieser beiden blauen Linien befindet. Verwendet man nun diese Restriktion in der vorhergehenden Low-Fidelity Optimierung, so reicht eine simple Verschiebung der Low-Fidelity Funktion bereits aus, um das gesuchte Minimum völlig zu verfehlen. Startet man nun mit dem Ergebnis aus der Low-Fidelity Optimierung die High-Fidelity Optimierung, so hat die Low-Fidelity Optimierung letztlich keinerlei Beschleunigung gebracht, im Gegenteil sogar.

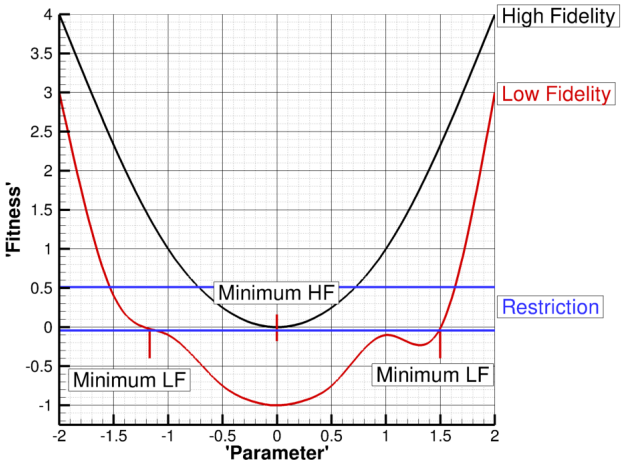


Abbildung 3.5:

Diese Nachteile existieren mit einem Multifidelity Ersatzmodell wie dem CO-Kriging praktisch nicht. Das Ersatzmodell würde die Low-Fidelity Daten nur als Tendenzen mitnehmen insofern diese Informationen über die High-Fidelity Funktion enthalten. Im schlimmsten Fall, würden die Low-Fidelity Daten einfach nicht verwendet.

Das Ersatzmodell bringt zusätzlich noch die Information inwiefern sich Standardabweichung des Vorhersagefehlers durch ein Low-Fidelity Member reduziert. Diese Information kann verwendet werden, wenn es notwendig ist zu entscheiden, ob ein neuer Member in der Optimierung mit der Low- oder der High-Fidelity Prozesskette berechnet werden soll.

Die maximal mögliche Beschleunigung der Multifidelity Optimierung ist der zeitliche Unterschied zwischen den Prozessketten, ist die Low-Fidelity Prozesskette 5x so schnell wie die High-Fidelity Prozesskett, dann ist das auch die maximal mögliche Beschleunigung der Optimierung. In der Realität wird dieser Wert natürlich niemals ganz erreicht, im Weiteren sollen aber verschiedene Verfahren vorgestellt werden mit denen man möglichst nah an diesen theoretischen Wert kommen kann.

Ein weiterer Vorteil der Multifidelity Optimierung ist, dass bei der initialen (oftmals zufälligen) Sample Erzeugung sehr viele Low-Fidelity Samples erzeugt werden können. Dies kann vorteilhaft sein, um größere Bereiche der Funktion bereits mit Low-Fidelity Daten abzudecken und so den gesamten Funktionsverlauf besser einschätzen zu können. Die Wahrscheinlichkeit ein besseres lokales Minimum zu finden oder sogar das globale Optimum wird so erhöht.

Nachteilig an der Verwendung von Multifidelity Erstzmodellen ist insbesondere der zusätzliche Entwicklungsaufwand. Zudem müssen für eine Optimierung 2 Prozessketten bereitgestellt werden und die Low-Fidelity Funktion sollte auch in irgendeiner Form mit

der High-Fidelity Funktion korreliert sein. Ist die Low-Fidelity Funktion vollständig unkorreliert, wird dies nach einer Zeit vom Ersatzmodell zwar erkannt, allerdings benötigt das Modell bis dahin einige Low-Fidelity Samples, die in diesem Fall dann umsonst gerechnet worden sind. Verwendet man aber bspw. ein grobes und ein feines Rechennetz, so ist kaum davon auszugehen, dass die Funktionen komplett unkorreliert sind.

Ein weiterer Nachteil ist, dass das Training für ein Multifidelity Ersatzmodell aufwendiger und auch komplexer ist. In der Regel ist die Trainingszeit aber zu vernachlässigen.

3.3 Allgemeine Annahmen über die Erzeugung neuer Member in einer Multi-Fidelity Optimierung

Ein großes Problem bei der Durchführung einer Multi-Fidelity Optimierung ist die Erzeugung eines neuen Members. Der grundlegende Ablauf der Erzeugung ist in Kapitel 2.2 beschrieben. Die Schwierigkeit besteht darin zu entscheiden, ob dieser Member mit der Low-Fidelity Prozesskette oder der High-Fidelity Prozesskette berechnet werden soll. Auf der einen Seite sollte die Erzeugung eines Low-Fidelity Members eine vergleichsweise kurze Zeit $t_{low} < t_{high}$ benötigen. Auf der anderen Seite aber einen größeren Fehler $F_{low} \geq F_{high}$ haben als das High-Fidelity Modell und dadurch weniger Informationszugewinn für das CO-Kriging Modell und die Optimierung liefern. Benötigt wird also eine Entscheidungsfunktion f die diese Entscheidung optimal vornimmt. Als optimal nehmen wir an, dass die Entscheidungsfunktion ihre Auswahl so trifft, dass bei minimaler Zeit der maximale Informationszugewinn I für die gesamte Optimierung erreicht wird. Eine Entscheidungsfunktion zu finden, die dies erfüllt 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.

Als erstes nehmen wir an, dass es nur zwei mögliche Fidelities gibt, also eine Prozesskette mit einer hohen Güte $high$ und eine mit einer niedrigeren Güte low. Zudem definieren wir eine Zeit t, welche im Wesentlichem der Zeit entspricht, die für die Berechnung der jeweiligen Prozesskette $t \in \{t_{low}, t_{high}\}$ benötigt wird. Die Zeit für das Ersatzmodelltraining soll hier vernachlässigt werden. Wie bereits schon erwähnt, definieren wir zusätzlich noch eine abstrakte Größe, welche den Informationszugewinn oder den Fortschritt definiert, der durch einen Member in der Optimierung erreicht wird. Diese Größe $I \in \{I_{low}, I_{high}\}$ ist für jeden Member und für jede Fidelity unterschiedlich.

Für eine Multi-Fidelity Optimierung stellt sich außerdem die Frage, wie man die Gütebestimmung der Member durchführt, z.B. 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. Nimmt man beispielsweise an, dass alle Low-Fidelity Member zu schlechteren Fitness Werten verschoben sind, so würden diese grundsätzlich schlechter bewertet werden. Außerdem interessiert den Anwender in der Regel nur das High-Fidelity Ergebnis. Das wiederum bedeutet, dass nur High-Fidelity Member einen direkten Optimierungsfortschritt 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. Diese Thematik wird im Weiteren aber noch genauer besprochen. Zusammengefasst ergeben sich also folgende allgemeine Annahmen und Bedingungen:

- Zwei mögliche Fidelities werden betrachtet
- Die Zeit für das Ersatzmodelltraining wird vernachlässigt
- Nur High Fidelity Member werden für den Optimierungsfortschritt (z.B. Volumenzugewinn, Paretorang) verwendet
  - Der letzte Member sollte ein High Fidelity Member sein
  - Low-Fidelity Member können nur das Ersatzmodell verbessern und haben so nur indirekten Einfluss auf den Optimierungfortschritt

In Abbildung 3.6 sollen einige Möglichkeiten der Membererzeugung abgebildet und kategorisiert werden. Es wird zuerst unterschieden, ob der Informationszugewinn in irgendeiner Weise modelliert werden soll oder nicht.

3.4 Initiales Sampling

LHC/Random mit irgendeiner Wahrscheinlichkeit

3.5 Informationszugewinn nicht modellieren

Die einfachere 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.

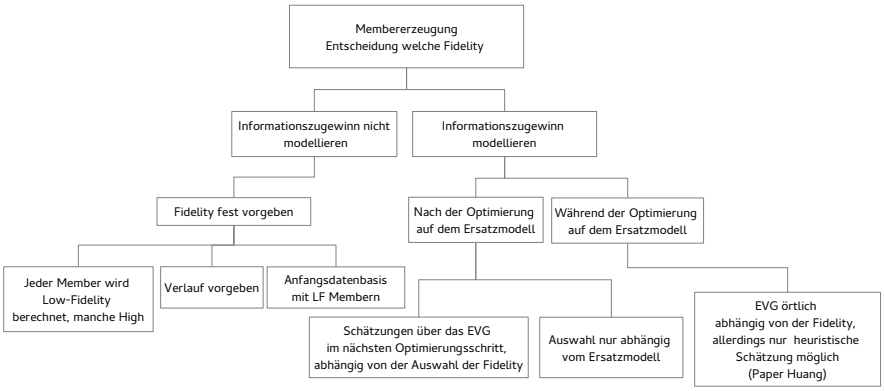


Abbildung 3.6: Möglichkeiten der Membererzeugung mit mehreren Fidelities

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 und 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.

CO-Kriging Formulierungen, welche auf einem Differenzmodell beruhen (z.B. [19]) 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.

Sind die Low- und High-Fidelity Prozessketten allerdings völlig unabhängig voneinander, verliert man diesen Zeitgewinn und damit wird die gesamte Optimierung deutlich länger benötigen. Betrachten wir daher die folgenden 3 Möglichkeiten:

1. Alle n Member werden High-Fidelity berechnet. Die Gesamtzeit beträgt: $t_{ges} = n * t_{high}$
2. Alle n Member werden Low-Fidelity berechnet, $x \in \{1...n\}$ Member High-Fidelity. Die Prozessketten sind unabhängig voneinander. Die Gesamtzeit beträgt also: $t_{ges} = n * t_{low} + x * t_{high}$
3. Alle n Member werden Low-Fidelity berechnet, $x \in \{1...n\}$ Member High-Fidelity. Ein High-Fidelity Member beinhaltet die Low-Fidelity Prozesskette. Die Gesamtzeit beträgt also: $t_{ges} = (n - x)\, t_{low} + x * t_{high}$

Wir gehen zusätzlich davon aus, dass für jeden der 3 Fälle ein eigener durchschnittlicher Informationszugewinn pro Member existiert. Wobei bei Fall 2 und 3 davon ausgegangen wird, dass nur die Berechnung eines High-Fidelity Members einen Informationsgewinn bringen kann. Dieser Informationsgewinn kann aber durch Low-Fidelity Member höher ausfallen, als bei der reinen High-Fidelity Optimierung.

	Fall 1	Fall 2	Fall 3
Gesamtzeit	$n * t_{high}$	$n * t_{low} + x * t_{high}$	$(n - x)\, t_{low} + x * t_{high}$
Mittl. Inf. Gewinn/Member	I_1	I_2	I_3
Ges. Informationszugewinn	$n * I_1$	$x * I_2$	$x * I_3$
Ges. Informationsgewinn/Zeit	$\frac{n*I_1}{n*t_{high}} = \frac{I_1}{t_{high}}$	$\frac{I_2*x}{n*t_{low}+x*t_{high}}$	$\frac{I_3*x}{(n-x)t_{low}+x*t_{high}}$

Tabelle 3.1: Drei mögliche Strategien für die

Um diese Fälle zu vergleichen, nehmen wir Fall 1 als Referenzfall und vergleichen die Informationsgehalt zu Zeit Verhältnisse miteinander. Als erstes Fall 1 und Fall 2:

$$\frac{I_1}{t_{high}} = \frac{I_2 * x}{n * t_{low} + x * t_{high}}$$

Stellt man nach $\frac{I_2}{I_1}$ um, so erhält man folgende Gleichung:

$$\frac{I_2}{I_1} = \frac{n}{x}\frac{t_{low}}{t_{high}} + 1$$

Diese Gleichung sagt aus, wie viel mehr Informationsgehalt pro berechneten High-Fidelity Member benötigt wird, damit Fall 2 und Fall 1 dasselbe Informationsgehalt zu Zeit Verhältnis erhalten. Das bedeutet, dass die x High-Fidelity Member die erzeugt werden, durch die vorher schon berechneten Low-Fidelity Member den Faktor $\frac{I_2}{I_1}$ mehr an Information liefern müssen, damit die Optimierungen im Fall 1 und Fall 2 gleichwertig sind. Analoges gilt für Fall 1 und Fall 3:

$$\frac{I_1}{t_{high}} = \frac{I_3 * x}{(n-x)\, t_{low} + x * t_{high}}$$

$$\frac{I_1}{I_3} = \frac{t_{high} * x}{(n-x)\, t_{low} + x * t_{high}}$$

$$\frac{I_3}{I_1} = \frac{(n-x)\, t_{low} + x * t_{high}}{t_{high} * x}$$

$$\frac{I_3}{I_1} = \frac{(n-x)\, t_{low}}{x * t_{high}} + 1$$

$$\frac{I_3}{I_1} = \frac{n}{x}\frac{t_{low}}{t_{high}} - \frac{t_{low}}{t_{high}} + 1$$

Man kann sehen, dass das Verhältnis bei Fall 3 grundsätzlich kleiner ausfällt und zwar um die Differenz von $\frac{t_{low}}{t_{high}}$. Wenn die Low-Fidelity Prozesskette also deutlich schneller ist, als die High-Fidelity Prozesskette, so ist der Unterschied zwischen den Strategien vernachlässigbar. Als Werte einer CFD-Optimierung wären bspw. die Erzeugung von

1000 Member denkbar und davon ca. 100 mit der High-Fidelity Prozesskette, der zeitliche Unterschied könnte im Bereich von Faktor 2 liegen. Diese Werte sind natürlich nur beispielhaft und basieren auf Erfahrungswerten, selbstverständlich haben sie keine allgemeine Gültigkeit. In Fall 2 würde herauskommen, dass der Informationsgehalt pro High-Fidelity Member um den Faktor $\frac{I_2}{I_1} = 5$ größer sein müsste, als bei der reinen High-Fidelity Optimierung. Bei Fall 3 würde der Faktor $\frac{I_2}{I_1} = 4.5$ herauskommen. Wenn die Low-Fidelity Prozesskette um den Faktor 5 schneller wäre, würden die Faktoren $\frac{I_2}{I_1} = 2$, $\frac{I_3}{I_1} = 1.8$. Es ist natürlich sehr schwierig den Informationsgehalt pro Member oder den Faktor $\frac{I_2}{I_1}$ bzw. $\frac{I_3}{I_1}$ quantitativ zu bewerten, dies soll an dieser Stelle aber auch gar nicht geschehen. Zudem ist dies auch eine stark vereinfachte Betrachtung, da der Informationszugewinn im Laufe der Optimierung immer kleiner wird und ein Mittelwert daher eine starke Vereinfachung darstellt. Aber es kann einen Eindruck davon vermitteln, dass selbst mit solch relativ simplen Multi-Fidelity Strategien eine Beschleunigung erreicht werden kann. Man sieht zudem, dass diese Beschleunigung allerdings sehr stark von der Güte und Geschwindigkeit des Low-Fidelity Modells abhängt. Mit einer Strategie wie sie in Fall 3 angenommen wurde, kann man die Beschleunigung insbesondere für langsamere Low-Fidelity Prozessketten verbessern. Zudem spielt natürlich die Entscheidung, wann die High-Fidelity Prozesskette ausgeführt werden soll eine große Rolle, damit kann insbesondere der Informationsgehalt pro High-Fidelity Member erhöht werden. Ein nicht zu vernachlässigender Faktor ist natürlich auch immer der Umsetzungs- und Implementierungsaufwand und dieser ist bei solchen Verfahren vergleichsweise gering, was von Vorteil ist.

Im Weiteren sollen aber auch Verfahren vorgestellt werden, die durch Abschätzung/-Extrapolation des Informationsgehalt zu Zeit Verhältnis eine dynamische Entscheidung ermöglichen welche Fidelity verwendet werden soll und damit einer optimalen Lösung näher kommen.

3.6 Informationszugewinn modellieren

Die andere Möglichkeit (siehe Abbildung 3.6) ist es, ein Modell für den Informationszugewinn zu finden. Eine Entscheidungsfunktion sollte dann immer die Fidelity wählen, welche das größte Verhältnis von Informationszugewinn pro Zeit verspricht. Hierbei sind zwei Möglichkeiten denkbar:

Während der Optimierung auf dem Ersatzmodell Das Ergebnis und der Verlauf der Optimierung auf dem Ersatzmodell würde verändert durch die Entscheidungsfunktion.

Nach der Optimierung auf dem Ersatzmodell Der Entscheidungsfunktion wird also ein fester Ort übergeben und diese entscheidet dann welche Prozesskette ausgeführt wird.

Die erste Variante müsste in die Optimierung auf dem Ersatzmodell integriert werden. Während der Optimierung auf dem Ersatzmodell müsste also bewertet werden, inwiefern die Erzeugung eines Low- oder High-Fidelity Members auf dem Ersatzmodell die Optimierung positiv beeinflusst oder nicht. Dazu müsste man allerdings auch wissen, inwiefern die Erzeugung eines neuen Members während der Optimierung auf dem Ersatzmodell das Ersatzmodell selbst beeinflusst. Dieses Problem ist analytisch nur sehr schwer bis gar nicht lösbar und ob diese Variante zu besseren Ergebnissen führt als die zweite ist zudem fraglich.

In der Arbeit von Huang et. al. (siehe [20]) wurde versucht über eine heuristische Funktion den ortsabhängigen Einfluss der Fidelity Wahl auf das Expected Volume Gain abzuschätzen. Durch die Ortsabhängigkeit, verändert sich natürlich auch die gesamte Optimierung auf dem Ersatzmodell und damit auch der gesamte Optimierungsverlauf. Inwiefern ein heuristischer Ansatz immer zu einer Beschleunigung führt, kann nur schwer abgeschätzt werden.

Für die zweite Variante geht man also davon aus, dass aus der Optimierung auf dem Ersatzmodell bereits ein Ort $\vec{x} \in \mathbb{R}^n$ bekannt ist und eine Funktion, welche den Ort auf den Informationszugewinn abbildet:

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

Wobei es für jede Fidelity jeweils nur einen Informationszugewinn gibt, jedoch viele Zielfunktionale. Zusätzlich benötigt man eine Entscheidungsfunktion:

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

Diese Funktion wählt den Member mit dem größten Verhältnis an Informationszugewinn pro Zeit. Um den Informationszugewinn besser modellieren zu können, ist es sinnvoll, einige Randbedingungen für diesen zu definieren. Durch die Wahl der ortsabhängigen Kovarianzfunktionen (siehe Kapitel **??**) ergeben sich einige Randbedingungen für den Informationszugewinn eines neuen Ortes $\vec{x}_0$:

1. Falls $\forall cov_{err}\left(\vec{x_0}, \vec{y}\right) = 0$, sollte gelten: $I_{high}\left(\vec{x}_0\right) = 0$
2. Falls $\forall cov_{low}\left(\vec{x_0}, \vec{y}\right) = 0$, sollte gelten: $I_{low}\left(\vec{x}_0\right) = 0$
3. Ist der Ort bereits Low-Fidelity gerechnet worden, sollte gelten: $I_{low}\left(\vec{x}_0\right) = 0$
4. Sobald ein neuer Low-Fidelity Member für die weitere Optimierung keinen weiteren Fortschritt mehr bringen wird, soll High-Fidelity berechnet werden.
5. Da es in der Regel mehrere Zielfunktionale für eine Fidelity gibt, müssen diese Zielfunktionale auf einen Informationszugewinn abgebildet werden können.

Hierzu einige Erläuterungen:

1. Beim Start einer Optimierung werden zuerst einige High- und Low-Fidelity Member zufällig erzeugt. Bei der Erzeugung neuer Member auf dem Ersatzmodell ist also bereits ein CO-Kriging Modell vorhanden. Das Training der Modelle kann dazu führen, dass eine der beiden Kovarianzfunktionen ausgeschaltet wird. Dies passiert dann, wenn die Low- und High-Fidelity Daten vollkommen unkorreliert sind oder das High-Fidelity Modell komplett durch das Low-Fidelity Modell beschrieben werden kann, was bei einer konstanten Verschiebung des Funktionswertes der Fall wäre. Die Entscheidungsfunktion sollte also nur noch Low-Fidelity Member erzeugen, da diese schneller zu berechnen sind und die High-Fidelity Daten keinen Informationszugewinn bringen.

2. Entspricht prinzipiell dem ersten Fall, es sollten aber nur noch High-Fidelity Member gerechnet werden, da das Low-Fidelity Modell keinen Informationszugewinn liefert.

3. Allgemein wird in einer automatisierten Optimierung ein Ranking der bereits berechneten Member erstellt, z.B. durch ein Pareto Ranking oder durch den Volumenzugewinn eines neuen Members bezogen auf die Paretofront. In der Regel würde man bei einem solchen Ranking nur die High-Fidelity Member einbeziehen, da die Low-Fidelity Member eine völlig andere Position im Fitnessraum aufweisen können. Z.B. könnten alle Low-Fidelity Fitnesswerte zu einem schlechteren Wert hin verschoben sein und wären damit nur sehr schwer mit den High-Fidelity Membern vergleichbar. Optimiert man nun auf dem Ersatzmodell und bezieht die Low-Fidelity Member nicht mit in das

Ranking ein kann es passieren, dass mehrfach hintereinander ein Low-Fidelity Member an dieselbe Stelle gelegt wird, da dieser nicht in das Ranking miteinbezogen wird. Um dies zu vermeiden, muss der Informationszugewinn bei Wahl der Low-Fidelity Prozesskette klein werden. Zudem sollte der Informationszugewinn auch kleiner werden, wenn sich bereits Low-Fidelity Punkte in unmittelbarer Nähe zu dem neuen Punkt $\vec{x}_0$ befinden. Die Kovarianzfunktion des Co-Kriging Modells wäre z.B. eine geeignete Funktion dafür.

4. Um diesen Punkt zu erläutern, betrachten wir als Beispiel eine fast auskonvergierte Multi-Fidelity-Optimierung die der Einfachheit halber nur eine Zielfunktion hat. Die noch zu erreichenden Fortschritte in der Zielfunktion sind also nur noch sehr klein. Zusätzlich soll angenommen werden, dass die Low-Fidelity Funktion eine stark geglättete Variante der High-Fidelity Funktion ist. Die High-Fidelity Funktion hingegen soll eine sehr raue Funktion sein, welche hoch aufgelöst werden muss, um diese gut interpolieren zu können. In diesem Fall könnte man gegen Ende der Optimierung das CO-Kriging Ersatzmodell nicht mehr durch zusätzliche Low-Fidelity Daten verbessern, da diese die benötigte Information einfach nicht mehr beinhaltet. Aus diesem Grund ist es erforderlich, dass der Informationszugewinn des Low-Fidelity Modells in einem solchen Fall gegen Null geht. Dies setzt allerdings eine Schätzung des Informationszugewinns im nächsten Schritt voraus.

5. Erfahrungsgemäß haben die meisten Optimierungen mehr als nur ein Zielfunktional und oftmals sind die Zielfunktionale aus mehreren Funktionen zusammengesetzt. Beispielsweise ist der isentrope Wirkungsgrad zusammengesetzt aus dem Totaldruckverhältnis und dem Totaltemperaturverhältnis. Der Wirkungsgrad könnte also durch zwei Ersatzmodelle bestimmt werden, anstatt durch ein Ersatzmodell, welches direkt auf den Wirkungsgrad trainiert wird. Es muss also aus mehreren zusammengesetzten Zielfunktionalen ein Informationszugewinn pro Fidelity abgebildet werden können.

3.6.1 Entscheidungsfunktion nur abhängig vom Ersatzmodell

3.6.1.1 Varianz des Fehlers als Gütemaß

Varianzvorhersage, Informationsmaß wäre Varianzvorhersage HighFid und Varianzvorhersage LowFidelity-Varianzvorh High Fidelity

$$crit = \frac{t_{train} + t_{optiOnMM} + t_{high}}{t_{train} + t_{optiOnMM} + t_{low}} * \sum^{Modelle} \frac{\sigma_{high} - \sigma_{high+low}}{\sigma_{high} - 0}$$

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 3.2:

Was noch fehlt wäre die Gewichtung der einzelnen Ersatzmodelle, da jedes Modell ja für die Optimierung unterschiedliche wichtig ist, sollte man hier eigentlich unterscheiden !!!

dingt möchte. Zudem müsste man die Likelihood Funktion neu optimieren

3.6.1.2 Globale Varianzreduktion

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:

$$f_{high} = 2x + 0.1 \sin(20x)$$

$$f_{low} = 0.1 \sin(20x) + 0.2$$

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.

Formales Kriterium

Für ein Ersatzmodell:

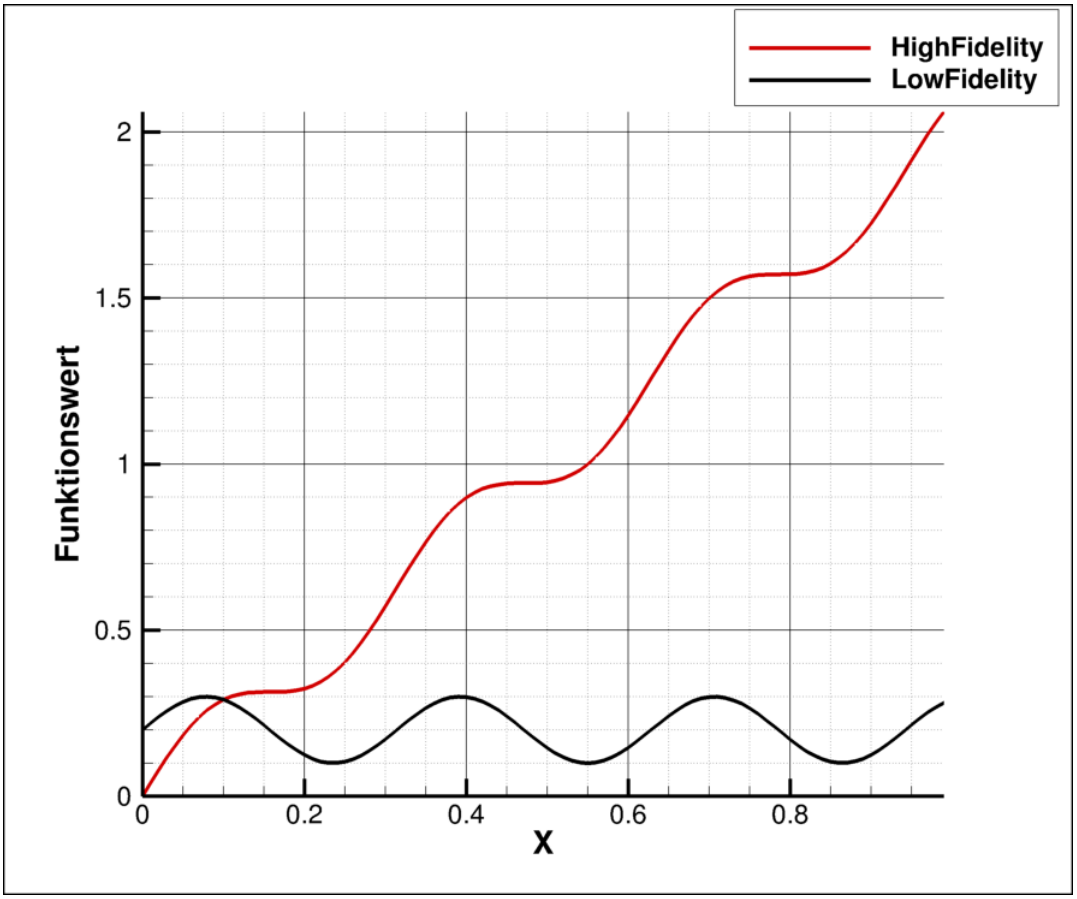


Abbildung 3.7:

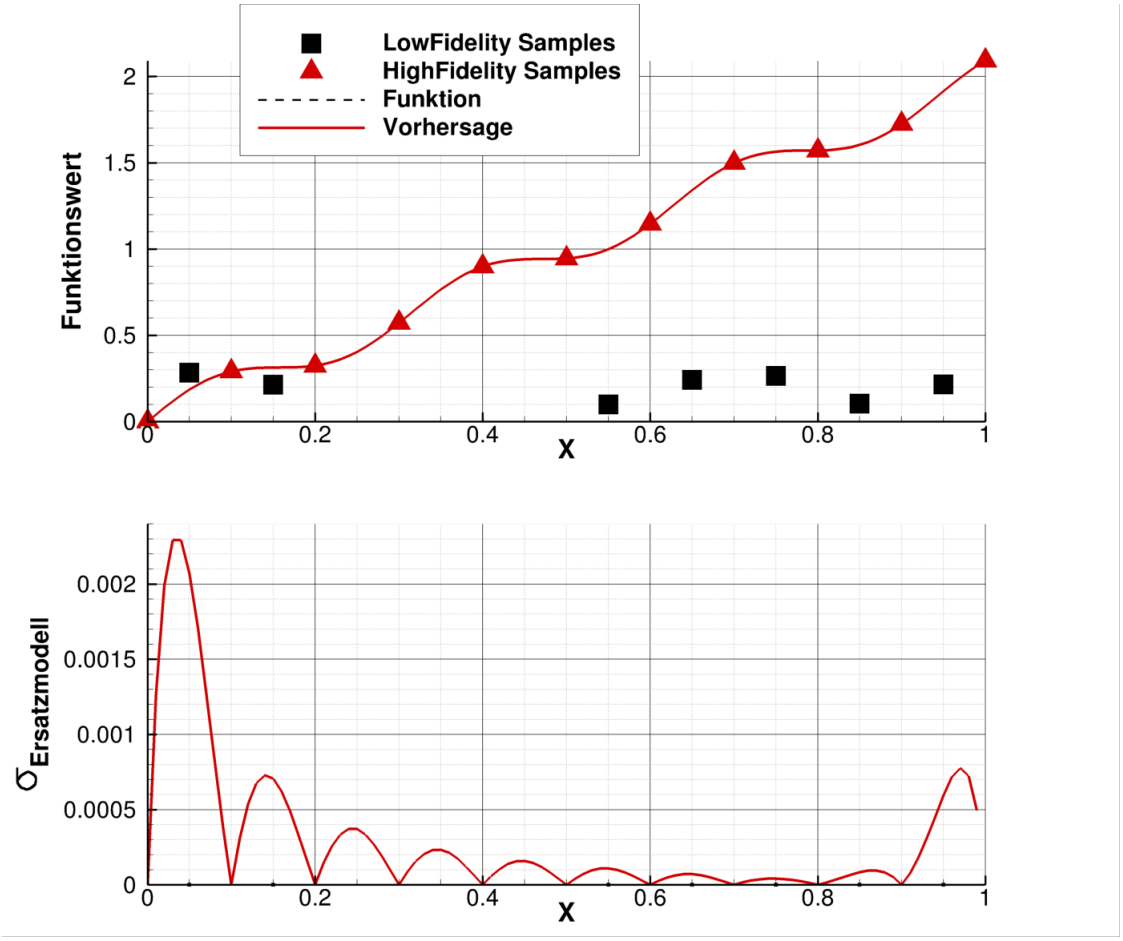


Abbildung 3.8:

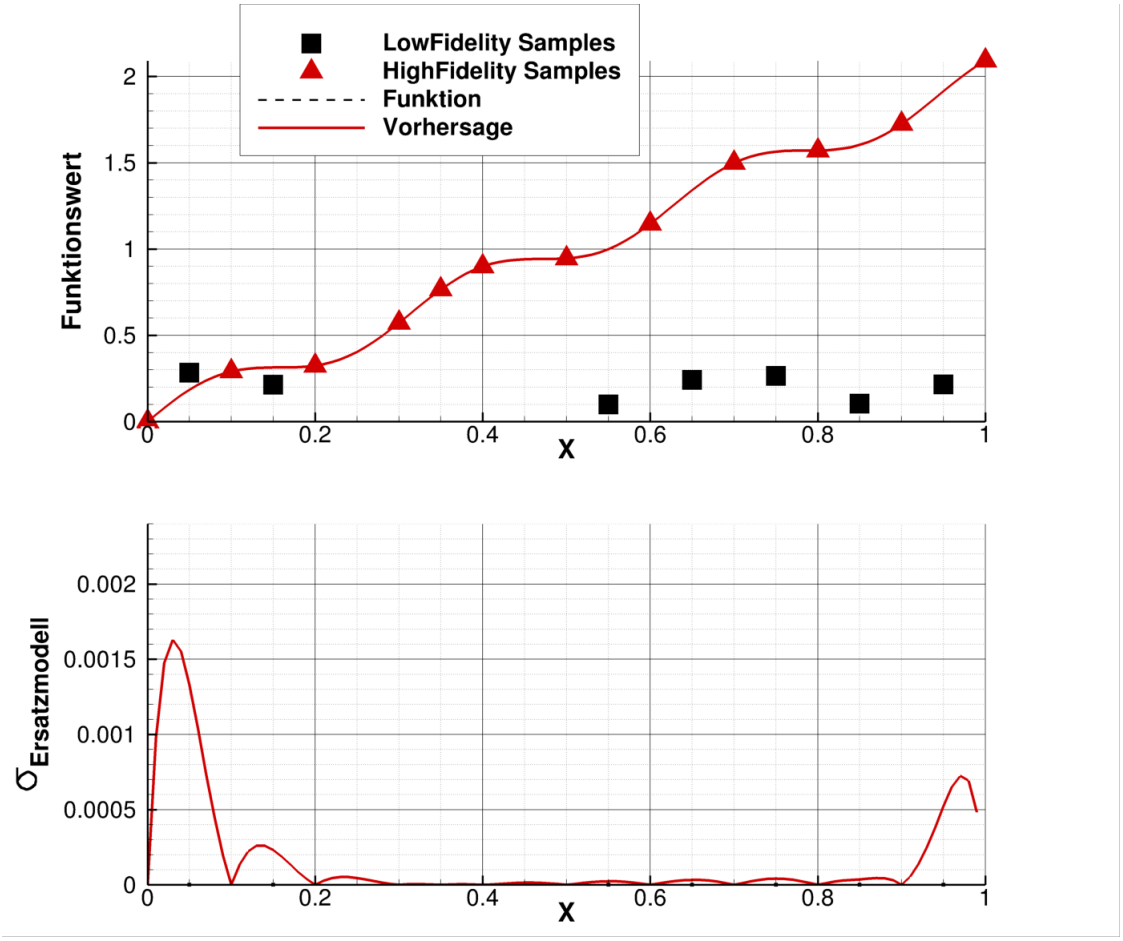


Abbildung 3.9:

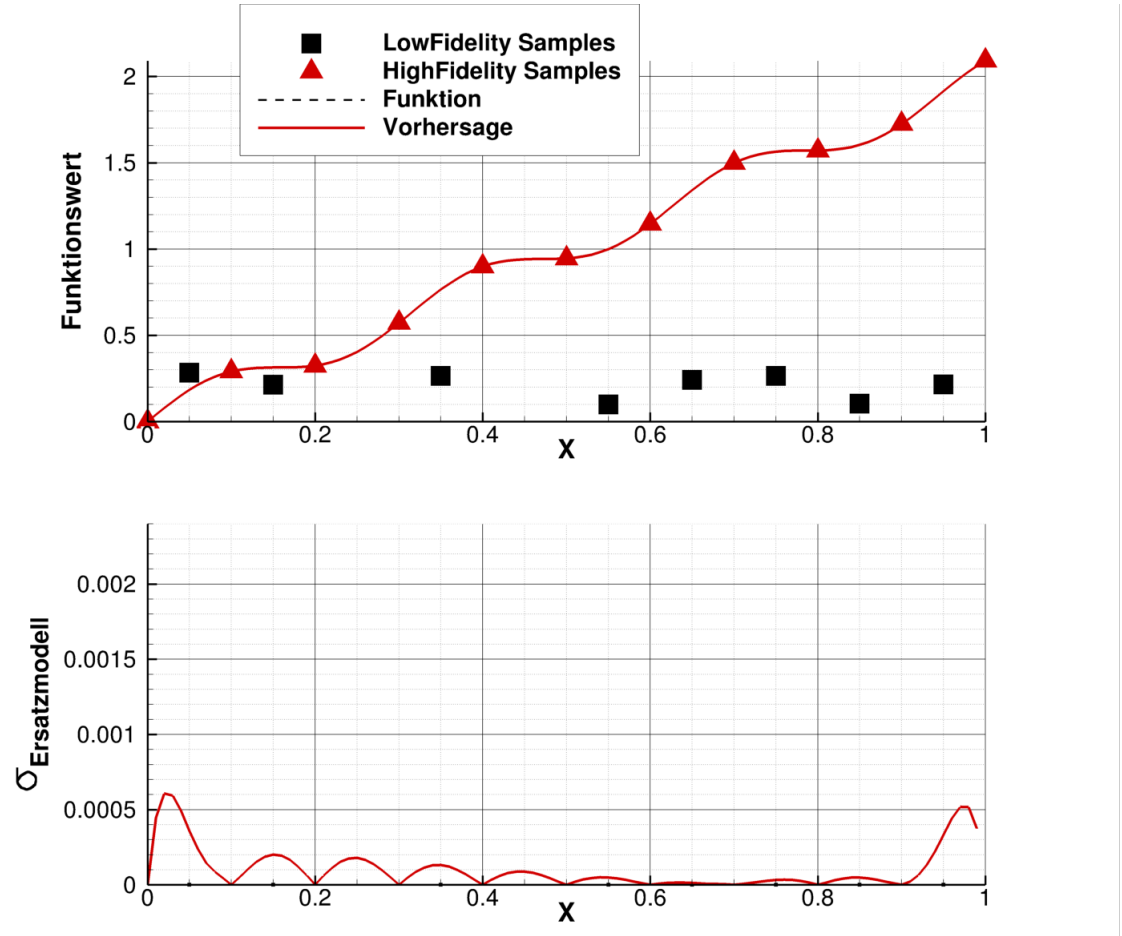


Abbildung 3.10:

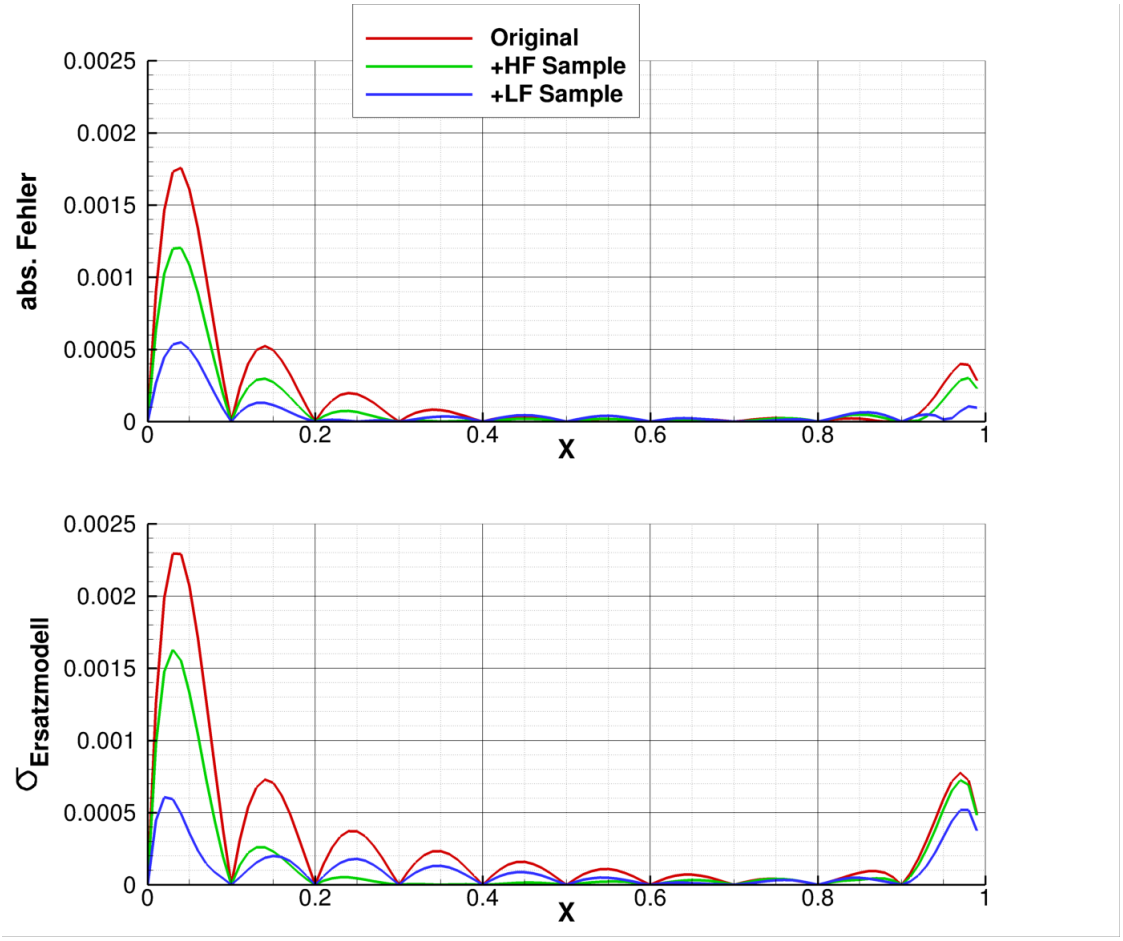


Abbildung 3.11:

$$\triangle H = \int \left(\sigma_H^2\left(\vec{x}\right) - \sigma_H^2\left(\vec{x}\right) \mid y_H\left(\vec{x}_0\right)\right) d\vec{x}$$

$$\triangle L = \int \left(\sigma_H^2\left(\vec{x}\right) - \sigma_H^2\left(\vec{x}\right) \mid y_L\left(\vec{x}_0\right)\right) d\vec{x}$$

Die relative Änderung sähe dann folgendermaßen aus:

$$\triangle H_{rel} = \int \left(\frac{\sigma_H^2\left(\vec{x}\right) - \sigma_H^2\left(\vec{x}\right) \mid y_H\left(\vec{x}_0\right)}{\sigma_H^2\left(\vec{x}\right) - \sigma_H^2\left(\vec{x}\right) \mid y_L\left(\vec{x}_0\right)}\right) dx$$

Für mehrere Ersatzmodelle relative Varianzreduktion:

$$crit = \frac{t_{train} + t_{optiOnMM} + t_{high}}{t_{train} + t_{optiOnMM} + t_{low}} * \sum^{Modelle} \int \left(\frac{\sigma_H^2\left(\vec{x}\right) - \sigma_H^2\left(\vec{x}\right) \mid y_H\left(\vec{x}_0\right)}{\sigma_H^2\left(\vec{x}\right) - \sigma_H^2\left(\vec{x}\right) \mid y_L\left(\vec{x}_0\right)}\right) dx$$

Die Schwierigkeit liegt in der numerischen Berechnung der Integrale, hierfür bietet sich eine Monte Carlo Integration an.

3.6.2 Entscheidungsfunktion über Volumenzugewinn

In diesem Abschnitt soll im Detail darauf eingegangen werden, welche Möglichkeiten der Membererzeugung innerhalb einer Multifidelity Optimierung bestehen, wenn man den Informationszugewinn modellieren möchte. Um den Informationszugewinn zu modellieren bzw. zu schätzen, benötigen wir zuerst ein geeignetes Maß für diesen. Als bisher gutes Maß für den aktuellen Optimierungsfortschritt hat sich bisher der kumulierte Volumenzugewinn herausgestellt. Ein großer Vorteil der für dieses Maß spricht ist, dass mehrere Zielfunktionale sinnvoll bewertet werden können. Für weitere Informationen sei der Leser auf folgende Literatur verwiesen: [10, 18].

In Abbildung 3.12 ist diese Möglichkeit grün markiert. Die nächste Frage die sich stellt, ist wie man den Informationszugewinn schätzen kann. In diesem Fall muss also konkret eine Schätzung über den Volumenzugewinn im nächsten Optimierungsschritt angestellt werden. Das in AutoOpti umgesetzte Verfahren zur Membererzeugung, kann mit den Ersatzmodellvorhersagen einen erwarteten Volumenzugewinn berechnen. Um einen neuen Member zu erzeugen, wird dieser erwartete Volumenzugewinn dann maximiert. Als Ergebnis erhält man einen Parametersatz/Ort mit dem höchsten erwarteten Volumenzugewinn. Wir gehen also davon aus, dass der Ort bereits durch eine Optimierung auf dem Ersatzmodell gegeben ist und sich in der Datenbasis bereits einige Low- und High-Fidelity Member befinden. Um sich einer geeigneten Schätzung

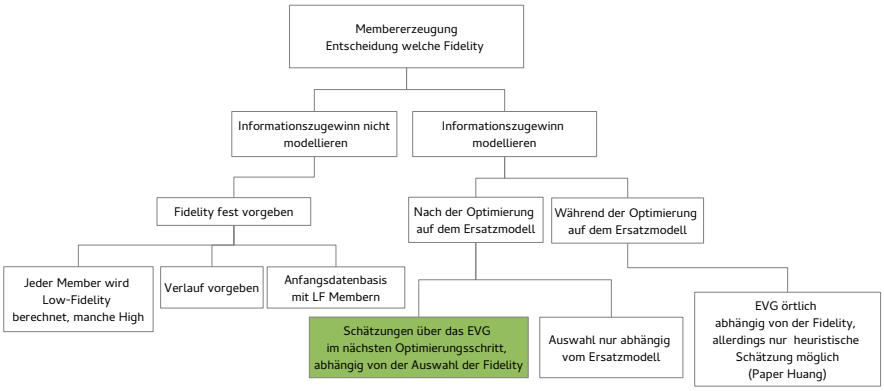


Abbildung 3.12: Möglichkeiten der Membererzeugung mit mehreren Fidelities

zu nähern, soll zuerst eine ideale aber unpraktikable Lösung betrachtet werden, die dann durch einige Annahmen und Einschränkungen zur Anwendung gebracht werden soll. Es stellt sich außerdem noch die Frage, in welchem Raum (z.B. Low- oder High-Fidelity Raum) die Zielfunktionale der Optimierung betrachtet werden sollen. Es erscheint sinnvoll, die Zielfunktion(en) im High-Fidelity Raum zu betrachten und die Optimierung auf dem Ersatzmodell ebenfalls im High-Fidelity Raum durchzuführen. Die Verbindung zwischen High- und Low-Fidelity soll also nur über das Ersatzmodell erfolgen.

Angenommen man könnte den gesamten Verlauf einer Multi-Fidelity Optimierung mit 2 Fidelities vom Anfang bis zum Ende voraussagen, wobei der Optimierungsfortschritt durch den kumulierten Volumenzugewinn im High-Fidelity Raum definiert ist. Es werden n Member erzeugt und für jeden Member i hätte man die Wahl, ob dieser mit der Low- oder High-Fidelity Prozesskette berechnet werden soll. Aus diesen Überlegungen heraus gäbe es 2^n mögliche Optimierungsverläufe, wobei jeder dieser Verläufe eine eigene Zeit t_i benötigt und einen kumulierten Volumenzugewinn von V_i erreicht hat. Die Berechnung eines Low-Fidelity Members soll immer die gleiche Zeit t_{low} benötigen und die eines High-Fidelity Members die Zeit t_{high}. Außerdem soll gelten, dass

$t_{low} < t_{high}$. Daraus ergeben sich zusätzlich die minimal mögliche Zeit $t_{min} = \sum^{n} t_{low}$ und die maximal mögliche Zeit $t_{max} = \sum^{n} t_{high}$. Es muss nun noch eine Annahme darüber getroffen werden, welcher der 2^n Optimierungsverläufe als ideal angesehen wird. Möglich wäre natürlich die Wahl des Optimierungverlaufs mit dem größten kumulierten Volumenzugewinn. Allerdings würde die benötigte Zeit völlig außer Acht gelassen. Daher erscheint es sinnvoll zu sagen, dass der Optimierungsverlauf i_{best} als ideal angenommen wird, der die größte Steigung des kumulierten Volumenzugewinns aufweist:

$$i_{best} = \left\{ \begin{array}{l} max\left(\frac{V_i}{t_i}\right) \\ i \in \{1; n\} \end{array} \; | t_{min} \leq t_i \leq t_{max} \right\}$$

Selbstverständlich können wir unmöglich einen Verlauf, geschweige denn alle möglichen Verläufe einer Optimierung vorhersagen. Wenn dies möglich wäre, bräuchten man nicht mehr zu optimieren. Also schränken wir diesen idealisierten Fall weiter ein und versuchen eine Abschätzung darüber zu bekommen, inwiefern sich die Wahl der Prozesskette auf die nächsten ein bis zwei Schritte auswirken. Um dies zu bewerkstelligen, müssen wir wissen wie stark sich das Ersatzmodell verbessert, wenn wir den nächsten Member High- oder Low-Fidelity berechnen und dann dem Ersatzmodell hinzufügen. Gehen wir also davon aus, dass wir uns innerhalb einer Optimierung befinden und den nächsten zu berechnenden Member bestimmen möchten. Hierfür wird eine Optimierung auf dem Ersatzmodell durchgeführt, welche dann einen Ort für den nächsten zu berechnenden Member vorschlägt. An diesem Zeitpunkt oder Optimierungsschritt j stellt sich nun die Frage, ob dieser vorgeschlagene Member nun mit der High- oder Low-Fidelity Prozesskette berechnet werden soll. Um nun zu bestimmen, welche Auswirkungen die Wahl auf den nächsten Optimierungsschritt hat, wären theoretisch folgende Schritte notwendig:

1. Aus der Optimierung auf dem Ersatzmodell wird ein Ort vorgegeben.
2. Member Low-Fidelity berechnen
3. Das Ersatzmodell mit dem neuen Low-Fidelity Member trainieren
4. Das globale Optimum für den erwarteten Volumenzugewinn auf dem Ersatzmodell finden
5. Member High-Fidelity berechnen
6. Das Ersatzmodell mit dem neuen High-Fidelity Member trainieren
7. Das globale Optimum für den erwarteten Volumenzugewinn auf dem Ersatzmodell finden

8. Die Prozesskette mit dem maximalen Volumenzugewinn pro Zeit wählen

Diese Lösung ist natürlich immer noch unpraktikabel, da man in diesem Fall beide Prozessketten durchlaufen muss (Punkte 2 und 5) und damit jeglichen Zeitvorteil verliert. Außerdem kann man das globale Optimum des Volumenzugewinns (Punkte 4 und 7) nur mit großem Zeitaufwand bestimmen. Den Punkt 5-7 hat man bei der Verwendung des Expected-Volume-Gain bereits abgedeckt, hierfür wird auf dem Ersatzmodell optimiert und ein Member vorgeschlagen, der den höchsten zu erwartenden Volumenzugewinn im High-Fidelity Zielfunktionsraum verspricht. Da dieses Thema schon in anderen Arbeiten behandelt wurde (siehe [10, 18]), beschäftigen wir uns im Weiteren nur noch mit der Modellierung von Punkt 2-4. Hierfür müssen wir weitere Annahmen und Vereinfachungen treffen. Da die Zielfunktion nur im High-Fidelity Raum bewertet werden soll, bringt ein neuer Low-Fidelity Member zuerst keinen direkten Volumenzugewinn. Vielmehr möchte man mit einem neuen Low-Fidelity Member das Ersatzmodell verbessern und damit den Suchraum weiter einschränken. Oder etwas einfacher formuliert, möchte man Dadurch kann der Volumenzugewinn in den darauffolgenden Schritten gesteigert werden. Betrachtet man z.B. einen Extremfall, bei dem es nur eine Zielfunktion gibt und sich Low- und High-Fidelity Member nicht unterscheiden. Die Low-Fidelity Prozesskette kann aber schneller berechnet werden. In diesem Fall könnte es darauf hinauslaufen, dass man die gesamte Optimierung ausschließlich Low-Fidelity Member erzeugt und nur den letzten Member mit der High-Fidelity Prozesskette berechnet. Durch das Ersatzmodell, könnte der Zielfunktionsraum so stark eingegrenzt worden sein, dass der letzte High-Fidelity Member exakt das Minimum trifft. Voraussetzung wäre natürlich eine entsprechend genaue und schnelle Low-Fidelity Prozesskette und dass es nur ein Minimum gibt. Jeder andere Optimierungsverlauf wäre in diesem Fall nicht optimal gewesen, da man insgesamt mehr Zeit verbraucht hätte. Im Folgenden sollen einige Möglichkeiten besprochen werden, um die Punkte 2-4 zu modellieren.

Modellierung Punkt 2: Member Low-Fidelity berechnen

Diesen Punkt zu modellieren ist relativ schwierig bzw. aufwendig, da man bereits vor der Durchführung der Low-Fidelity Prozesskette die Entscheidung treffen möchte, welche Prozesskette aufgerufen wird. Eine mögliche Lösung für dieses Problem, ist die Verwendung des Ersatzmodells um den Low-Fidelity Member abzuschätzen. Das hier vorgestellte CO-Kriging Ersatzmodell ist in der Lage, sowohl Low- als auch High-Fidelity Vorhersagen zu treffen (siehe die Kapitel **??** und **??**). Statt nun einen Low-Fidelity Member mit der Prozesskette zu berechnen, kann man diesen für die Entscheidung erst mit dem Co-Kriging Modell vorhersagen. Das Problem daran ist, dass man eine Verteilung als Antwort bekommt, also einen Erwartungswert $E\left[\vec{x}_0\right]$ an der Stelle $\vec{x}_0$ und eine Varianz des Fehlers $var\left[F\left(\vec{x}_0\right)\right]$ an derselben Stelle. Dies wie-

derum bedeutet, dass sich unser echter Low-Fidelity Wert innerhalb dieser Verteilung bewegen wird, es gibt also eine Unsicherheit bei dem Low-Fidelity Wert. Es stellt sich dann natürlich die Frage, wie man einen Member mit in das Ersatzmodell aufnehmen kann, der mit einer Unsicherheit behaftet ist und zudem ja auch aus dem Ersatzmodell stammt. Dies erscheint erst einmal unsinnig, da man ja keine echte neue Information in das Ersatzmodell steckt.

Lassen wir diesen Punkt zuerst außen vor und beschäftigen uns mit der Frage, wie man diese Verteilung mit in das Ersatzmodell aufnehmen kann. Es gäbe zunächst die Möglichkeit, einfach den Erwartungswert mit in das Ersatzmodell aufzunehmen, da dies der wahrscheinlichste Wert ist, in diesem Fall lässt man die Unsicherheit allerdings komplett unbeachtet.

Eine andere Möglichkeit ist es, die Verteilung einfach zu simulieren, man erzeugt also zufällig viele Member aus der Low-Fidelity Verteilung und verwendet diese dann für das Ersatzmodell. Der Nachteil liegt hierbei natürlich darin, dass man dieses Vorgehen sehr oft durchführen muss. Grundlegend erscheint dieses Vorgehen aber zunächst sinnvoll, wie genau die Samples bei der Entscheidung verwendet werden können, soll in den nächsten Schritten erklärt werden.

Eine sinnvolle Strategie zur Erzeugung solcher Samples aus einer Normalverteilung ist die Box-Muller Methode (siehe [2])

Das Training der Ersatzmodelle in den Punkten 3 und 6 möchte man ebenfalls weitestgehend vermeiden.

Einschub: Box Muller Methode Mit dieser Methode kann man aus zwei unabhängigen gleichverteilten Zufallszahlen (U_1, U_2), zwei standardnormalverteilte Zufallszahlen(Z_1, Z_2) generieren. Die zwei gleichverteilten Zufallszahlen kann man bspw. aus einem Zufallszahlengenerator gewinnen, solche Generatoren sind z.B. in der Boost Bibliothek verfügbar (siehe [21]).

1. Aus der Optimierung auf dem Ersatzmodell wird ein Ort vorgegeben.
2. Schätzung des Expected Volume Gain falls der neue Member als Low- oder High-Fidelity berechnet wird.
3. Der Informationszugewinn wäre dann das geschätzte Expected Volume Gain für jede Fidelity

Oder

1. Aus der Optimierung auf dem Ersatzmodell wird ein Ort vorgegeben.

2. Schätzung über die Verbesserung des Ersatzmodells, wenn der Member als Low-Fidelity hinzugefügt würde
3. Schätzung über die Verbesserung des Ersatzmodells, wenn der Member als High-Fidelity hinzugefügt würde
4. Der Informationszugewinn wäre dann die Verbesserung des Ersatzmodells

Da die Modelle der Kovarianzfunktionen in diesem Kriging Verfahren nur Abhängig vom Ort sind, ist man in der Lage die Kovarianzen zwischen dem neuen Member und allen bereits vorhandenen zu bilden.

Hierbei hat man zwei mögliche Annahmen, der neue Member ist ein High Fidelity Member oder ein Low Fidelity Member. Es kämen jeweils andere Kovarianzvektoren heraus.

Nimmt man bspw an, man hat ein bereits vorhandenes Modell mit 2 High- und 2 Low-Fidelity Membern und einen neu erzeugten Member $\vec{x}_0$.

$$\vec{x} = \begin{pmatrix} \vec{x}_{1h} \\ \vec{x}_{2h} \\ \vec{x}_{3l} \\ \vec{x}_{4l} \end{pmatrix}$$

Nimmt man an, dass der neue Vektor ein Low Fidelity Member ist, ergibt sich folgender Kovarianzvektor:

$$\vec{cov}\left(\vec{x}_{0h}, \vec{x}\right) = \begin{pmatrix} \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0h}, \vec{x}_{1h}\right) + \mathrm{cov}_{\mathrm{err}}\left(\vec{x}_{0h}, \vec{x}_{1h}\right) \\ \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0h}, \vec{x}_{2h}\right) + \mathrm{cov}_{\mathrm{err}}\left(\vec{x}_{0h}, \vec{x}_{2h}\right) \\ \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0h}, \vec{x}_{3l}\right) \\ \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0h}, \vec{x}_{4l}\right) \end{pmatrix}$$

Im anderen Fall:

$$\vec{cov}\left(\vec{x}_{0l}, \vec{x}\right) = \begin{pmatrix} \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0l}, \vec{x}_{1h}\right) \\ \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0l}, \vec{x}_{2h}\right) \\ \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0l}, \vec{x}_{3l}\right) \\ \mathrm{cov}_{\mathrm{low}}\left(\vec{x}_{0l}, \vec{x}_{4l}\right) \end{pmatrix}$$

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.

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 [22] basiert. Das Verfahren wurde später nach ihm benannt. Der französische Mathematiker Georges Matheron (1963) entwickelte aus Kriges Ar-

beit, schließlich die Kriging Theorie[23]. 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 [24], 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 [25], 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 [24], 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 [26].

- 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 [27, 28] ,[29] und [26]. Die Unterschiede des hier vorgestellten Ansatzes zu den anderen Arbeiten werden in Kapitel 4.8 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 $\vec{x} \in \mathbb{R}^k$ 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 $\vec{x} \in \mathbb{R}^k$ 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.

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

$$\Leftrightarrow E [Z (\vec{x}_0)] - E [Z_k^* (\vec{x}_0)] = 0$$

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

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

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

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

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:

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

Bestimmung der Gewichte

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

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

Im Anhang A.1 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 $\overrightarrow{\mathrm{cov}} \in \mathbb{R}^{\mathrm{n_{all}}}$ den Kovarianzvektor darstellt:

$$\mathrm{var}\left[F\left(\vec{x}_0\right)\right] = \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - 2\overrightarrow{\mathrm{cov}}\left(Z\left(\vec{x}_0\right), Z\left(\vec{x}\right)\right)^T * \vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \tag{4.6}$$

Damit lässt sich folgende Minimierungsaufgabe mit Nebenbedingung stellen:

$$\begin{cases} \min_{w_1,...,w_n} \mathrm{var}\left[Z\left(\vec{x}_0\right) - \sum_{j=1}^{n_{all}} w_i Z\left(\vec{x}_i\right)\right] \\ \beta - \sum_{i=1}^{m_f}\beta_i\sum_{j=1}^{n_i} w_{i,j} = 0 \end{cases} \tag{4.7}$$

Mit dieser Formulierung kann nun die Minimierungsaufgabe aus Gleichung **??** gemäß

der Multiplikatorenmethode von Lagrange gelöst werden [30]. Als Lagrange Funktion $\Lambda(w, \lambda)$ mit dem Lagrange Multiplikator λ ergibt sich:

$$\Lambda(w, \lambda) := \mathrm{var}\left[F(\vec{x}_0)\right] + \lambda\left(\beta - \sum_{i=1}^{m_f}\beta_i \sum_{j=1}^{n_i} w_{i,j}\right)$$

$$\nabla_{\lambda,w}\left(\mathrm{var}\left[F(\vec{x}_0)\right] + \lambda\left(\beta - \sum_{i=1}^{m_f}\beta_i \sum_{j=1}^{n_i} w_{i,j}\right)\right) = \vec{0}$$

Daraus lässt sich das folgende Gleichungssystem ableiten:

$$\nabla_w\left(\mathrm{var}\left[Z(\vec{x}_0)\right] - 2\overrightarrow{\mathrm{cov}}\left(Z(\vec{x}_0), Z(\vec{x}_i)\right)^T * \vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} + \lambda\left(\beta - \sum_{i=1}^{m_f}\beta_i \sum_{j=1}^{n_i} w_{i,j}\right)\right) = \vec{0}$$

$$\wedge\ \nabla_\lambda\left(\mathrm{var}\left[Z(\vec{x}_0)\right] - 2\overrightarrow{\mathrm{cov}}\left(Z(\vec{x}_0), Z(\vec{x}_i)\right)^T * \vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} + \lambda\left(\beta - \sum_{i=1}^{m_f}\beta_i \sum_{j=1}^{n_i} w_{i,j}\right)\right) = 0 \tag{4.8}$$

Es werden folgende Variablen eingeführt:

$$\mathrm{var}\left[Z(\vec{x}_0)\right] = \sigma_r^2$$

$$\overrightarrow{\mathrm{cov}}\left(Z(\vec{x}_0), Z(\vec{x}_i)\right) = \vec{c}$$

Damit ergibt sich:

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

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

Aufgelöst:

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

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

Zwischenschritt:

$$-2\vec{c}^T + 2\mathbf{Cov} * \vec{w} + \nabla_w \beta\lambda - \nabla_w \lambda \sum_{i=1}^{m_f} \beta_i \sum_{j=1}^{n_i} w_{i,j} = \vec{0}$$

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

Zwischenschritt:

$$-2\vec{c}^T + 2\mathbf{Cov} * \vec{w} - \nabla_w \lambda \sum_{i=1}^{m_f} \beta_i \sum_{j=1}^{n_i} w_{i,j} = \vec{0}$$

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

Zwischenschritt:

$$-2\vec{c}^T + 2\mathbf{Cov} * \vec{w} - \lambda\nabla_w \sum_{i=1}^{m_f} \beta_i \sum_{j=1}^{n_i} w_{i,j} = \vec{0}$$

AllVarsZero:1

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

Definiere $\vec{F}$:

$$\vec{F} = \begin{bmatrix} \beta_{1,1} \\ \vdots \\ \beta_{1,n_1} \\ \vdots \\ \beta_{m_f,n_{m_f}} \end{bmatrix}$$

Zwischenschritt:

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

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

Umgeformt:

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

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

In Matrix Schreibweise:

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

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

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

Die Inverse der Blockmatrix ergibt sich nach [31] 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:

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

Daraus ergibt sich folgende Formulierung der gesuchten Gewichte:

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

Einschub 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:

$$c(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sqrt{\mathrm{var}(X)}\sqrt{\mathrm{var}(Y)}} \tag{4.11}$$

Zudem soll der räumliche Zusammenhang zwischen zwei Zufallsvariablen nicht von seiner absoluten Lage im Raum abhängig sein, sondern nur von deren Abstandsvektor $h \in \mathbb{R}^n = \{x_1 - y_1, ..., x_n - y_n\}$

Die Kovarianz zwischen zwei gleichen Zufallsvariablen ist per Definition die Varianz dieser Zufallsvariable:

$$\mathrm{cov}(Z,Z) = \mathrm{var}(Z)$$

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.

$$\operatorname{cov}(Z, Z) = \operatorname{cov}\left(h = \vec{0}\right) = const.$$

Und somit ist auch die Varianz konstant:

$$\Longrightarrow \operatorname{var}[Z] = \sigma^2 = const. \tag{4.12}$$

Durch die Gleichungen 4.11 und 4.12 kann man die Kovarianzfunktion durch die Korrelationsfunktion ersetzen

$$c(X, Y)\,\sigma^2 = \operatorname{cov}(X, Y)$$

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 $\vec{r} \in \mathbb{R}^{\mathrm{n}} = (c[Z(\vec{x}_0), Z(\vec{x}_1)], ..., c[Z(\vec{x}_0), Z(\vec{x}_n)])^T, \vec{x} \in \mathbb{R}^{\mathrm{k}}$ bezeichnet wird. Dieser beschreibt die Korrelationen zwischen dem Funktionswert, der vorhersagt werden soll und dem Stützstellenvektor $\vec{y}_s$. Da der Funktionswert $y(x_0)$ 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 $\vec{r} \in \mathbb{R}^{\mathrm{n}} = (c[\vec{x}_0, \vec{x}_1], ..., c[\vec{x}_0, \vec{x}_n])^T, \vec{x} \in \mathbb{R}^{\mathrm{k}}$. Wie eine solche Korrelationsfunktion konkret umgesetzt wird, folgt in Kaptitel 5.1.2.

4.2.1 Allgemeine Kriging Vorhersage

4.2.1.1 Erwartungswert

Setzt man die Gewichte nun in Gleichung **??** ein, erhält man folgende Gleichung:

$$Z^*\left(\vec{x}_0\right) = \vec{c}^T \mathbf{Cov}^{-1} \vec{y}_s - \vec{c}^T \frac{\mathbf{Cov}^{-1} \vec{F} \vec{F}^T \mathbf{Cov}^{-1}}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}} \vec{y}_s + \frac{\vec{F}^T \mathbf{Cov}^{-1} \vec{y}_s}{\vec{F}^T \mathbf{Cov}^{-1} \vec{F}} \beta$$

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

$$Z^*\left(\vec{x}_0\right) = \vec{c}^T \mathbf{Cov}^{-1} \left(\vec{y}_s - \mu \vec{F}\right) + \mu\beta \tag{4.13}$$

An dieser Stelle stellt sich die Frage, inwiefern μ und der Likelihood Schätzer für β_i zusammenhängen.

$$\left(G^T \mathbf{Cov}^{-1} G\right)^{-1} \left(G^T \mathbf{Cov}^{-1} \overrightarrow{y}\right) = \vec{\tilde{\beta}}$$

Wobei $\vec{\tilde{\beta}} \in \mathbb{R}^{1 \times \mathrm{m_f}}$

Der Vektor $\vec{F} \in \mathbb{R}^{\mathrm{n} \times 1}$ lässt sich auch umformulieren zu, wobie $G \in \mathbb{R}^{\mathrm{n} \times \mathrm{m_f}}$:

$$\vec{F} = \begin{bmatrix} \vec{\beta}_1 \\ \vec{\beta}_2 \end{bmatrix} = G\vec{\tilde{\beta}} = G \left(G^T \mathbf{Cov}^{-1} G\right)^{-1} \left(G^T \mathbf{Cov}^{-1} \overrightarrow{y}\right)$$

$$\mu = \frac{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} \vec{y}_s}{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} G\vec{\tilde{\beta}}}$$

$$\mu = \frac{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} \vec{y}_s}{\left(G\vec{\tilde{\beta}}\right)^T \mathbf{Cov}^{-1} G\vec{\tilde{\beta}}}$$

$$\mu = \frac{\left(G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)\right)^T\mathbf{Cov}^{-1}\vec{y}_s}{\left(G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)\right)^T\mathbf{Cov}^{-1}G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)}$$

$$\mu = \frac{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}G^T\mathbf{Cov}^{-1}\vec{y}_s}{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}G^T\mathbf{Cov}^{-1}G\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)}$$

$$\mu = \frac{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}G^T\mathbf{Cov}^{-1}\vec{y}_s}{\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)^T\left(G^T\mathbf{Cov}^{-1}G\right)^{-1}\left(G^T\mathbf{Cov}^{-1}\overrightarrow{y}\right)}$$

$$\mu = 1$$

$$Z^*\left(\vec{x}_0\right) = \vec{c}^T\mathbf{Cov}^{-1}\left(\vec{y}_s - \vec{F}\right) + \beta \tag{4.14}$$

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.

Eine Änderung ist nur auf Ebene der Vorhersage Formeln notwendig. Geht man z.B. davon aus, dass zwei Fidelity Stufen verwendet werden, dann muss man zum Vorhersagen einer bestimmten Fidelity nur der Wert β mit dem entsprechenden Eintrag aus dem Vektor $\vec{\beta}$ (siehe Kapitel **??**) ersetzen. Außerdem muss der Vektor $\vec{c}$ entsprechend gewählt werden. Möchte man einen Punkt $\vec{x}_0$ im High-Fidelity Raum vorhersagen, dann muss dieser Punkt auch als High-Fidelity angesehen werden und die entsprechenden Kovarianzfunktion gewählt werden:

$$\vec{c} = \left[\begin{array}{c|c} & Z_{high}\left(\vec{x}_0\right) \\ \hline Z_{high}\left(\vec{x}_1\right) & cov\left(\vec{x}_1, \vec{x}_0\right) \\ Z_{high}\left(\vec{x}_2\right) & cov\left(\vec{x}_2, \vec{x}_0\right) \\ Z_{low}\left(\vec{x}_3\right) & cov\left(\vec{x}_3, \vec{x}_0\right) \\ Z_{low}\left(\vec{x}_4\right) & cov\left(\vec{x}_4, \vec{x}_0\right) \end{array}\right] = \left[\begin{array}{c} cov\left(\vec{x}_{1high}, \vec{x}_{0high}\right) \\ cov\left(\vec{x}_{2high}, \vec{x}_{0high}\right) \\ cov\left(\vec{x}_{3low}, \vec{x}_{0high}\right) \\ cov\left(\vec{x}_{4low}, \vec{x}_{0high}\right) \end{array}\right]$$

Analog dazu für die Low-Fidelity Vorhersage, muss der Punkt $\vec{x}_0$ als Low-Fidelity Punkt angesehen werden:

$$\vec{c} = \left[\begin{array}{c|c} & Z_{low}\left(\vec{x}_0\right) \\ \hline Z_{high}\left(\vec{x}_1\right) & cov\left(\vec{x}_1, \vec{x}_0\right) \\ Z_{high}\left(\vec{x}_2\right) & cov\left(\vec{x}_2, \vec{x}_0\right) \\ Z_{low}\left(\vec{x}_3\right) & cov\left(\vec{x}_3, \vec{x}_0\right) \\ Z_{low}\left(\vec{x}_4\right) & cov\left(\vec{x}_4, \vec{x}_0\right) \end{array}\right] = \left[\begin{array}{c} cov\left(\vec{x}_{1high}, \vec{x}_{0low}\right) \\ cov\left(\vec{x}_{2high}, \vec{x}_{0low}\right) \\ cov\left(\vec{x}_{3low}, \vec{x}_{0low}\right) \\ cov\left(\vec{x}_{4low}, \vec{x}_{0low}\right) \end{array}\right]$$

Die genaue Formulierung der Kovarianzfunktion und die Umsetzung wird in den nächsten Kapiteln erläutert.

4.2.1.2 Varianz des Fehlers

Die Vorhersage der Varianz des Fehlers wir damit zu:

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

4.2.1.3 Kovarianzen zwischen zwei Samples

sdfsdfsdf

4.2.1.4 Verschiedene Fidelities vorhersagen

<sdfsdf

4.2.2 Kovarianzmodell

Der hier vorgestellte CO-Kriging Ansatz ist in den Grundzügen aus den Arbeiten von Kennedy und Forrester hervorgegangen [29][32].

4.2.3 Beta Bestimmung

askjfnsöadgf

WAS IST BETA BEWEIß

4.3 Simple und Ordinary Kriging als Spezialfall vom Co-Kriging

Nur kurz erklären, mit Hinblick auf die gemeinsame Machbarkeit im Code

4.4 Gradient Enhanced Kriging als Spezialfall vom Oridnary Kriging

Nur kurz erklären, mit Hinblick auf die gemeinsame Machbarkeit im Code

4.5 Maximum Likelihood für alle Kriging Verfahren

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.

4.5.1 Noch nicht eingeordnet

- Additive Brücke vergleichen

- Kennedy

-Keane

-Braunschweiger [33][28][27]

- 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 [29] 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.

4.6 Zusammenhang zwischen Bedingter Normalverteilung und Kriging

4.7 Regularisierungsterm und Nugget für alle Verfahren

4.8 Vergleiche zu anderen Verfahren

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.

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

4.9 Vorhersagen Beispiele

Beispiele zeigen, wo A,B und C geändert werden und die Faktoren/Summanden sich entsprechend anpassen. (my, sigma und beta)

Vorhersage von LF Funktion zeigen !!!!

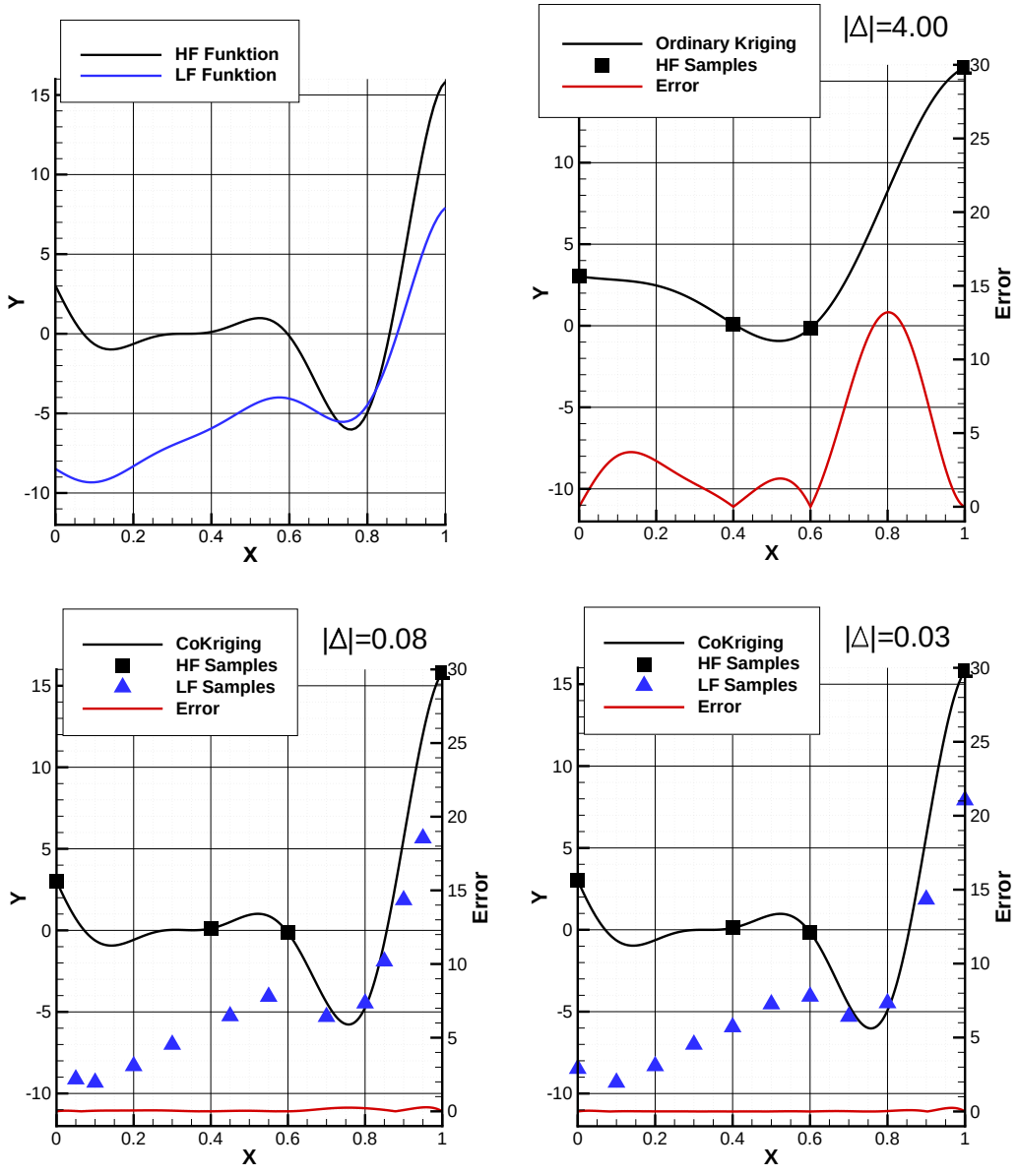


Abbildung 4.1:

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 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 [34] 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. 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:

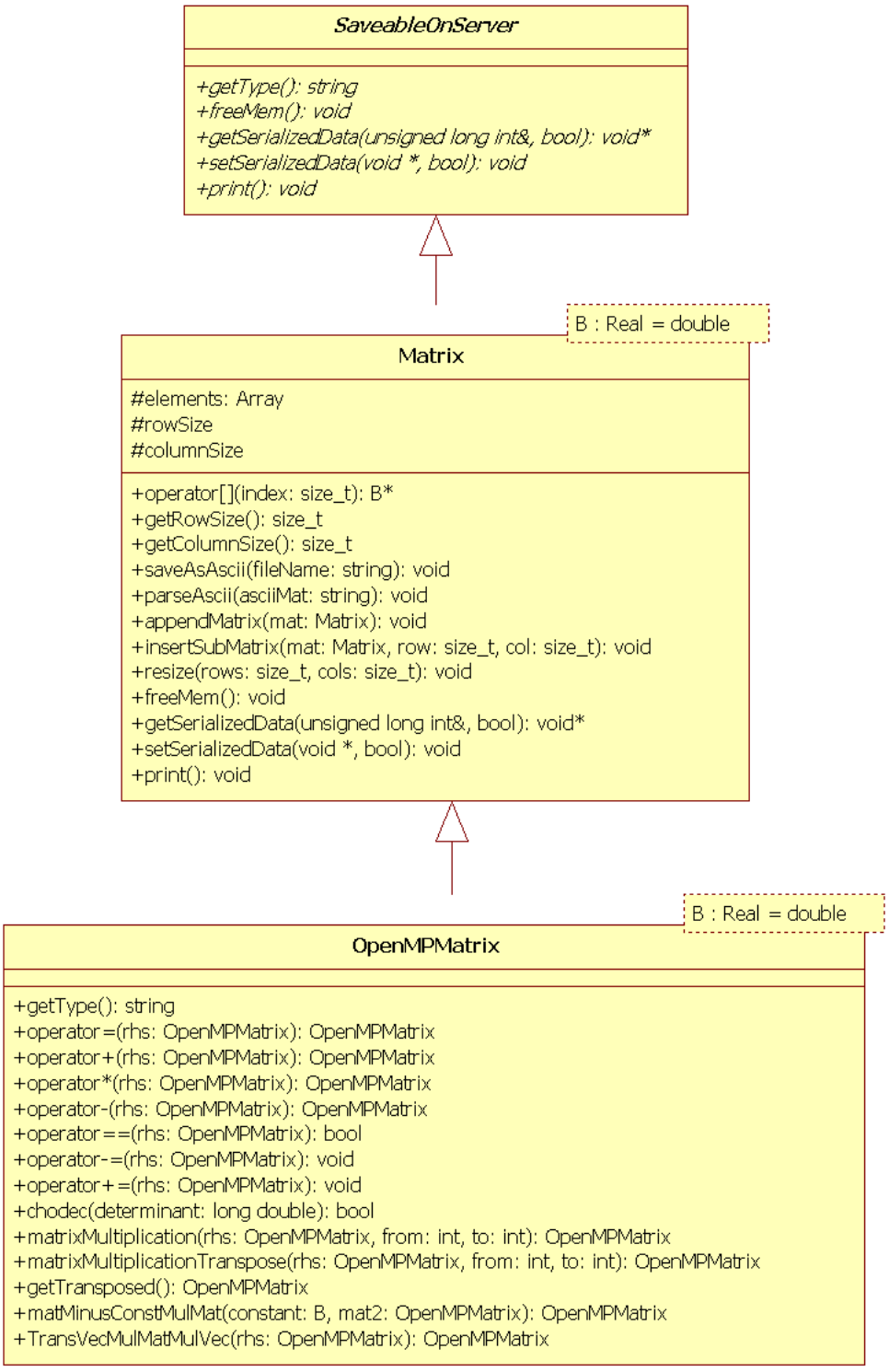


Abbildung 5.1: UML Diagramm der Matrixklasse mit einer Spezialisierung (OpenMPMatrix) und der abstrakten Klasse SaveableOnServer, welche prinzipiell ein Interface darstellt. Interfaces weden in C++ jedoch nicht unterstützt, daher wird eine abstrakte Klasse verwendet.

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 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.

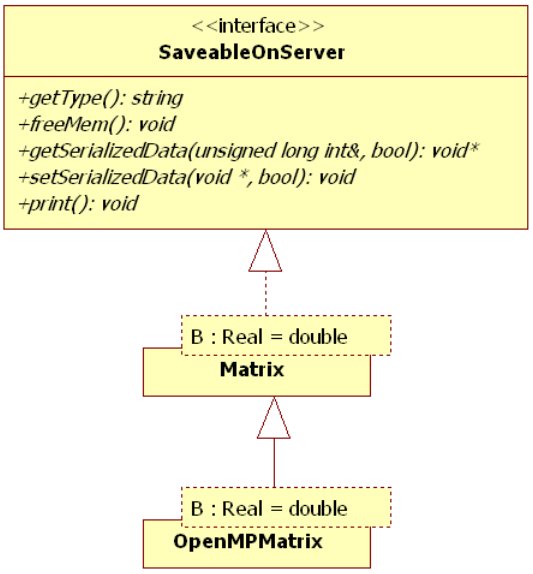


Abbildung 5.2: UML Diagramm der Matrix Klasse, wobei SaveableOnServer in diesem Fall durch ein kontextspezifisches Client Server Interface dargestellt wird.

5.1.2 Korrelationsfunktionen

Um die Korrelationsmatrix 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:

$$c\left(\vec{x}_1, \vec{x}_2\right) = e^{-\frac{1}{2}\sum_{l=1}^{l<k}\left(e^{\theta_l}|x_{1l}-x_{2l}|^2\right)} \tag{5.1}$$

Die Vektoren $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^k$ 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 $\vec{\theta} \in \mathbb{R}^k$ (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 5.3 ist eine beispielhafte Korrelationsfunktion mit zwei verschiedenen θ Werten zu sehen, wobei die Stützstellen in der

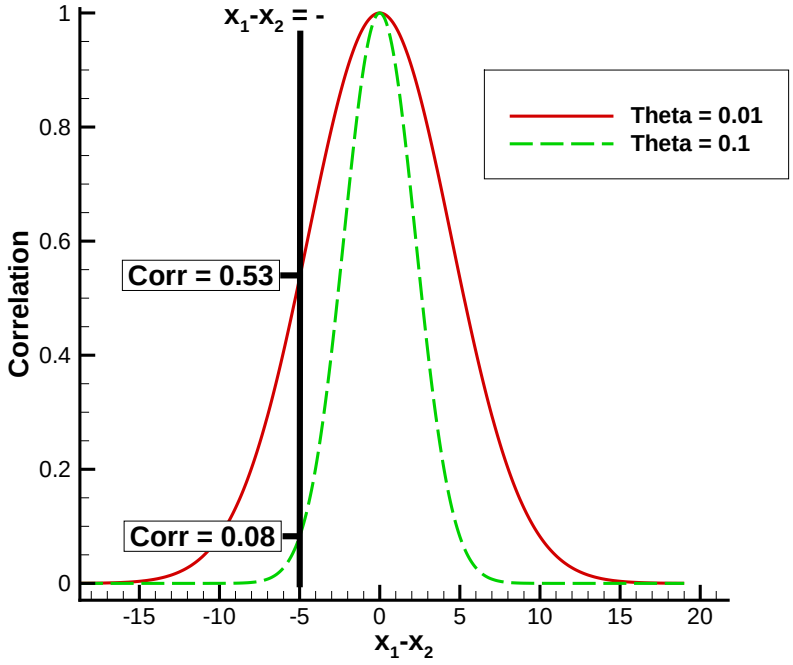


Abbildung 5.3: Beispiel einer Gauss Korrelationsfunktion mit einer freien Variable und zwei unterschiedlichen θ Einstellungen und deren Einfluss auf den Korrelationswert

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 5.4 wird eine Korrelationsfunktion zwischen zwei Stützstellen $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^k$ mit zwei freien Variablen $k = 2$ gezeigt. Für diesen Fall hat der Vektor $\vec{\theta} \in \mathbb{R}^k$ 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.

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));
}
```

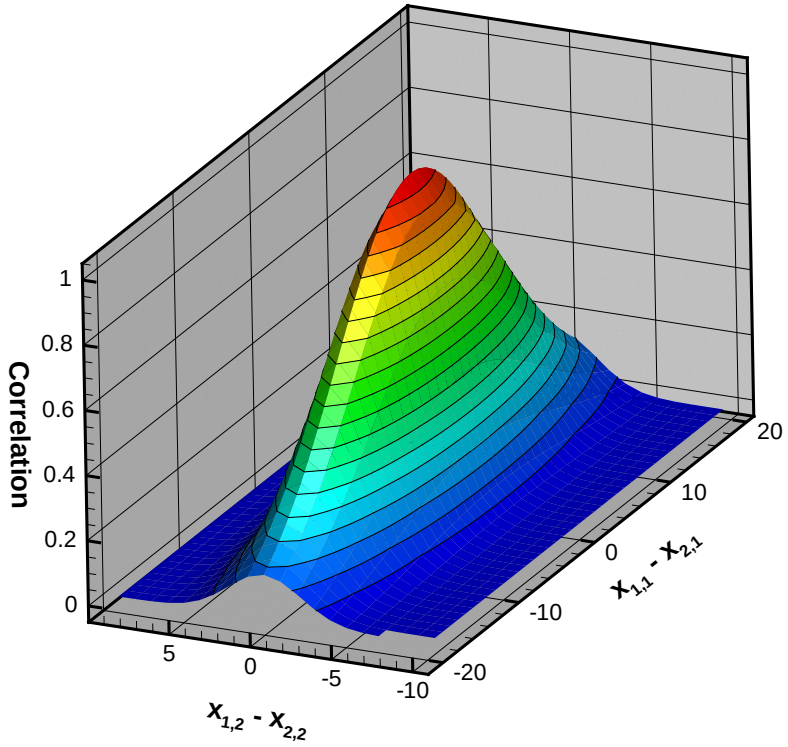


Abbildung 5.4: Beispiel einer Gauss Korrelationsfunktion mit zwei freien Variablen

```
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 [35, 36]. 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);
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

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

Tabelle 5.1: 128Bit Speicherausrichtung

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 Metho-

den berechnet. Die gemessenen Geschwindigkeitsvorteile lagen bei ca. 20 % - 30 %.

5.1.3 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 5.5 ist das UML Diagramm der entsprechenden Klasse dargestellt.

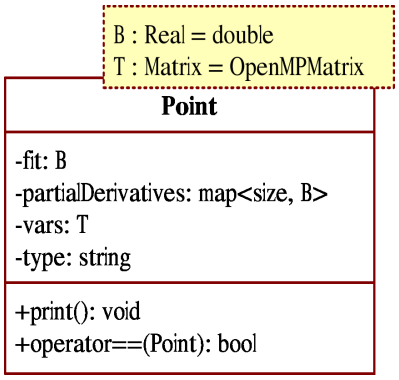


Abbildung 5.5: UML Diagramm der Klasse Point, welche einen Member/Stützstelle beschreibt

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 5.6 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:

$$\mathbf{R} = \left[\begin{array}{c|cc|cc} & Z\left(\vec{x}_1\right) & \frac{\partial Z(\vec{x}_1)}{\partial x_1^1} & Z\left(\vec{x}_2\right) & \frac{\partial Z(\vec{x}_2)}{\partial x_1^1} \\ \hline Z\left(\vec{x}_1\right) & c\left(\vec{x}_1,\vec{x}_1\right) & \frac{\partial c(\vec{x}_1,\vec{x}_1)}{\partial x_1^1} & c\left(\vec{x}_1,\vec{x}_2\right) & \frac{\partial c(\vec{x}_1,\vec{x}_2)}{\partial x_2^1} \\ \frac{\partial Z(\vec{x}_1)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_1)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_1)}{\partial x_1^1 \partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_2)}{\partial x_1^1} & \frac{\partial c(\vec{x}_1,\vec{x}_2)}{\partial x_1^1 \partial x_2^1} \\ Z\left(\vec{x}_2\right) & c\left(\vec{x}_2,\vec{x}_1\right) & \frac{\partial c(\vec{x}_2,\vec{x}_1)}{\partial x_1^1} & c\left(\vec{x}_2,\vec{x}_2\right) & \frac{\partial c(\vec{x}_2,\vec{x}_2)}{\partial x_2^1} \\ \frac{\partial Z(\vec{x}_2)}{\partial x_1^1} & \frac{\partial c(\vec{x}_2,\vec{x}_1)}{\partial x_2^1} & \frac{\partial c(\vec{x}_2,\vec{x}_1)}{\partial x_2^1 \partial x_1^1} & \frac{\partial c(\vec{x}_2,\vec{x}_2)}{\partial x_2^1} & \frac{\partial c(\vec{x}_2,\vec{x}_2)}{\partial x_2^1 \partial x_2^1} \end{array}\right] \tag{5.2}$$

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

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

Abbildung 5.6: UML Diagramm der abstrakten Klasse CorrelationFunction mit zwei Subklassen, diese stellen spezifische Korrelationsfunktionen dar

Die Methode CreateCorrelationMatrix

Das UML Diagramm in Abbildung 5.6 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 5.1.2 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 5.2), 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 5.5) 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 5.5) 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 5.6), 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 5.6) 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 5.2.3 wird der Diagonalaufschlag genauer erläutert.

Innerhalb des If- Else Blocks wird die Methode getAllCorrelation der entsprechenden Subklasse von CorrelationFunction (siehe Abbildung 5.6) 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 5.6) definiert ist, soll alle Korrelationswerte zwischen zwei Stützstellen berechnen. Diese werden in die Korrelationsmatrix eingetragen und von der Funktion createCorrelationMatrix (siehe Kapitel 5.1.3) 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 5.2.3 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 5.1.2 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 5.2, 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 5.1.2.

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 5.6 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.

5.1.4 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

5.1.5 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.

5.1.6 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 5.7 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:

$$\vec{R}_f = \mathbf{R}^{-1}\overrightarrow{F} \tag{5.3}$$

$$f_{rf} = \vec{F}^T\vec{R}_f \tag{5.4}$$

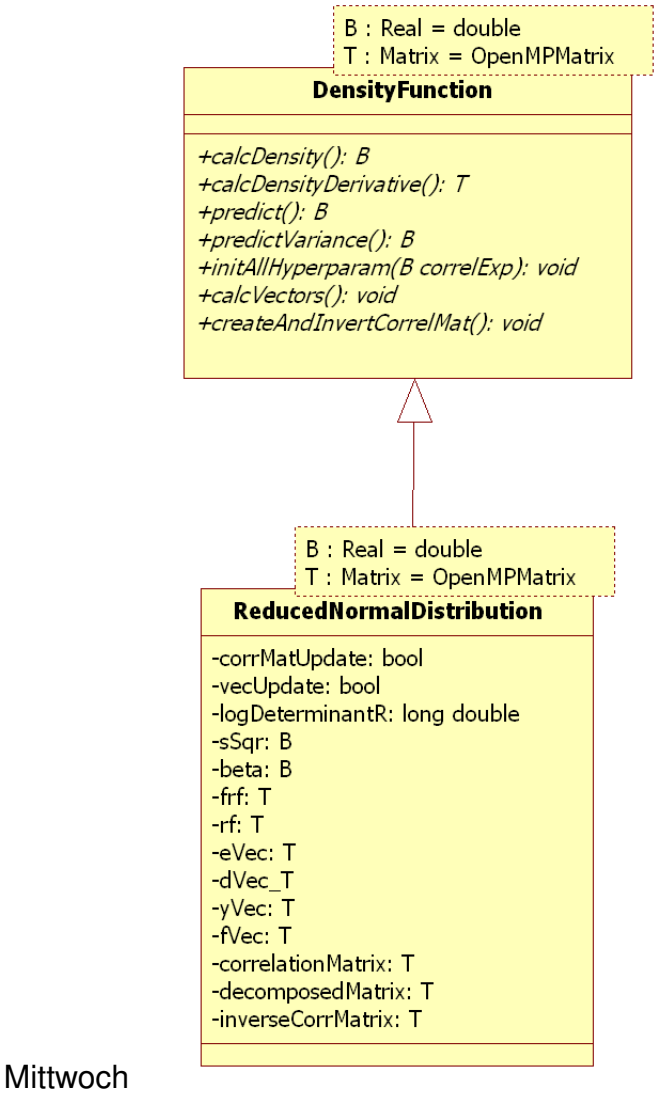
$$\overrightarrow{e} = (\vec{y}_s - \beta * \overrightarrow{F}) \tag{5.5}$$

$$\overrightarrow{d} = \mathbf{R}^{-1}\overrightarrow{e} \tag{5.6}$$

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 5.1.1 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 5.7: 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 5.3-5.6) 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 5.7) 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 5.2.3 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 5.2.3) 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:

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

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

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 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 Ursprungs-

matrix 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 Einheitsmatrix 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:

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

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

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 Untergrenze für den Diagonalaufschlag:

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

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

5.2.3 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 $\mathrm{var}(X) = E[X^2] - E[X]^2$. Die im Modell verwendeten Daten werden grundsätzlich auf einen Erwartungswert von Null und eine Standardabweichung von Eins normiert.

$$E[X] = 0$$

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

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

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

Also

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

und analog dazu:

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

Daraus folgt:

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

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

$$\log\left(-\frac{\log\left(c_{erw}\right)}{k}\right) = \theta \tag{5.7}$$

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 5.2.3 vorgestellte Verfahren, für den Erwartungswert der Korrelationen sollte man hier einen hohen Wert wählen, bspw. 0.99

$$\log\left(-\frac{\log\left(0.99\right)}{k}\right) = \theta_{min} \tag{5.8}$$

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:

$$\log\left(-\log\left(0.01\right)\right) = \theta_{max} \tag{5.9}$$

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 5.1.5). 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 ebenfalls günstiger.

5.2.4 Minimierungsverfahren

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

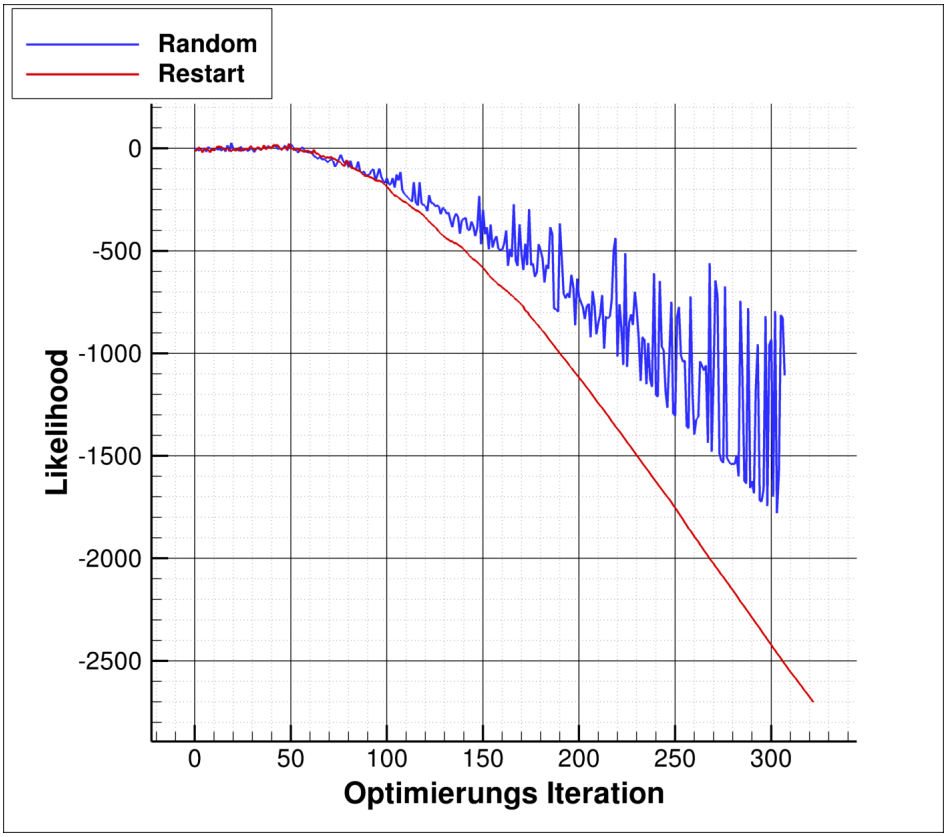


Abbildung 5.8: Vergleich verschiedener Initialisierungsverfahren und deren Auswirkung auf eine Testoptimierung

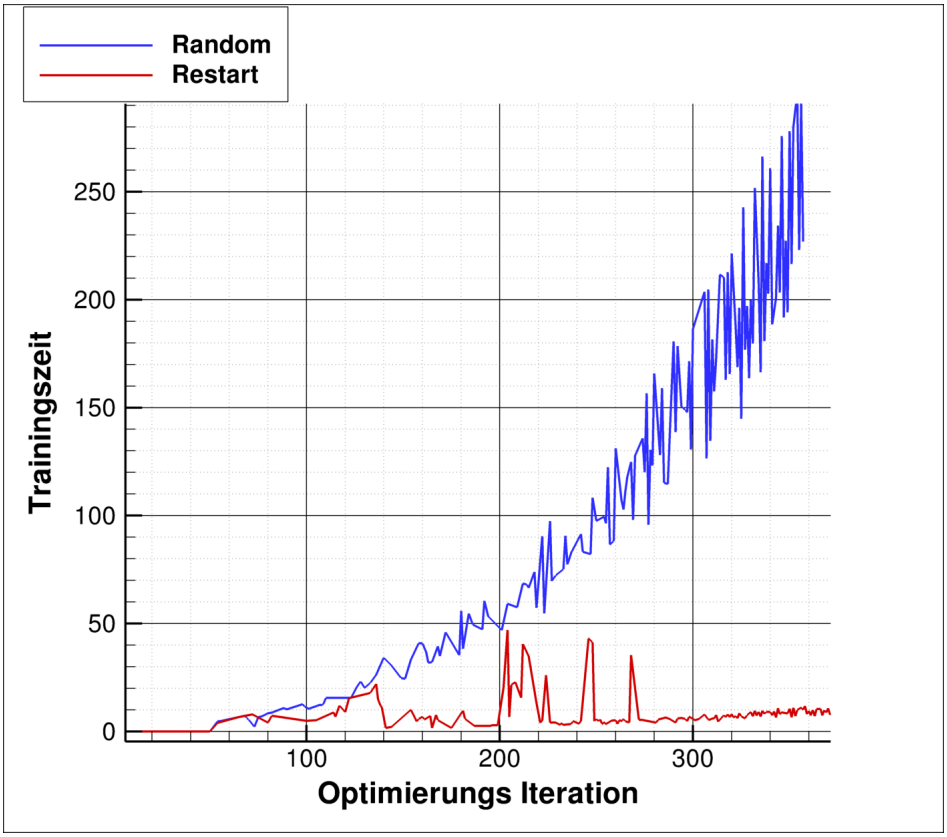


Abbildung 5.9: Vergleich verschiedener Initialisierungsverfahren und deren Auswirkung auf eine Testoptimierung

Minimierungsverfahren angelehnt an Resilient Backpropagation

Das erste hier verwendete Minimierungsverfahren ist angelehnt an ein Trainingsverfahren für Neuronale Netzwerke, genannt RPROP (Resilient Backpropagation) [37, 38] 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 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

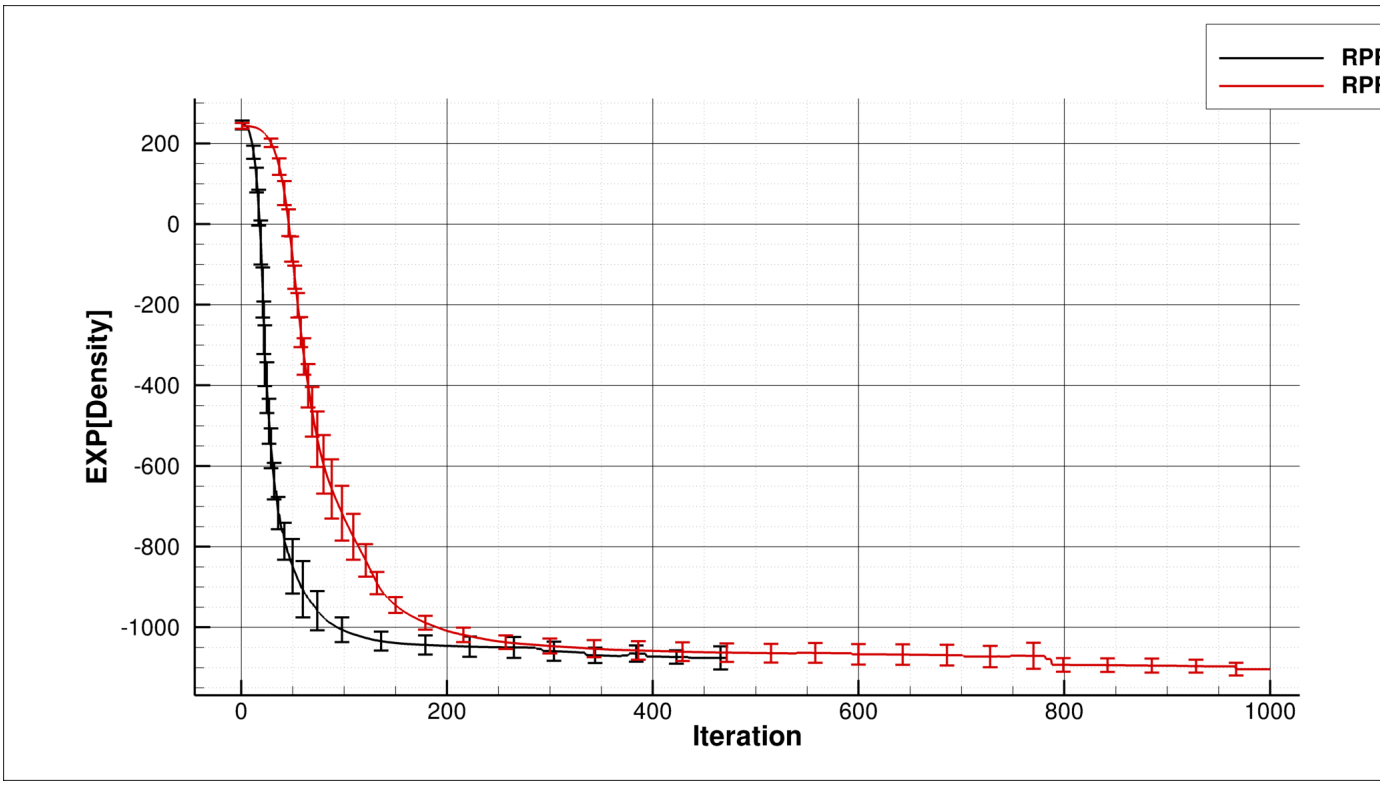


Abbildung 5.10:

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 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 [**?**, 6, 7] gefunden werden.

5.2.5 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 5.11 zeigt die Klasse Trainer, welche das Training steuern und verwalten soll.

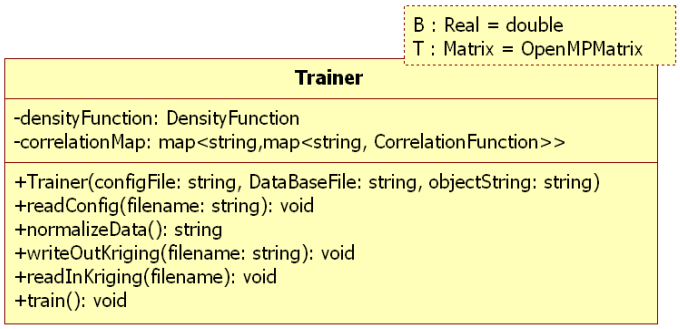


Abbildung 5.11: UML Diagramm der Trainer Klasse, welche das Training des Kriging Modells steuert

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 5.1.2). 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 5.2.3.

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 5.12 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 5.1.5.

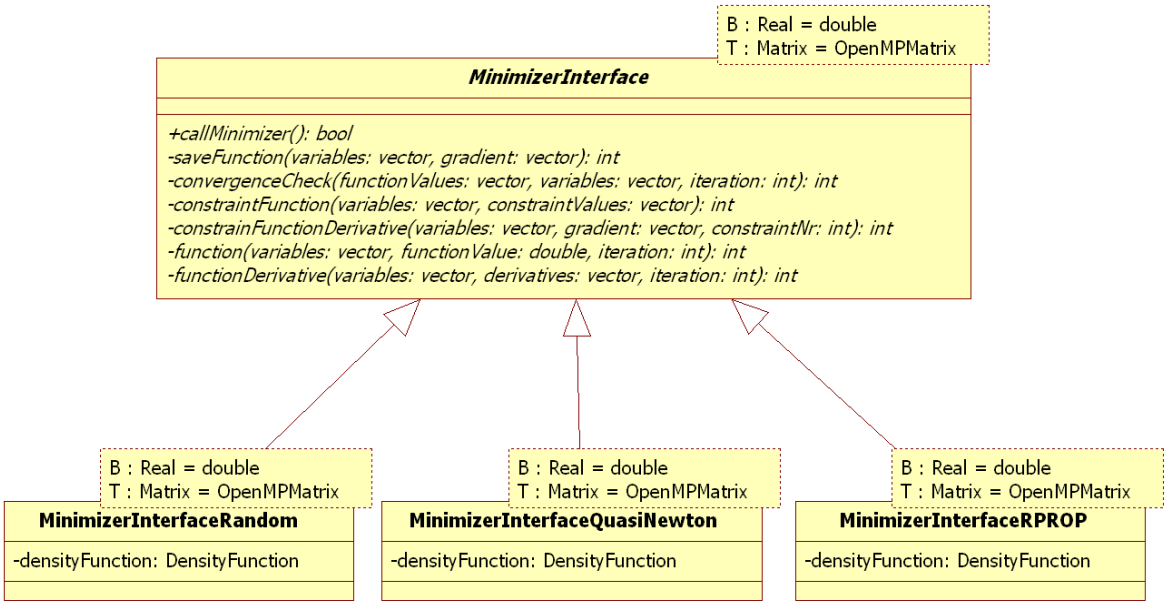


Abbildung 5.12: UML Diagramm der Klassenstruktur der Minimierungsalgorithmen

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 wür-

den. 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 [39] 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.

5.2.6 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 Membern unterschiedliche Werte bekommen.

Daraus ergibt sich folgende notwendige Bedingung:

$$cov\left(\vec{x}_{1alt}, \vec{x}_{2alt}\right) = cov\left(\vec{x}_{1neu}, \vec{x}_{2neu}\right) \tag{5.10}$$

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 5.10 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 **??**.

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 Algorithmische Effizienz steigern

5.3.1 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.

....

....

....

5.3.2 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 5.4.1). 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.

5.3.3 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 [40, 41, **?**]

$$Spur\left(R\right) = E\left[\vec{d}^T R\vec{d}\right]$$

$$Spur\left(R\right) \approx \frac{1}{N}\sum \vec{d}^T R\vec{d}$$

$$\vec{d} = \begin{bmatrix} N\left(0,1\right) \\ \vdots \\ N\left(0,1\right) \end{bmatrix}$$

Die Approximation benötigt leider einen sehr großen Zufallsvektor $\vec{d}$ um eine ausrei-

chende 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.

5.3.4 Vollständiger Verzicht auf die Inverse durch Rückwärtsdifferentiation der Cholesky Zerlegung

Wie in Kapitel 5.3.3 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 $\mathcal{O}(n^2)$ 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 5.4.1. 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 5.3.3 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 [42] finden. Dieser bedient sich der algorithmischen Differentiation im Rückwärtsmodus, der interessierte Leser sei auf [43, 44] 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 [45] 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 erzeugt werden.

Eine andere Möglichkeit bietet die alleinige Rückwärtsdifferentiation der Determinante der Kovarianzmatrix. Dieser Ansatz bietet in manchen Fällen Vorteile, wie sich im Weiteren herausstellen wird. Es wird allerdings eine rückwärtsdifferenzierte Version des Cholesky Algorithmus benötigt. Es werden mehrere effiziente Implementierungen aufgezeigt. 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:

$$\frac{\partial \left(f_{lndet} \left(f_{chol} \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l} = D_{flndet} D_{chol} D_{cov}$$

$$\frac{\partial \left(f_{lndet} \left(f_{chol} \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l} = \sum_i \sum_{j<=i} \sum_k \sum_{m<=k} \frac{\partial f_{lndet}}{\partial \mathbf{L_{i,j}}} \frac{\partial \mathbf{L_{i,j}}}{\partial \mathbf{C_{k,m}}} \frac{\partial \mathbf{C_{k,m}}}{\partial h_l} \tag{5.11}$$

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 5.11 stark vereinfachen, da nur noch die Einträge $\frac{\partial f_{lndet}}{\partial \mathbf{L_{i,j}}} \neq 0 \,|i = j$ sind:

$$\frac{\partial \left(f_{lndet}\left(f_{chol}\left(\mathbf{Cov}\left(h_l\right)\right)\right)\right)}{\partial h_l} = \sum_i \sum_k \sum_{m<=k} \frac{\partial f_{lndet}}{\partial \mathbf{L_{i,i}}} \frac{\partial \mathbf{L_{i,i}}}{\partial \mathbf{C_{k,m}}} \frac{\partial \mathbf{C_{k,m}}}{\partial h_l} \tag{5.12}$$

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 [46]. Das ist immer noch deutlich schneller als eine Vor- und Rückwärtssubstitution. Der in [46] beschriebene Algorithmus ist allerdings nur schwer parallelisierbar und auch für SIMD Architekturen nur schlecht geeignet. Im Folgenden soll der Originalalgorithmus gezeigt werden und eine parallelisierte und SSE beschleunigte Variante. In der Arbeit von [47] wurde noch eine modernere Variante präsentiert, welche es ermöglicht Standard Level 3 BLAS Routinen zu verwenden. Hierdurch wird es möglich auch GPUs zur Berechnung zu verwenden.

Algorithmus nach Smith [46] sdfykljg

Symbolische Differenzierung in [48]

Alternative Herleitung der symbolischen Differenzierung

$$\frac{\partial \left(ln \left(det \left(\mathbf{Cov} \left(h_l \right) \right) \right) \right)}{\partial h_l} = \text{Spur} \left(\mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l} \right)$$

$$\phi_{ij}^{L} \left(A \right) = \begin{cases} A_{ij} & i > j \\ 0.5 A_{ij} & i = j \\ 0 & i < j \end{cases}$$

$$\phi_{ij}^{U} \left(A \right) = \begin{cases} 0 & i > j \\ 0.5 A_{ij} & i = j \\ A_{ij} & i < j \end{cases}$$

$$\text{Spur} \left(\mathbf{Cov}^{-1} \frac{\partial \mathbf{Cov}}{\partial h_l} \right) = \text{Spur} \left(\left(\phi^{U} \left(\mathbf{Cov}^{-1} \right) + \phi^{L} \left(\mathbf{Cov}^{-1} \right) \right) \frac{\partial \mathbf{Cov}}{\partial h_l} \right)$$

$$= \text{Spur}\left(\phi^U\left(\mathbf{Cov}^{-1}\right)\frac{\partial \mathbf{Cov}}{\partial h_l} + \phi^L\left(\mathbf{Cov}^{-1}\right)\frac{\partial \mathbf{Cov}}{\partial h_l}\right)$$

$$= \text{Spur}\left(\phi^U\left(\mathbf{Cov}^{-1}\right)\frac{\partial \mathbf{Cov}}{\partial h_l}\right) + \text{Spur}\left(\phi^L\left(\mathbf{Cov}^{-1}\right)\frac{\partial \mathbf{Cov}}{\partial h_l}\right)$$

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.

5.4 Verwendung von GPGPU

General Purpose Computation on Graphics Processing Unit blablabla#

5.4.1 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 [45] wird dieser Ansatz nochmals verändert und mithilfe der linearen Algebra Ansätze von [49] 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 [45] 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 [47] 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

sdfgsdf

sdf

Name	Max. GFlops (double FMA)	Straßenpreis (
Tesla P100 NVLink	5300	Unbeka
Tesla P100 PCIe	4700	Unbeka
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	Unbekannt	16GB	490	245W
7250	3046	Unbekannt	16GB	490	215W
7120	1210	4000€	16GB (DDR4)	352	300W

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

$$\overrightarrow{v} * \mathbf{R} * \overrightarrow{v} > 0 \; f\ddot{u}r\, alle\, Vektoren\, \overrightarrow{v}$$

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:

$$det\,(\mathbf{R}) = \prod_{i=1}^{n} L_{i,i}^2$$

Der verwendete Algorithmus ist in [**?**, 7] 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.

5.5 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	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

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

5.6 Analysesoftware von Krigingmodellen während der Laufzeit

Khon.py beschreiben

5.7 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:[50], CERN Paper mit ZeroMQ Empfehlung:[51]

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

Ordinary Kriging

GEK

COkriging

Trainingszeit, Vorhersagegenauigkeit

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

Reale Anwendung

- Mukoko Verdichter (ca. 200 Parameter)

- HF mit vollständiger Lastervertilgungsfreigabe unrealistisch daher MF

- Benchmark sinnlos, da viele Einstellungen während der Optimierungs geänder werden müssen und daher ein Vergleich sehr schwierig ist

- Axiale Lastverteilung zeigen, Änderungen an den Schaufeln und deren Aerodynamik
- GPU Nutzung Vorteile zeigen

8 Fazit und Ausblick

asdas

A Anhang

A.1 Varianz der Fehlerfunktion

$$\text{var}\left[F\left(\vec{x}_0\right)\right] = \text{var}\left[Z\left(\vec{x}_0\right) - Z^*\left(\vec{x}_0\right)\right] = \text{var}\left[Z\left(\vec{x}_0\right) - \sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right)\right]$$

$$= E\left[\left(\left(Z\left(\vec{x}_0\right) - \sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right)\right) - E\left[Z\left(\vec{x}_0\right) - \sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right)\right]\right)^2\right]$$

$$= E\left[\left(Z\left(\vec{x}_0\right) - \sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right) - E\left[Z\left(\vec{x}_0\right)\right] + E\left[\sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right)\right]\right)^2\right]$$

$$= E\left[\left(\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right] - \left[\sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right) - \sum_{i=1}^{n_{all}} E\left[w_i Z\left(\vec{x}_i\right)\right]\right]\right)^2\right]$$

$$= E\left[\left(\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right] - \left[\sum_{i=1}^{n_{all}} \left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]\right)^2\right]$$

$$= E\left[\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]^2 - 2\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]\left[\sum_{i=1}^{n_{all}} \left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right] + \left[\sum_{i=1}^{n_{all}} \left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]^2\right]$$

$$= E\left[\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]^2\right] - E\left[2\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]\left[\sum_{i=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]\right] + E\left[\left(\sum_{i=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right)^2\right]$$

$$= \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - E\left[2\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]\left[\sum_{i=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]\right] + E\left[\left(\sum_{i=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right)^2\right]$$

$$= \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - E\left[2\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]\left[\sum_{i=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]\right] + E\left[\sum_{i=1}^{n_{all}}\sum_{j=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right) * \left(w_j Z\left(\vec{x}_j\right) - E\left[w_j Z\left(\vec{x}_j\right)\right]\right)\right]$$

$$= \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - E\left[2\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right]\left[\sum_{i=1}^{n_{all}}\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]\right] + \sum_{i=1}^{n_{all}}\sum_{j=1}^{n_{all}} E\left[\left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right) * \left(w_j Z\left(\vec{x}_j\right) - E\left[w_j Z\left(\vec{x}_j\right)\right]\right)\right]$$

$$= \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - E\left[2\left[\sum_{i=1}^{n_{all}}\left[Z\left(\vec{x}_0\right) - E\left[Z\left(\vec{x}_0\right)\right]\right] * \left(w_i Z\left(\vec{x}_i\right) - E\left[w_i Z\left(\vec{x}_i\right)\right]\right)\right]\right] + \sum_{i=1}^{n_{all}}\sum_{j=1}^{n_{all}} cov\left[w_i Z\left(\vec{x}_i\right), w_j Z\left(\vec{x}_j\right)\right]$$

$$\mathrm{var}\left[F\left(\vec{x}_0\right)\right] = \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - 2\sum_{i=1}^{n_{all}} \mathrm{cov}\left(Z\left(\vec{x}_0\right), w_i Z\left(\vec{x}_i\right)\right) + \sum_{i=1}^{n_{all}}\sum_{j=1}^{n_{all}} w_i w_j \mathrm{cov}\left(Z\left(\vec{x}_i\right), Z\left(\vec{x}_j\right)\right)$$

In Matrix-Schreibweise ergibt sich die folgende Formulierung, wobei $\vec{w} \in \mathbb{R}^{\mathrm{n_{all}}}$ den Ge-

wichtsvektor, $\mathbf{Cov} \in \mathbb{R}^{\mathrm{n_{all}} \times \mathrm{n_{all}}}$ die Kovarianzmatrix und $\overrightarrow{\mathrm{cov}} \in \mathbb{R}^{\mathrm{n_{all}}}$ den Kovarianzvektor darstellt:

$$\mathrm{var}\left[F\left(\vec{x}_0\right)\right] = \mathrm{var}\left[Z\left(\vec{x}_0\right)\right] - 2\overrightarrow{\mathrm{cov}}\left(Z\left(\vec{x}_0\right), Z\left(\vec{x}\right)\right)^T * \vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \tag{A.1}$$

A.2 Varianz der Fehlerfunktion

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

$$= E\left[\left(\left(Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right) - E\left[Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right]\right)^2\right]$$

$$= E\left[\left(Z_k\left(\vec{x}\right) - \sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i - E\left[Z_k\left(\vec{x}\right)\right] + E\left[\sum_{i=1}^{s} \vec{Z}_i^T \vec{w}_i\right]\right)^2\right]$$

Im folgenden Schritt wird das Skalarprodukt als Summe formuliert:

$$= E\left[\left(\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right] - \left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} Z_i\left(\vec{x}_j\right) w_{i,j} - \sum_{i=1}^{s}\sum_{j=1}^{n_i} E\left[Z_i\left(\vec{x}_j\right)\right] w_{i,j}\right]\right)^2\right]$$

$$= E\left[\left(\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right] - \left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i\left(\vec{x}_j\right) w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)\right] w_{i,j}\right)\right]\right)^2\right]$$

$$= E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]^2 - 2\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i\left(\vec{x}_j\right) w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)\right] w_{i,j}\right)\right] + \left[\sum_{i=1}^{s}\sum_{j=1}^{n_i} \left(Z_i\left(\vec{x}_j\right) w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)\right] w_{i,j}\right)\right]^2\right]$$

$$= E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]^2\right] - 2E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]\right]$$
$$+ E\left[\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]^2\right]$$

$$= \mathrm{var}\left[Z_k\left(\vec{x}\right)\right] - 2E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]\right]$$
$$+ E\left[\left[\sum_{i=1}^{s}\left(\vec{Z}_i^T\vec{w}_i - E\left[Z_i\right]1^{\vec{T}}\vec{w}_i\right)\right]^2\right]$$

$$= \mathrm{var}\left[Z_k\left(\vec{x}\right)\right] - 2E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]\right]$$
$$+ E\left[\left[\sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\left(Z_k\left(\vec{x}_l\right)w_{k,l} - E\left[Z_k\left(\vec{x}_l\right)w_{k,l}\right]\right)\right]\right]$$

$$= \mathrm{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i}E\left[\left[Z_k\left(\vec{x}\right) - E\left[Z_k\left(\vec{x}\right)\right]\right]\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\right]$$
$$+ \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}E\left[\left(Z_i\left(\vec{x}_j\right)w_{i,j} - E\left[Z_i\left(\vec{x}_j\right)w_{i,j}\right]\right)\left(Z_k\left(\vec{x}_l\right)w_{k,l} - E\left[Z_k\left(\vec{x}_l\right)w_{k,l}\right]\right)\right]$$

$$= \mathrm{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i}\mathrm{cov}\left(Z_k\left(\vec{x}\right), Z_i\left(\vec{x}_j\right)w_{i,j}\right)$$
$$+ \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i}\mathrm{cov}\left(Z_i\left(\vec{x}_j\right)w_{i,j}, Z_k\left(\vec{x}_l\right)w_{k,l}\right)$$

$$= \text{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i} w_{i,j}\text{cov}\left(Z_k\left(\vec{x}\right), Z_i\left(\vec{x}_j\right)\right) + \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i} w_{i,j}w_{k,l}\text{cov}\left(Z_i\left(\vec{x}_j\right), Z_k\left(\vec{x}_l\right)\right)$$

$$\text{var}\left[F\left(\vec{x}\right)\right] = \text{var}\left[Z_k\left(\vec{x}\right)\right] - 2\sum_{i=1}^{s}\sum_{j=1}^{n_i} w_{i,j}\text{cov}\left(Z_k\left(\vec{x}\right), Z_i\left(\vec{x}_j\right)\right) + \sum_{i=1}^{s}\sum_{j=1}^{n_i}\sum_{k=1}^{s}\sum_{l=1}^{n_i} w_{i,j}w_{k,l}\text{cov}\left(Z_i\left(\vec{x}_j\right), Z_k$$

In Matrix-Schreibweise ergibt sich die folgende Formulierung, wobei $\vec{w} \in \mathbb{R}^{\text{n}_{\text{all}}}$ den Gewichtsvektor, $\mathbf{Cov} \in \mathbb{R}^{\text{n}_{\text{all}} \times \text{n}_{\text{all}}}$ die Kovarianzmatrix und $\overrightarrow{\text{cov}} \in \mathbb{R}^{\text{n}_{\text{all}}}$ den Kovarianzvektor darstellt:

$$\text{var}\left[F\left(\vec{x}_0\right)\right] = \text{var}\left[Z\left(\vec{x}_0\right)\right] - 2\overrightarrow{\text{cov}}\left(Z\left(\vec{x}_0\right), Z\left(\vec{x}\right)\right)^T * \vec{w} + \vec{w}^T * \mathbf{Cov} * \vec{w} \tag{A.2}$$

A.3 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:

$$\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} = \left(\left(a^2\sigma_{low}^2 C_{11A} + a^2\sigma_{err}^2 C_{11B}\right) - a\sigma_{low}^2 C_{12}\left(\sigma_{low}^2 C_{22}\right)^{-1} a\sigma_{low}^2 C_{12}\right)^{-1}$$

$$= \left(a^2\left(\sigma_{low}^2 C_{11A} + \sigma_{err}^2 C_{11B}\right) - a^2 C_{12}\left(C_{22}\right)^{-1}\sigma_{low}^2 C_{12}\right)^{-1}$$

$$= \left(a^2\left(\sigma_{low}^2 C_{11A} + \sigma_{err}^2 C_{11B} - \sigma_{low}^2 C_{12}\left(C_{22}\right)^{-1} C_{12}\right)\right)^{-1}$$

$$\mathbf{Cov^{-1}}\vec{G} = \begin{bmatrix} \left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} & -\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1} \\ -C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} & C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1} \end{bmatrix} \begin{bmatrix} \vec{1} & \vec{0} \\ \vec{0} & \vec{1} \end{bmatrix}$$

$$\mathbf{Cov^{-1}}\vec{G} = \begin{bmatrix} \left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & -\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\vec{1} \\ -C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & \left(C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\right)\vec{1} \end{bmatrix}$$

$$\vec{G}^T\mathbf{Cov^{-1}}\vec{G} = \begin{bmatrix} \vec{1}^T\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & -\vec{1}^T\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-1}\vec{1} \\ -\vec{1}^T C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1}\vec{1} & \vec{1}^T\left(C_{22}^{-1} + C_{22}^{-1}C_{12}\left(C_{11} - C_{12}C_{22}^{-1}C_{12}\right)^{-1} C_{12}C_{22}^{-} \end{bmatrix}$$

Vorausgesetzt man kennt die Low Varianz bereits:

R-Matrix Bildungsvorschrift, mit 2 Ableitungen pro Punkt

$$R = \left[\begin{array}{c|ccccccccc}
 & x1 & x2 & x3 & x1/p1 & x1/p2 & x2/p1 & x2/p2 & x2/p3 & x3/p1 \\ \hline
x1 & f\left(x_1,x_1\right) & f\left(x_1,x_2\right) & f\left(x_1,x_3\right) & \frac{\partial f(x_1,x_1)}{\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_2^3} & \frac{\partial f(x_1,x_3}{\partial x_3^1} \\
x2 & f\left(x_2,x_1\right) & f\left(x_2,x_2\right) & f\left(x_2,x_3\right) & \frac{\partial f(x_2,x_1)}{\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^3} & \frac{\partial f(x_2,x_3}{\partial x_3^1} \\
x3 & f\left(x_3,x_1\right) & f\left(x_3,x_2\right) & f\left(x_3,x_3\right) & \frac{\partial f(x_3,x_1)}{\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_2^1} & \frac{\partial f(x_3,x_2)}{\partial x_2^2} & \frac{\partial f(x_3,x_2)}{\partial x_2^3} & \frac{\partial f(x_3,x_3}{\partial x_3^1} \\
x1/p1 & \frac{\partial f(x_1,x_1)}{\partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1} & \frac{\partial f(x_1,x_3)}{\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^1\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^1\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^1\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^1\partial x_2^3} & \frac{\partial f(x_1,x_3}{\partial x_1^1\partial x_3^1} \\
x1/p2 & \frac{\partial f(x_1,x_1)}{\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2} & \frac{\partial f(x_1,x_3)}{\partial x_1^2} & \frac{\partial f(x_1,x_1)}{\partial x_1^2\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^2\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^2\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2\partial x_2^3} & \frac{\partial f(x_1,x_3}{\partial x_1^2\partial x_3^1} \\
x2/p1 & \frac{\partial f(x_2,x_1)}{\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^1} & \frac{\partial f(x_2,x_3)}{\partial x_2^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^1\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1\partial x_2^3} & \frac{\partial f(x_2,x_3}{\partial x_2^1\partial x_3^1} \\
x2/p2 & \frac{\partial f(x_2,x_1)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_3)}{\partial x_2^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^2\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^2\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^2\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2\partial x_2^3} & \frac{\partial f(x_2,x_3}{\partial x_2^2\partial x_3^1} \\
x2/p3 & \frac{\partial f(x_2,x_1)}{\partial x_2^3} & \frac{\partial f(x_2,x_2)}{\partial x_2^3} & \frac{\partial f(x_2,x_3)}{\partial x_2^3} & \frac{\partial f(x_2,x_1)}{\partial x_2^3\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^3\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^3\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^3\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^3\partial x_2^3} & \frac{\partial f(x_2,x_3}{\partial x_2^3\partial x_3^1} \\
x3/p1 & \frac{\partial f(x_3,x_1)}{\partial x_3^1} & \frac{\partial f(x_3,x_2)}{\partial x_3^1} & \frac{\partial f(x_3,x_3)}{\partial x_3^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^1\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^1\partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^1\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_3^1\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_3^1\partial x_2^3} & \frac{\partial f(x_3,x_3}{\partial x_3^1\partial x_3^1} \\
x3/p2 & \frac{\partial f(x_3,x_1)}{\partial x_3^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^2} & \frac{\partial f(x_3,x_3)}{\partial x_3^2} & \frac{\partial f(x_3,x_1)}{\partial x_3^2\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^2\partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^2\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_3^2\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_3^2\partial x_2^3} & \frac{\partial f(x_3,x_3}{\partial x_3^2\partial x_3^1}
\end{array}\right]$$

Durch Zeilen/Spaltentausch, kommt die für den Rechner effizientere Matrix:

$$R = \left[\begin{array}{c|ccccccccc}
 & x1 & x1/p1 & x1/p2 & x2 & x2/p1 & x2/p2 & x2/p3 & x3 & x3/p1 \\ \hline
x1 & f\left(x_1,x_1\right) & \frac{\partial f(x_1,x_1)}{\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^2} & f\left(x_1,x_2\right) & \frac{\partial f(x_1,x_2)}{\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_2^3} & f\left(x_1,x_3\right) & \frac{\partial f(x_1,x_3}{\partial x_3^1} \\
x1/p1 & \frac{\partial f(x_1,x_1)}{\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^1\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^1\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^1\partial x_2^3} & \frac{\partial f(x_1,x_3)}{\partial x_1^1} & \frac{\partial f(x_1,x_3}{\partial x_1^1\partial x_3^1} \\
x1/p2 & \frac{\partial f(x_1,x_1)}{\partial x_1^2} & \frac{\partial f(x_1,x_1)}{\partial x_1^2\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^2\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^2\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2\partial x_2^3} & \frac{\partial f(x_1,x_3)}{\partial x_1^2} & \frac{\partial f(x_1,x_3}{\partial x_1^2\partial x_3^1} \\
x2 & f\left(x_2,x_1\right) & \frac{\partial f(x_2,x_1)}{\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_1^2} & f\left(x_2,x_2\right) & \frac{\partial f(x_2,x_2)}{\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^3} & f\left(x_2,x_3\right) & \frac{\partial f(x_2,x_3}{\partial x_3^1} \\
x2/p1 & \frac{\partial f(x_2,x_1)}{\partial x_2^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^1\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^1\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1\partial x_2^3} & \frac{\partial f(x_2,x_3)}{\partial x_2^1} & \frac{\partial f(x_2,x_3}{\partial x_2^1\partial x_3^1} \\
x2/p2 & \frac{\partial f(x_2,x_1)}{\partial x_2^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^2\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^2\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^2\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2\partial x_2^3} & \frac{\partial f(x_2,x_3)}{\partial x_2^2} & \frac{\partial f(x_2,x_3}{\partial x_2^2\partial x_3^1} \\
x2/p3 & \frac{\partial f(x_2,x_1)}{\partial x_2^3} & \frac{\partial f(x_2,x_1)}{\partial x_2^3\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^3\partial x_1^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^3\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^3\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^3\partial x_2^3} & \frac{\partial f(x_2,x_3)}{\partial x_2^3} & \frac{\partial f(x_2,x_3}{\partial x_2^3\partial x_3^1} \\
x3 & f\left(x_3,x_1\right) & \frac{\partial f(x_3,x_1)}{\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_1^2} & f\left(x_3,x_2\right) & \frac{\partial f(x_3,x_2)}{\partial x_2^1} & \frac{\partial f(x_3,x_2)}{\partial x_2^2} & \frac{\partial f(x_3,x_2)}{\partial x_2^3} & f\left(x_3,x_3\right) & \frac{\partial f(x_3,x_3}{\partial x_3^1} \\
x3/p1 & \frac{\partial f(x_3,x_1)}{\partial x_3^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^1\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^1\partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^1} & \frac{\partial f(x_3,x_2)}{\partial x_3^1\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_3^1\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_3^1\partial x_2^3} & \frac{\partial f(x_3,x_3)}{\partial x_3^1} & \frac{\partial f(x_3,x_3}{\partial x_3^1\partial x_3^1} \\
x3/p2 & \frac{\partial f(x_3,x_1)}{\partial x_3^2} & \frac{\partial f(x_3,x_1)}{\partial x_3^2\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^2\partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^2\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_3^2\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_3^2\partial x_2^3} & \frac{\partial f(x_3,x_3)}{\partial x_3^2} & \frac{\partial f(x_3,x_3}{\partial x_3^2\partial x_3^1}
\end{array}\right]$$

Invariante Ausdrücke eingefügt:

$$R = \left[\begin{array}{c|cccccccccc}
 & x1 & x1/p1 & x1/p2 & x2 & x2/p1 & x2/p2 & x2/p3 & x3 & x3/p1 \\ \hline
x1 & 1+e^{\lambda} & 0 & 0 & f\left(x_1,x_2\right) & \frac{\partial f(x_1,x_2)}{\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_2^3} & f\left(x_1,x_3\right) & \frac{\partial f(x_1,x_3}{\partial x_3^1} \\
x1/p1 & 0 & e^{\theta_1} & 0 & \frac{\partial f(x_1,x_2)}{\partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^3} & \frac{\partial f(x_1,x_3)}{\partial x_1^1} & \frac{\partial f(x_1,x_3}{\partial x_1^1 \partial x_3^1} \\
x1/p2 & 0 & 0 & e^{\theta_2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^3} & \frac{\partial f(x_1,x_3)}{\partial x_1^2} & \frac{\partial f(x_1,x_3}{\partial x_1^2 \partial x_3^1} \\
x2 & f\left(x_2,x_1\right) & \frac{\partial f(x_2,x_1)}{\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_1^2} & 1+e^{\lambda} & 0 & 0 & 0 & f\left(x_2,x_3\right) & \frac{\partial f(x_2,x_3}{\partial x_3^1} \\
x2/p1 & \frac{\partial f(x_2,x_1)}{\partial x_2^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1 \partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1 \partial x_1^2} & 0 & e^{\theta_1} & 0 & 0 & \frac{\partial f(x_2,x_3)}{\partial x_2^1} & \frac{\partial f(x_2,x_3}{\partial x_2^1 \partial x_3^1} \\
x2/p2 & \frac{\partial f(x_2,x_1)}{\partial x_2^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^2 \partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^2 \partial x_1^2} & 0 & 0 & e^{\theta_2} & 0 & \frac{\partial f(x_2,x_3)}{\partial x_2^2} & \frac{\partial f(x_2,x_3}{\partial x_2^2 \partial x_3^1} \\
x2/p3 & \frac{\partial f(x_2,x_1)}{\partial x_2^3} & \frac{\partial f(x_2,x_1)}{\partial x_2^3 \partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^3 \partial x_1^2} & 0 & 0 & 0 & e^{\theta_3} & \frac{\partial f(x_2,x_3)}{\partial x_2^3} & \frac{\partial f(x_2,x_3}{\partial x_2^3 \partial x_3^1} \\
x3 & f\left(x_3,x_1\right) & \frac{\partial f(x_3,x_1)}{\partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_1^2} & f\left(x_3,x_2\right) & \frac{\partial f(x_3,x_2)}{\partial x_2^1} & \frac{\partial f(x_3,x_2)}{\partial x_2^2} & \frac{\partial f(x_3,x_2)}{\partial x_2^3} & 1+e^{\lambda} & 0 \\
x3/p1 & \frac{\partial f(x_3,x_1)}{\partial x_3^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^1 \partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^1 \partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^1} & \frac{\partial f(x_3,x_2)}{\partial x_3^1 \partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_3^1 \partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_3^1 \partial x_2^3} & 0 & e^{\theta_1} \\
x3/p2 & \frac{\partial f(x_3,x_1)}{\partial x_3^2} & \frac{\partial f(x_3,x_1)}{\partial x_3^2 \partial x_1^1} & \frac{\partial f(x_3,x_1)}{\partial x_3^2 \partial x_1^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^2} & \frac{\partial f(x_3,x_2)}{\partial x_3^2 \partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_3^2 \partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_3^2 \partial x_2^3} & 0 & 0
\end{array}\right]$$

$$R = \left[\begin{array}{c|cccccc}
 & x1 & x2 & x1/p2 & x1/p1 & x2/p2 & x2/p1 \\ \hline
x1 & f\left(x_1,x_1\right) & f\left(x_1,x_2\right) & \frac{\partial f(x_1,x_1)}{\partial x_1^2} & \frac{\partial f(x_1,x_1)}{\partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_2^1} \\
x2 & f\left(x_2,x_1\right) & f\left(x_2,x_2\right) & \frac{\partial f(x_2,x_1)}{\partial x_1^2} & \frac{\partial f(x_2,x_1)}{\partial x_1^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1} \\
x1/p2 & \frac{\partial f(x_1,x_1)}{\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2} & \frac{\partial f(x_1,x_1)}{\partial x_1^2 \partial x_1^2} & \frac{\partial f(x_1,x_1)}{\partial x_1^2 \partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^1} \\
x1/p1 & \frac{\partial f(x_1,x_1)}{\partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1} & \frac{\partial f(x_1,x_1)}{\partial x_1^1 \partial x_1^2} & \frac{\partial f(x_1,x_1)}{\partial x_1^1 \partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^1} \\
x2/p2 & \frac{\partial f(x_2,x_1)}{\partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^2 \partial x_1^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^2 \partial x_1^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^2 \partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1 \partial x_2^2} \\
x2/p1 & \frac{\partial f(x_2,x_1)}{\partial x_2^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1 \partial x_1^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^1 \partial x_1^1} & \frac{\partial f(x_2,x_2)}{\partial x_2^1 \partial x_2^2} & \frac{\partial f(x_2,x_2)}{\partial x_2^1 \partial x_2^1}
\end{array}\right]$$

$$R = \left[\begin{array}{c|cccccc}
 & x1 & x1/p1 & x1/p2 & x2 & x2/p1 & x2/p2 \\ \hline
x1 & 1+e^{\lambda} & 0 & 0 & f\left(x_1,x_2\right) & \frac{\partial f(x_1,x_2)}{\partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_2^2} \\
x1/p1 & 0 & e^{\theta_1} & 0 & \frac{\partial f(x_1,x_2)}{\partial x_1^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^1 \partial x_2^2} \\
x1/p2 & 0 & 0 & e^{\theta_2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^1} & \frac{\partial f(x_1,x_2)}{\partial x_1^2 \partial x_2^2} \\
x2 & f\left(x_2,x_1\right) & \frac{\partial f(x_2,x_1)}{\partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_1^2} & 1+e^{\lambda} & 0 & 0 \\
x2/p1 & \frac{\partial f(x_2,x_1)}{\partial x_2^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1 \partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^1 \partial x_1^2} & 0 & e^{\theta_1} & 0 \\
x2/p2 & \frac{\partial f(x_2,x_1)}{\partial x_2^2} & \frac{\partial f(x_2,x_1)}{\partial x_2^2 \partial x_1^1} & \frac{\partial f(x_2,x_1)}{\partial x_2^2 \partial x_1^2} & 0 & 0 & e^{\theta_2}
\end{array}\right]$$

$$f(x,y) = f(y,x) \Longrightarrow Matrix\ ist\ symmetrisch$$

Die 2ten Ableitungen sind unabhängig von der Reihenfolge der Ableitung:

$$\frac{\partial f\left(x_2,x_1\right)}{\partial x_2^2 \partial x_1^1} = \frac{\partial f\left(x_2,x_1\right)}{\partial x_2^1 \partial x_1^2}$$

bei Gauss:

$$\frac{\partial f\left(x_{2,}x_1\right)}{\partial x_1^1} = -\frac{\partial f\left(x_{2,}x_1\right)}{\partial x_2^1}$$

r-Vektor Bildungsvorschrift: Wie RMatrix Zeile/Spalte 1 nur mit Inputvektor gebildet

A.4 Korrelationsfunktionen

Test

Gauss (Theta quadrat, um negative Thetas direkt zu vermeiden. Der RPROP muss dann nicht mehr nach unten begrenzt werden):

$$f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) = e^{-\frac{1}{2}\sum^{k}\left(\theta_k^2 |x_{1k}-x_{2k}|^2\right)} + \delta\lambda^2$$

$$\frac{\partial f}{\partial \theta_d} = f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\left(-\theta_d \left|x_{1d}-x_{2d}\right|^2\right)$$

$$\frac{\partial f}{\partial \lambda} = 2\lambda\delta_{ij} \Rightarrow \frac{\partial R}{\partial \lambda} = 2\lambda E$$

Schlecht, da die Ableitung bei $\theta_d = 0$ auch Null ist. Ein Gradientenverfahren kommt dort also nicht mehr von selbst raus.

Daher wird eine andere Funktion verwendet:

$$f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) = e^{-\frac{1}{2}\sum^{k}\left(e^{\theta_k} |x_{1k}-x_{2k}|^2\right)} + \delta e^{\lambda}$$

$$\frac{\partial f}{\partial \theta_d} = f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\left(-\frac{1}{2}e^{\theta_d}\left|x_{1d}-x_{2d}\right|^2\right)$$

$$\frac{\partial f}{\partial \lambda} = e^{\lambda}\delta \Rightarrow \frac{\partial R}{\partial \lambda} = e^{\lambda}E$$

$$\frac{\partial f}{\partial x_1^k} = \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) - \delta e^{\lambda}\right)\left(-e^{\theta_k}\left(x_{1k} - x_{2k}\right)\right)$$

$$\frac{\partial f}{\partial x_2^k} = \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) - \delta e^{\lambda}\right)\left(e^{\theta_k}\left(x_{1k} - x_{2k}\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^l} = \begin{cases} -e^{\theta_l + \theta_k}\left(x_{1l} - x_{2l}\right)\left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) & k \neq l \\ e^{\theta_k}\left(1 - \left(x_{1k} - x_{2k}\right)^2 e^{\theta_k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) & k = l \end{cases}$$

Algorithmisch besser:

$$\frac{\partial f}{\partial x_1^k \partial x_2^l} = \begin{cases} -e^{\theta_l}\left(x_{1l} - x_{2l}\right)\frac{\partial f}{\partial x_2^k} & k \neq l \\ \left(e^{\theta_k} - \left[\left(x_{1k} - x_{2k}\right)e^{\theta_k}\right]^2\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) & k = l \end{cases}$$

Falls

$$\frac{\partial f\left(\overrightarrow{x}_1, \overrightarrow{x}_1\right)}{\partial x_1^k} = 0$$

$$\frac{\partial f\left(\overrightarrow{x}_2, \overrightarrow{x}_2\right)}{\partial x_2^k} = 0$$

$$\frac{\partial f\left(\overrightarrow{x}_1, \overrightarrow{x}_1\right)}{\partial x_1^k \partial x_1^l} = \begin{cases} 0 & k \neq l \\ e^{\theta_k} & k = l \end{cases}$$

Berechnet sind in der vorherigen Funktion bereits

$$e^{\theta_k}, f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right), \left(x_{1k} - x_{2k}\right), e^{\theta_k}$$

Diese Werte können einfach übergeben werden und man spart dadurch 2x die exp Funktion, eine Differenz, und 2 Multiplikationen

Nebenrechnung

$$\frac{\partial f}{\partial x_1^k \partial x_2^l} = \frac{\partial}{\partial x_2^l}\left(-e^{\theta_k}\left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^k} = -e^{\theta_k}\left(\frac{\partial}{\partial x_2^k}\left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^k} = -e^{\theta_k}\left(\frac{\partial\left(x_{1k} - x_{2k}\right)}{\partial x_2^k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + \left(x_{1k} - x_{2k}\right)\frac{\partial f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)}{\partial x_2^k}\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^k} = -e^{\theta_k}\left(-f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\left(e^{\theta_k}\left(x_{1k} - x_{2k}\right)\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^k} = -e^{\theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\left(\left(x_{1k} - x_{2k}\right)^2 e^{\theta_k} - 1\right)$$

Erste Ableitung nach Theta ableiten:

x1 k=p

$$\frac{\partial f}{\partial x_1^k \theta_k} = \left(x_{1k} - x_{2k}\right)\frac{\partial}{\partial \theta_k}\left(-e^{\theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \theta_k} = -\left(x_{1k} - x_{2k}\right)\left(\frac{\partial}{\partial \theta_k} e^{\theta_k} * f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + e^{\theta_k}\frac{\partial}{\partial \theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \theta_k} = -e^{\theta_k} \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(1 - \frac{1}{2} e^{\theta_k} \left(x_{1k} - x_{2k}\right)^2\right)$$

$$\frac{\partial f}{\partial x_1^k \theta_k} = -\left(x_{1k} - x_{2k}\right) e^{\theta_k} \frac{\partial}{\partial \theta_l} \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

x1 k!=p

$$\frac{\partial f}{\partial x_1^k \theta_p} = -\left(x_{1k} - x_{2k}\right) e^{\theta_k} \frac{\partial}{\partial \theta_p} \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \theta_p} = \frac{1}{2} \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(e^{\theta_p + \theta_k} \left(x_{1p} - x_{2p}\right)^2\right)$$

x2 k=p

$$\frac{\partial f}{\partial x_2^k \theta_k} = \left(x_{1k} - x_{2k}\right) \frac{\partial}{\partial \theta_k} \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) e^{\theta_k}\right)$$

$$\frac{\partial f}{\partial x_2^k \theta_k} = \left(x_{1k} - x_{2k}\right) \left(\frac{\partial}{\partial \theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) e^{\theta_k} + f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \frac{\partial}{\partial \theta_k} e^{\theta_k}\right)$$

$$\frac{\partial f}{\partial x_2^k \theta_k} = \left(x_{1k} - x_{2k}\right) \left(-\frac{1}{2} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(x_{1k} - x_{2k}\right)^2 e^{2\theta_k} + f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) e^{\theta_k}\right)$$

$$\frac{\partial f}{\partial x_2^k \theta_k} = f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(x_{1k} - x_{2k}\right) e^{\theta_k} \left(1 - \frac{1}{2} \left(x_{1k} - x_{2k}\right)^2 e^{\theta_k}\right)$$

x2 k!=p

$$\frac{\partial f}{\partial x_2^k \theta_p} = \left(x_{1k} - x_{2k}\right) e^{\theta_k} \frac{\partial}{\partial \theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)$$

$$\frac{\partial f}{\partial x_2^k \theta_p} = -\frac{1}{2} e^{\theta_k + \theta_p} \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(x_{1p} - x_{2p}\right)^2$$

Zweite Ableitung nach Theta ableiten:

k=l=p

$$\frac{\partial f}{\partial x_1^p \partial x_2^p \theta_p} = \frac{\partial}{\partial \theta_p} \left(\left(e^{\theta_p} - \left(x_{1p} - x_{2p}\right)^2 e^{2\theta_p}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$= \frac{\partial}{\partial \theta_p} \left(e^{\theta_p} - \left(x_{1p} - x_{2p}\right)^2 e^{2\theta_p}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + \left(e^{\theta_p} - \left(x_{1p} - x_{2p}\right)^2 e^{2\theta_p}\right) \frac{\partial}{\partial \theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)$$

$$= \left(e^{\theta_p} - \left(x_{1p} - x_{2p}\right)^2 2e^{2\theta_p}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + \left(e^{\theta_p} - \left(x_{1p} - x_{2p}\right)^2 e^{2\theta_p}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(-\frac{1}{2} e^{\theta_p} \left(x_{1p} - x_{2p}\right)^2\right)$$

$$= e^{\theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(1 - \left(x_{1p} - x_{2p}\right)^2 2e^{\theta_p}\right) - \frac{1}{2} e^{\theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(1 - \left(x_{1p} - x_{2p}\right)^2 e^{\theta_p}\right) \left(e^{\theta_p} \left(x_{1p} - x_{2p}\right)^2\right)$$

$$= e^{\theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left[\left(1 - \left(x_{1p} - x_{2p}\right)^2 2e^{\theta_p}\right) - \frac{1}{2} e^{\theta_p} \left(1 - \left(x_{1p} - x_{2p}\right)^2 e^{\theta_p}\right) \left(x_{1d} - x_{2d}\right)^2\right]$$

k=l!=p

$$\frac{\partial f}{\partial x_1^k \partial x_2^k \theta_p} = \frac{\partial}{\partial \theta_p} \left(\left(e^{\theta_k} - \left(x_{1k} - x_{2k}\right)^2 e^{2\theta_k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$= \left(e^{\theta_k} - \left(x_{1k} - x_{2k}\right)^2 e^{2\theta_k}\right) \frac{\partial}{\partial \theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^k \theta_p} = -\frac{1}{2} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(e^{\theta_k} - \left(x_{1k} - x_{2k}\right)^2 e^{2\theta_k}\right) \left(e^{\theta_p} \left(x_{1p} - x_{2p}\right)^2\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^k \theta_p} = -\frac{1}{2} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) e^{\theta_k} e^{\theta_p} \left(1 - \left(x_{1k} - x_{2k}\right)^2 e^{\theta_k}\right) \left(x_{1p} - x_{2p}\right)^2$$

k!=l; k=p

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_k} = -e^{\theta_l} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) \frac{\partial}{\partial \theta_k} \left(e^{\theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$= -e^{\theta_l} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) \left(\frac{\partial}{\partial \theta_k} e^{\theta_k} * f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + e^{\theta_k} \frac{\partial}{\partial \theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$= -e^{\theta_l} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) \left(e^{\theta_k} * f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) + e^{\theta_k} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(-\frac{1}{2} e^{\theta_k} \left(x_{1k} - x_{2k}\right)^2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_k} = -e^{\theta_l + \theta_k} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(1 - \frac{1}{2} e^{\theta_k} \left(x_{1k} - x_{2k}\right)^2\right)$$

k!=l; l=p

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_l} = -e^{\theta_k} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) \frac{\partial}{\partial \theta_l} \left(e^{\theta_l} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_l} = -e^{\theta_k + \theta_l} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(1 - \frac{1}{2} e^{\theta_l} \left(x_{1l} - x_{2l}\right)^2\right)$$

k!=l; k!=p; l!=p

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_p} = -e^{\theta_l + \theta_k} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) \frac{\partial}{\partial \theta_p} f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_p} = -e^{\theta_l + \theta_k} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(-\frac{1}{2} e^{\theta_p} \left(x_{1p} - x_{2p}\right)^2\right)$$

$$\frac{\partial f}{\partial x_1^k \partial x_2^l \theta_p} = \frac{1}{2} e^{\theta_l + \theta_k + \theta_p} \left(x_{1l} - x_{2l}\right) \left(x_{1k} - x_{2k}\right) f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) \left(x_{1p} - x_{2p}\right)^2$$

Wenn x1=x2:

$$\frac{\partial f}{\partial x_1^k \theta_k} = 0$$

$$\frac{\partial f}{\partial x_1^k \theta_l} = 0$$

$$\frac{\partial f}{\partial x_1^p \partial x_1^p \theta_p} = e^{\theta_p}$$

$$\frac{\partial f}{\partial x_1^k \partial x_1^k \theta_p} = 0$$

$$\frac{\partial f}{\partial x_1^k \partial x_1^l \theta_k} = 0$$

$$\frac{\partial f}{\partial x_1^k \partial x_1^l \theta_l} = 0$$

$$\frac{\partial f}{\partial x_1^k \partial x_1^l \theta_p} = 0$$

Exponential Ansatz:

$$f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) = e^{-\frac{1}{2}\sum^{k}\left(e^{\theta_k}|x_{1k} - x_{2k}|^p\right)} + \delta e^{\lambda}$$

$$\frac{\partial f}{\partial \theta_d} = f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right)\left(-\frac{1}{2}e^{\theta_k}\left|x_{1d} - x_{2d}\right|^p\right)$$

$$\frac{\partial f}{\partial p} = \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) - \delta e^{\lambda}\right)\left(-\frac{1}{2}\frac{\partial}{\partial p}\left(\sum^{k}\left(e^{\theta_k}\left|x_{1k} - x_{2k}\right|^p\right)\right)\right)$$

$$\frac{\partial f}{\partial p} = \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) - \delta e^{\lambda}\right)\left(-\frac{1}{2}\sum^{k} e^{\theta_k}\frac{\partial}{\partial p}\left(\left|x_{1k} - x_{2k}\right|^p\right)\right)$$

$$\frac{\partial f}{\partial p} = \left(f\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) - \delta e^{\lambda}\right)\left(-\frac{1}{2}\sum^{k} e^{\theta_k}\left(\left|x_{1k} - x_{2k}\right|^p\right) log\left(\left|x_{1k} - x_{2k}\right|\right)\right)$$

$$\frac{\partial f}{\partial \lambda} = e^{\lambda}\delta \Rightarrow \frac{\partial R}{\partial \lambda} = e^{\lambda}E$$

A.4.1 Kubischer Spline:

$$f\left(x_1, x_2\right) = \prod_{j=1}^{N} p_j + \delta e^{\lambda}$$

$$\xi_j = e^{\theta_j}\left|x_{1j} - x_{2j}\right|$$

$$p_j\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) = \begin{cases} 1 - \frac{3}{a}\xi_j^2 + \frac{1+a}{a^2}\xi_j^3 & , 0 \leq \xi_j \leq a \\ \frac{1}{1-a}\left(1 - \xi_j\right)^3 & , a < \xi_j < 1 \\ 0 & , \xi_j > 1 \end{cases}$$

$$\frac{\partial f}{\partial \theta_j} = \frac{\partial p_j}{\partial \theta_j} \prod_{i \neq j}^{N} p_i$$

$$\frac{\partial p_j}{\partial \theta_j} = \begin{cases} -\frac{6}{a} \left| x_{1j} - x_{2j} \right|^2 e^{2\theta_j} + 3\frac{1+a}{a^2} \left| x_{1j} - x_{2j} \right|^3 e^{3\theta_j} & , 0 \leq \xi_j \leq a \\ -\frac{3}{1-a} \left| x_{1j} - x_{2j} \right| e^{\theta_j} \left(1 - \left| x_{1j} - x_{2j} \right| e^{\theta_j} \right)^2 & , a < \xi_j < 1 \\ 0 & , \xi_j > 1 \end{cases}$$

$$\frac{\partial p_j}{\partial \theta_j} = \begin{cases} 3\xi_j^2 \left(\frac{1+a}{a^2} \xi_j - \frac{2}{a} \right) & , 0 \leq \xi_j \leq a \\ -\frac{3}{1-a} \xi_j \left(1 - \xi_j \right)^2 & , a < \xi_j < 1 \\ 0 & , \xi_j > 1 \end{cases}$$

$$\frac{\partial f}{\partial a} = \sum_{j=1}^{N} \left(\frac{\partial p_j}{\partial a} \prod_{i \neq j}^{N} p_i \right)$$

$$\frac{\partial p_j}{\partial a} = \begin{cases} \frac{3}{a^2} \xi_j^2 - \frac{a+2}{a^3} \xi_j^3 & , 0 \leq \xi_j \leq a \\ \frac{1}{(1-a)^2} \left(1 - \xi_j \right)^3 , & a < \xi_j < 1 \\ 0 & , \xi_j > 1 \end{cases}$$

$$\frac{\partial f}{\partial \lambda} = e^{\lambda} \delta \Rightarrow \frac{\partial R}{\partial \lambda} = e^{\lambda} E$$

A.4.1.1 Kubischer Spline nach x

$$\frac{\partial f\left(x_1, x_2\right)}{\partial x_1^k} = \frac{\partial p_k}{\partial x_1^k} \prod_{j=1; j \neq k}^{N} p_j$$

$$p_j\left(\overrightarrow{x}_1, \overrightarrow{x}_2\right) = \begin{cases} 1 - \frac{3}{a}\left(e^{\theta_j}\left|x_{1j} - x_{2j}\right|\right)^2 + \frac{1+a}{a^2}\left(e^{\theta_j}\left|x_{1j} - x_{2j}\right|\right)^3 & , 0 \leq \xi_j \leq a \\ \frac{1}{1-a}\left(1 - e^{\theta_j}\left|x_{1j} - x_{2j}\right|\right)^3 & , a < \xi_j < 1 \\ 0 & , \xi_j > 1 \end{cases}$$

$$\frac{\partial p_k}{\partial x_1^k} = \begin{cases} -\frac{3}{a}2e^{\theta_k}\left|x_{1k} - x_{2k}\right| e^{\theta_k} sgn\left(x_{1k} - x_{2k}\right) 1 + \frac{1+a}{a^2}3e^{2\theta_k}\left|x_{1k} - x_{2k}\right|^2 e^{\theta_k} sgn\left(x_{1k} - x_{2k}\right. \\ \frac{1}{1-a}3\left(1 - e^{\theta_k}\left|x_{1k} - x_{2k}\right|\right)^2\left(-e^{\theta_k} sgn\left(x_{1k} - x_{2k}\right) 1\right) \\ 0 \end{cases}$$

$$\frac{\partial p_k}{\partial x_2^k} = \begin{cases} -\frac{3}{a}e^{\theta_k}2\left|x_{1k} - x_{2k}\right| e^{\theta_k} sgn\left(x_{1k} - x_{2k}\right)(-1) + \frac{1+a}{a^2}3e^{2\theta_k}\left|x_{1k} - x_{2k}\right|^2 e^{\theta_k} sgn\left(x_{1k}\right. \\ \frac{1}{1-a}3\left(1 - e^{\theta_k}\left|x_{1k} - x_{2k}\right|\right)^2\left(-e^{\theta_k} sgn\left(x_{1k} - x_{2k}\right)(-1)\right) \\ 0 \end{cases}$$

$$\frac{\partial p_k}{\partial x_1^k} = \begin{cases} sgn\left(x_{1k} - x_{2k}\right)\frac{3}{a}e^{\theta_k}\xi_k\left(\frac{1+a}{a}\xi - 2\right) & , 0 \leq \xi_k \leq a \\ -sgn\left(x_{1k} - x_{2k}\right)\frac{3}{1-a}e^{\theta_k}\left(1 - \xi_k\right)^2 & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$\frac{\partial p_k}{\partial x_2^k} = \begin{cases} -sgn\left(x_{1k} - x_{2k}\right)\frac{3}{a}e^{\theta_k}\xi_k\left(\frac{1+a}{a}\xi - 2\right) & , 0 \leq \xi_k \leq a \\ sgn\left(x_{1k} - x_{2k}\right)\frac{3}{1-a}e^{\theta_k}\left(1 - \xi_k\right)^2 & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

A.4.1.2 2te Ableitung Kubischer Spline nach x

$$\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k x_2^k} = \frac{\partial p_k^2}{\partial x_1^k x_2^k}\prod_{j=1;j\neq k}^{N} p_j$$

$$\frac{\partial p_k^2}{\partial x_1^k x_2^k} = \begin{cases} sgn\left(x_{1k} - x_{2k}\right) \frac{3}{a} e^{2\theta_k} \frac{\partial}{\partial x_2^k} \left[\left| x_{1k} - x_{2k} \right| \left(\frac{1+a}{a} e^{\theta_k} \left| x_{1k} - x_{2k} \right| - 2 \right) \right] & , 0 \leq \xi_k \leq a \\ -\frac{3}{1-a} e^{\theta_k} sgn\left(x_{1k} - x_{2k}\right) \frac{\partial}{\partial x_2^k} \left[\left(1 - e^{\theta_k} \left| x_{1k} - x_{2k} \right| \right)^2 \right] & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$\frac{\partial p_k^2}{\partial x_1^k x_2^k} = \begin{cases} -\frac{6}{a} e^{2\theta_k} \left[\frac{1+a}{a} \xi_k - 1 \right] & , 0 \leq \xi_k \leq a \\ -\frac{6}{1-a} e^{2\theta_k} \left[(1 - \xi_k) \right] & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k x_2^l} = \frac{\partial p_k \partial p_l}{\partial x_1^k \partial x_2^l} \prod_{j=1; j \neq k; j \neq l}^{N} p_j$$

$$\frac{\partial p_l}{\partial x_2^l} = \begin{cases} -sgn\left(x_{1l} - x_{2l}\right) \frac{3}{a} e^{\theta_l} \xi \left(\frac{1+a}{a} \xi_l - 2 \right) & , 0 \leq \xi_l \leq a \\ sgn\left(x_{1l} - x_{2l}\right) \frac{3}{1-a} e^{\theta_l} \left(1 - \xi_l\right)^2 & , a < \xi_l < 1 \\ 0 & , \xi_l > 1 \end{cases}$$

$$\frac{\partial^2 f\left(x_1, x_1\right)}{\partial x_1^k x_1^k} = \frac{6}{a} e^{2\theta_k}$$

$$\frac{\partial^2 f\left(x_1, x_1\right)}{\partial x_1^k x_1^l} = 0$$

$$\frac{\partial f\left(x_1, x_1\right)}{\partial x_1^k} = 0$$

A.4.1.3 Kubischer Spline nach Theta, erste Ableitung:

$$p_j = 0 \Longrightarrow \frac{\partial f}{\partial \theta_p} = 0$$

$$\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k \partial \theta_k} = \frac{\partial}{\partial \theta_k}\left(\frac{\partial p_k}{\partial x_1^k} \prod_{j=1; j \neq k}^{N} p_j\right)$$

$$= \frac{\partial}{\partial \theta_k}\left(\frac{\partial p_k}{\partial x_1^k}\right) \prod_{j=1; j \neq k}^{N} p_j$$

$$\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k \partial \theta_p} = \frac{\partial}{\partial \theta_p}\left(\frac{\partial p_k}{\partial x_1^k} \prod_{j=1; j \neq k}^{N} p_j\right)$$

$$= \frac{\partial p_p}{\partial \theta_p} \frac{\partial p_k}{\partial x_1^k} \prod_{j=1; j \neq k; j \neq p}^{N} p_j$$

$$\frac{\partial}{\partial \theta_k}\left(\frac{\partial^2 f\left(x_1, x_1\right)}{\partial x_1^k x_1^k}\right) = \frac{12}{a} e^{2\theta_k}$$

$$\frac{\partial}{\partial \theta_p}\left(\frac{\partial^2 f\left(x_1, x_1\right)}{\partial x_1^k x_1^k}\right) = 0$$

$$\frac{\partial}{\partial \theta_k}\left(\frac{\partial^2 f\left(x_1, x_1\right)}{\partial x_1^k x_1^l}\right) = 0$$

$$\frac{\partial}{\partial \theta_k}\left(\frac{\partial f\left(x_1, x_1\right)}{\partial x_1^k}\right) = 0$$

A.4.1.4 Kubischer Spline nach Theta, zweite Ableitung:

p=k=l

$$\frac{\partial}{\partial \theta_k}\left(\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k \partial x_2^k}\right) = \frac{\partial}{\partial \theta_k}\left(\frac{\partial^2 p_k}{\partial x_1^k \partial x_2^k} \prod_{j=1; j\neq k}^{N} p_j\right)$$

$$= \frac{\partial}{\partial \theta_k}\left(\frac{\partial^2 p_k}{\partial x_1^k \partial x_2^k}\right) \prod_{j=1; j\neq k}^{N} p_j$$

p!=k=l

$$\frac{\partial}{\partial \theta_p}\left(\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k \partial x_2^k}\right) = \frac{\partial}{\partial \theta_p}\left(\frac{\partial^2 p_k}{\partial x_1^k \partial x_2^k} \prod_{j=1; j\neq k}^{N} p_j\right)$$

$$= \frac{\partial^2 p_k}{\partial x_2^k \partial x_1^k}\frac{\partial p_p}{\partial \theta_p} \prod_{j=1; j\neq k; j\neq p}^{N} p_j$$

p=k!=l

$$\frac{\partial}{\partial \theta_k}\left(\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k \partial x_2^l}\right) = \frac{\partial}{\partial \theta_k}\left(\frac{\partial p_k \partial p_l}{\partial x_1^k \partial x_2^l} \prod_{j=1; j\neq k; j\neq l}^{N} p_j\right)$$

$$= \frac{\partial}{\partial \theta_k}\left(\frac{\partial p_k}{\partial x_1^k}\right)\left(\frac{\partial p_l}{\partial x_2^l} \prod_{j=1; j\neq k; j\neq l}^{N} p_j\right)$$

p=l ; k!=l

$$\frac{\partial}{\partial \theta_l}\left(\frac{\partial^2 f\left(x_1, x_2\right)}{\partial x_1^k x_2^l}\right) = \frac{\partial}{\partial \theta_l}\left(\frac{\partial p_k \partial p_l}{\partial x_1^k \partial x_2^l} \prod_{j=1; j\neq k; j\neq l}^{N} p_j\right)$$

$$= \frac{\partial}{\partial \theta_l}\left(\frac{\partial p_l}{\partial x_2^l}\right)\frac{\partial p_k}{\partial x_1^k}\prod_{j=1;j\neq k;j\neq l}^{N} p_j$$

p!=k!=l

$$\frac{\partial}{\partial \theta_p}\left(\frac{\partial^2 f\left(x_1,x_2\right)}{\partial x_1^k x_2^l}\right) = \frac{\partial}{\partial \theta_p}\left(\frac{\partial p_k \partial p_l}{\partial x_1^k \partial x_2^l}\prod_{j=1;j\neq k;j\neq l}^{N} p_j\right)$$

$$= \frac{\partial p_k \partial p_l}{\partial x_1^k \partial x_2^l}\frac{\partial p_p}{\partial \theta_p}\prod_{j=1;j\neq k;j\neq l;j\neq p}^{N} p_j$$

A.4.1.5 Kubischer Spline nach a, erste Ableitung:

$$\frac{\partial^2 f\left(x_1,x_2\right)}{\partial x_1^k \partial a} = \frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\prod_{j=1;j\neq k}^{N} p_j\right)$$

$$= \frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right)\prod_{j=1;j\neq k}^{N} p_j + \frac{\partial p_k}{\partial x_1^k}\frac{\partial}{\partial a}\left(\prod_{j=1;j\neq k}^{N} p_j\right)$$

Für pj != 0, wenn pj = 0 ist die ganze Ableitung 0

$$= \frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right)\prod_{j=1;j\neq k}^{N} p_j + \frac{\partial p_k}{\partial x_1^k}\left(\prod_{j=1;j\neq k}^{N} p_j\right)\left(\sum_{j=1;j\neq k}^{N}\frac{\frac{\partial}{\partial a}p_j}{p_j}\right)$$

$$= \left(\prod_{j=1;j\neq k}^{N} p_j\right)\left(\frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right) + \frac{\partial p_k}{\partial x_1^k}\left(\sum_{j=1;j\neq k}^{N}\frac{\frac{\partial p_j}{\partial a}}{p_j}\right)\right)$$

$$\frac{\partial}{\partial a}\left(\frac{\partial^2 f\left(x_1,x_1\right)}{\partial x_1^k x_1^k}\right) = -\frac{6}{a^2}e^{2\theta_k}$$

$$\frac{\partial}{\partial a}\left(\frac{\partial^2 f\left(x_1,x_1\right)}{\partial x_1^k x_1^k}\right)=0$$

$$\frac{\partial}{\partial a}\left(\frac{\partial^2 f\left(x_1,x_1\right)}{\partial x_1^k x_1^l}\right)=0$$

$$\frac{\partial}{\partial a}\left(\frac{\partial f\left(x_1,x_1\right)}{\partial x_1^k}\right)=0$$

A.4.1.6 Kubischer Spline nach a, zweite Ableitung:

$$\frac{\partial}{\partial a}\left(\frac{\partial^2 f\left(x_1,x_2\right)}{\partial x_1^k x_2^k}\right)=\frac{\partial}{\partial a}\left(\frac{\partial p_k^2}{\partial x_1^k x_2^k}\prod_{j=1;j\neq k}^{N} p_j\right)$$

$$=\frac{\partial}{\partial a}\left(\frac{\partial p_k^2}{\partial x_1^k x_2^k}\right)\prod_{j=1;j\neq k}^{N} p_j+\frac{\partial p_k^2}{\partial x_1^k x_2^k}\frac{\partial}{\partial a}\left(\prod_{j=1;j\neq k}^{N} p_j\right)$$

$$=\frac{\partial}{\partial a}\left(\frac{\partial p_k^2}{\partial x_1^k x_2^k}\right)\prod_{j=1;j\neq k}^{N} p_j+\frac{\partial p_k^2}{\partial x_1^k x_2^k}\left(\prod_{j=1;j\neq k}^{N} p_j\right)\left(\sum_{j=1;j\neq k}^{N}\frac{\frac{\partial}{\partial a}p_j}{p_j}\right)$$

$$=\prod_{j=1;j\neq k}^{N} p_j\left[\frac{\partial}{\partial a}\left(\frac{\partial p_k^2}{\partial x_1^k x_2^k}\right)+\frac{\partial p_k^2}{\partial x_1^k x_2^k}\left(\sum_{j=1;j\neq k}^{N}\frac{\frac{\partial}{\partial a}p_j}{p_j}\right)\right]$$

$$\frac{\partial}{\partial a}\left(\frac{\partial^2 f\left(x_1,x_2\right)}{\partial x_1^k x_2^l}\right)=\frac{\partial}{\partial a}\left(\frac{\partial p_l}{\partial x_2^l}\frac{\partial p_k}{\partial x_1^k}\prod_{j=1;j\neq k;j\neq l}^{N} p_j\right)$$

$$=\frac{\partial}{\partial a}\left(\frac{\partial p_l}{\partial x_2^l}\right)\frac{\partial p_k}{\partial x_1^k}\prod_{j=1;j\neq k;j\neq l}^{N}p_j+\frac{\partial p_l}{\partial x_2^l}\frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right)\prod_{j=1;j\neq k;j\neq l}^{N}p_j+\frac{\partial p_l}{\partial x_2^l}\frac{\partial p_k}{\partial x_1^k}\frac{\partial}{\partial a}\left(\prod_{j=1;j\neq k;j\neq l}^{N}p_j\right)$$

$$=\prod_{j=1;j\neq k;j\neq l}^{N}p_j\left[\frac{\partial}{\partial a}\left(\frac{\partial p_l}{\partial x_2^l}\right)\frac{\partial p_k}{\partial x_1^k}+\frac{\partial p_l}{\partial x_2^l}\frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right)+\frac{\partial p_l}{\partial x_2^l}\frac{\partial p_k}{\partial x_1^k}\left(\sum_{j=1;j\neq k;j\neq l}^{N}\frac{\frac{\partial}{\partial a}p_j}{p_j}\right)\right]$$

A.4.1.7 Produktableitungen nach Theta

$$\xi_k=e^{\theta_k}\left|x_{1k}-x_{2k}\right|$$

$$\frac{\partial}{\partial\theta_k}\left(\frac{\partial p_k^2}{\partial x_1^k x_2^k}\right)=\begin{cases}\frac{\partial}{\partial\theta_k}\left(-\frac{6}{a}e^{2\theta_k}\left[\frac{1+a}{a}e^{\theta_k}\left|x_{1k}-x_{2k}\right|-1\right]\right) & ,0\leq\xi_k\leq a\\ -\frac{\partial}{\partial\theta_k}\left(\frac{6}{1-a}e^{2\theta_k}\left(1-e^{\theta_k}\left|x_{1k}-x_{2k}\right|\right)\right) & ,a<\xi_k<1\\ 0 & ,\xi_k>1\end{cases}$$

$$=\begin{cases}-\frac{6}{a}\frac{\partial}{\partial\theta_k}\left(\frac{1+a}{a}e^{3\theta_k}\left|x_{1k}-x_{2k}\right|-e^{2\theta_k}\right) & ,0\leq\xi_k\leq a\\ -\frac{6}{1-a}\frac{\partial}{\partial\theta_k}\left(e^{2\theta_k}-e^{3\theta_k}\left|x_{1k}-x_{2k}\right|\right) & ,a<\xi_k<1\\ 0 & ,\xi_k>1\end{cases}$$

$$=\begin{cases}-\frac{6}{a}\left(\frac{3(1+a)}{a}e^{3\theta_k}\left|x_{1k}-x_{2k}\right|-2e^{2\theta_k}\right) & ,0\leq\xi_k\leq a\\ -\frac{6}{1-a}\left(2e^{2\theta_k}-3e^{3\theta_k}\left|x_{1k}-x_{2k}\right|\right) & ,a<\xi_k<1\\ 0 & ,\xi_k>1\end{cases}$$

$$= \begin{cases} -\frac{6}{a}e^{2\theta_k}\left(\frac{3(1+a)}{a}\xi_k - 2\right) & ,0 \leq \xi_k \leq a \\ -\frac{6}{1-a}e^{2\theta_k}\left(2 - 3\xi_k\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$\frac{\partial}{\partial\theta_k}\left(\frac{\partial p_k}{\partial x_1^k}\right) = \begin{cases} \frac{\partial}{\partial\theta_k}\left(sgn\left(x_{1k} - x_{2k}\right)\frac{3}{a}e^{\theta_k}\xi_k\left(\frac{1+a}{a}\xi_k - 2\right)\right) & ,0 \leq \xi_k \leq a \\ -\frac{\partial}{\partial\theta_k}\left(sgn\left(x_{1k} - x_{2k}\right)\frac{3}{1-a}e^{\theta_k}\left(1-\xi_k\right)^2\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$= \begin{cases} sgn\left(x_{1k} - x_{2k}\right)\frac{3}{a}\left|x_{1k} - x_{2k}\right|\frac{\partial}{\partial\theta_k}\left(\frac{1+a}{a}e^{3\theta_k}\left|x_{1k} - x_{2k}\right| - 2e^{2\theta_k}\right) & ,0 \leq \xi_k \leq a \\ -sgn\left(x_{1k} - x_{2k}\right)\frac{3}{1-a}\frac{\partial}{\partial\theta_k}\left(e^{\theta_k}\left(1-\xi_k\right)\left(1-\xi_k\right)\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$= \begin{cases} sgn\left(x_{1k} - x_{2k}\right)\frac{3}{a}\left|x_{1k} - x_{2k}\right|\left(\frac{3(1+a)}{a}e^{3\theta_k}\left|x_{1k} - x_{2k}\right| - 4e^{2\theta_k}\right) \\ -sgn\left(x_{1k} - x_{2k}\right)\frac{3}{1-a}\left(\frac{\partial}{\partial\theta_k}\left(e^{\theta_k}\right)\left(1-\xi_k\right)\left(1-\xi_k\right) + e^{\theta_k}\frac{\partial}{\partial\theta_k}\left(1-\xi_k\right)\left(1-\xi_k\right) + e^{\theta_k}\left(1\right.\right. \\ 0 \end{cases}$$

$$= \begin{cases} sgn\left(x_{1k} - x_{2k}\right)\frac{3}{a}\left|x_{1k} - x_{2k}\right|\left(\frac{3(1+a)}{a}e^{3\theta_k}\left|x_{1k} - x_{2k}\right| - 4e^{2\theta_k}\right) & ,0 \leq \xi_k \leq a \\ -sgn\left(x_{1k} - x_{2k}\right)\frac{3}{1-a}\left(e^{\theta_k}\left(1-\xi_k\right)\left(1-\xi_k\right) - 2e^{\theta_k}\xi_k\left(1-\xi_k\right)\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$= \begin{cases} sgn\left(x_{1k} - x_{2k}\right) \frac{3}{a} \left|x_{1k} - x_{2k}\right| \left(\frac{3(1+a)}{a} e^{3\theta_k} \left|x_{1k} - x_{2k}\right| - 4e^{2\theta_k}\right) & , 0 \leq \xi_k \leq a \\ -sgn\left(x_{1k} - x_{2k}\right) \frac{3}{1-a} e^{\theta_k} \left(1 - \xi_k\right) \left(\left(1 - \xi_k\right) - 2\xi_k\right) & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$= \begin{cases} sgn\left(x_{1k} - x_{2k}\right) \frac{3}{a} e^{\theta_k} \xi_k \left(\frac{3(1+a)}{a} \xi_k - 4\right) & , 0 \leq \xi_k \leq a \\ -sgn\left(x_{1k} - x_{2k}\right) \frac{3}{1-a} e^{\theta_k} \left(1 - \xi_k\right) \left(1 - 3\xi_k\right) & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$\frac{\partial}{\partial \theta_p} \left(\frac{\partial p_k}{\partial x_1^k}\right) = \begin{cases} 0 & , 0 \leq \xi_k \leq a \\ 0 & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

A.4.1.8 Produktableitungen nach a

$$\frac{\partial}{\partial a} \left(\frac{\partial p_k^2}{\partial x_1^k x_2^k}\right) = \begin{cases} \frac{\partial}{\partial a} \left(-\frac{6}{a} e^{2\theta_k} \left[\frac{1+a}{a} e^{\theta_k} \left|x_{1k} - x_{2k}\right| - 1\right]\right) & , 0 \leq \xi_k \leq a \\ -\frac{\partial}{\partial a} \left(\frac{6}{1-a} e^{2\theta_k} \left(1 - e^{\theta_k} \left|x_{1k} - x_{2k}\right|\right)\right) & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$= \begin{cases} -6e^{2\theta_k} \frac{\partial}{\partial a} \left(\frac{1+a}{a^2} e^{\theta_k} \left|x_{1k} - x_{2k}\right| - \frac{1}{a}\right) & , 0 \leq \xi_k \leq a \\ -e^{2\theta_k} \left(1 - e^{\theta_k} \left|x_{1k} - x_{2k}\right|\right) \frac{\partial}{\partial a} \left(\frac{6}{1-a}\right) & , a < \xi_k < 1 \\ 0 & , \xi_k > 1 \end{cases}$$

$$= \begin{cases} -6e^{2\theta_k}\frac{1}{a^2}\left(1-\frac{2+a}{a}\xi_k\right) & ,0 \leq \xi_k \leq a \\ -e^{2\theta_k}\frac{6(1-\xi_k)}{(a-1)^2} & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$\frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right) = \begin{cases} \frac{\partial}{\partial a}\left(sgn\left(x_{1k}-x_{2k}\right)\frac{3}{a}e^{\theta_k}\xi_k\left(\frac{1+a}{a}\xi_k-2\right)\right) & ,0 \leq \xi_k \leq a \\ -\frac{\partial}{\partial a}\left(sgn\left(x_{1k}-x_{2k}\right)\frac{3}{1-a}e^{\theta_k}\left(1-\xi_k\right)^2\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$\frac{\partial}{\partial a}\left(\frac{\partial p_k}{\partial x_1^k}\right) = \begin{cases} \frac{\partial}{\partial a}\left(sgn\left(x_{1k}-x_{2k}\right)e^{\theta_k}\xi_k\left(\frac{3(1+a)}{a^2}\xi_k-\frac{6}{a}\right)\right) & ,0 \leq \xi_k \leq a \\ -\frac{\partial}{\partial a}\left(sgn\left(x_{1k}-x_{2k}\right)\frac{3}{1-a}e^{\theta_k}\left(1-\xi_k\right)^2\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

$$= \begin{cases} sgn\left(x_{1k}-x_{2k}\right)e^{\theta_k}\xi_k\left(-3\frac{(2+a)}{a^3}\xi_k+\frac{6}{a^2}\right) & ,0 \leq \xi_k \leq a \\ -\left(sgn\left(x_{1k}-x_{2k}\right)\frac{3}{(a-1)^2}e^{\theta_k}\left(1-\xi_k\right)^2\right) & ,a < \xi_k < 1 \\ 0 & ,\xi_k > 1 \end{cases}$$

A.5 Konditionierung als Nebenbedinung für das Training

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

Da R immer positiv definit sein muss

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

Der betragsmäßig maximale Eigenwerte kann mit dem Mises / Potenzverfahren gut gefunden werden. Um den minimalen zu finden nutzt man folgenden Zusammenhang:

$$\Xi_{min}\left(R\right) = \frac{1}{\Xi_{max}\left(R^{-1}\right)}$$

$\Xi_{max}\left(R^{-1}\right)$ kann mit dem Mises Verfahren ohne Probleme gefunden werden.
Die Ableitung der Konditionierung nach den Hyperparametern:

$$\frac{\partial \kappa}{\partial \theta} = \frac{\frac{\partial \Xi_{max}}{\partial \theta}\Xi_{min} - \Xi_{max}\frac{\partial \Xi_{min}}{\partial \theta}}{\Xi_{min}^2}$$

Die Eigenwerte werden wie folgt abgeleitet:

$$R\vec{x} = \Xi\vec{x}$$

$$\left(R\vec{x}\right)^T = \left(\Xi\vec{x}\right)^T$$

$$\vec{x}^T R^T = \Xi\vec{x}^T$$

$$\frac{\partial R\vec{x}}{\partial \theta} = \frac{\partial \Xi\vec{x}}{\partial \theta}$$

$$\frac{\partial R}{\partial \theta}\vec{x} + R\frac{\partial \vec{x}}{\partial \theta} = \frac{\partial \Xi}{\partial \theta}\vec{x} + \Xi\frac{\partial \vec{x}}{\partial \theta}$$

$$\vec{x}^T\frac{\partial R}{\partial \theta}\vec{x} + \vec{x}^T R\frac{\partial \vec{x}}{\partial \theta} = \vec{x}^T\frac{\partial \Xi}{\partial \theta}\vec{x} + \vec{x}^T\Xi\frac{\partial \vec{x}}{\partial \theta}$$

da die Matrix symmetrisch ist

$$R^T = R$$

$$\vec{x}^T \frac{\partial R}{\partial \theta} \vec{x} + \vec{x}^T R \frac{\partial \vec{x}}{\partial \theta} - \vec{x}^T R^T \frac{\partial \vec{x}}{\partial \theta} = \vec{x}^T \frac{\partial \Xi}{\partial \theta} \vec{x}$$

$$\vec{x}^T \frac{\partial R}{\partial \theta} \vec{x} + \vec{x}^T R \frac{\partial \vec{x}}{\partial \theta} - \vec{x}^T R \frac{\partial \vec{x}}{\partial \theta} = \vec{x}^T \frac{\partial \Xi}{\partial \theta} \vec{x}$$

$$\vec{x}^T \frac{\partial R}{\partial \theta} \vec{x} = \frac{\partial \Xi}{\partial \theta} \vec{x}^T \vec{x}$$

$$\frac{\vec{x}^T \frac{\partial R}{\partial \theta} \vec{x}}{\vec{x}^T \vec{x}} = \frac{\partial \Xi}{\partial \theta}$$

$$\frac{\partial \kappa}{\partial \theta} = \frac{\frac{\vec{x}_{max}^T \frac{\partial R}{\partial \theta} \vec{x}_{max}}{\vec{x}_{max}^T \vec{x}_{max}} \Xi_{min} - \Xi_{max} \frac{\vec{x}_{min}^T \frac{\partial R}{\partial \theta} \vec{x}_{min}}{\vec{x}_{min}^T \vec{x}_{min}}}{\Xi_{min}^2}$$

Die Eigenwerte nach dem Diagonalaufschlag abgeleitet:

$$\frac{\partial R \vec{x}}{\partial \lambda} = \frac{\partial \Xi \vec{x}}{\partial \lambda}$$

$$\frac{\vec{x}^T E e^{\lambda} \vec{x}}{\vec{x}^T \vec{x}} = \frac{\partial \Xi}{\partial \theta}$$

$$e^{\lambda} = \frac{\partial \Xi}{\partial \theta}$$

Die Gradienten der Nebenbedingung können hier sehr große werden und die Nebenbedingung tendiert dann immer gegen eine Einheitsmatrix für R. Zudem ist der Berech-

nungsaufwand relativ hoch und der Gradient basiert auf der Inversen, was widerum zu Fehlern führen kann.

A.6 Checksumme:

Eine einfache Checksumme, ob die Diagonale der Einheitsmatrix entspricht. Wenn die Invertierung nicht funktioniert hat, kommen hier typischerweise Werte > 0 raus. Dies ist ein schneller Check um die Qualität der Invertierung zu überprüfen. Da die Ableitung allerdings auch auf der Inversen beruht, ist es nicht möglich einen Gradienten für die Nebenbedinung zu finden.

$$Spur\left(RR^{-1}\right) - N = 0$$

A.7 CO-Kriging

A.7.1 Braunschweiger Model:

Ansatz von den Braunschweigern [28], nur für 2 Fidelities:

$$Z^*\left(\vec{x}_0\right) = \sum_{i=1}^{m_f} \sum_{j=1}^{n_i} w_{i,j} Z\left(\vec{x}_{i,j}\right)$$

$$Z^*\left(\vec{x}_0\right) = \sum_{i=1}^{n_{all}} w_i Z\left(\vec{x}_i\right)$$

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

$$\mathrm{MSE}\left[F\left(\vec{x}_0\right)\right] = \mathrm{E}\left[\left(Z\left(\vec{x}_0\right) - \sum_{i=1}^{m_f} \sum_{j=1}^{n_i} w_{i,j} Z\left(\vec{x}_{i,j}\right)\right)^2\right]$$

Annahme der Braunschweiger:

$$\mathrm{MSE}\left[F\left(\vec{x}_0\right)\right] = \mathrm{E}\left[\left(Z_{i=1}\left(\vec{x}_0\right) - \sum_{i=1}^{m_f} \sum_{j=1}^{n_i} w_{i,j} Z\left(\vec{x}_{i,j}\right)\right)^2\right]$$

$$E\left[Z^*\left(\vec{x}_0\right)\right] = \mathrm{E}\left[Z_{i=1}\left(\vec{x}_0\right)\right]$$

BLUE:

$$E\left[Z^*\left(\vec{x}_0\right)\right] - \mathrm{E}\left[Z_{i=1}\left(\vec{x}_0\right)\right] = 0$$

$$E\left[\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z\left(\vec{x}_{i,j}\right)\right] - \mathrm{E}\left[Z_{i=1}\left(\vec{x}_0\right)\right] = 0$$

$$\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} E\left[Z\left(\vec{x}_{i,j}\right)\right] - \mathrm{E}\left[Z_{i=1}\left(\vec{x}_0\right)\right] = 0$$

$$\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} = 1$$

MSE:

$$\mathrm{MSE}\left[F\left(\vec{x}_0\right)\right] = \mathrm{E}\left[\left(Z_{i=1}\left(\vec{x}_0\right) - \sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z\left(\vec{x}_{i,j}\right)\right)^2\right]$$

$$\mathrm{MSE}\left[F\left(\vec{x}_0\right)\right] = \mathrm{E}\left[{Z_{i=1}}^2 - 2Z_{i=1}\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j} + \left(\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right)^2\right]$$

$$\mathrm{MSE}\left[F\left(\vec{x}_0\right)\right] = \mathrm{E}\left[{Z_{i=1}}^2\right] - \mathrm{E}\left[2Z_{i=1}\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right] + \mathrm{E}\left[\left(\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right)^2\right]$$

Einschub, Verschiebungssatz:

$$E\left[XY\right] = Cov\left[X,Y\right] + E\left[X\right]E\left[Y\right]$$

Ergibt:

$$\mathrm{MSE}\left[F\left(\vec{x}_0\right)\right] = \mathrm{E}\left[{Z_{i=1}}^2\right] - 2\left(Cov\left(Z_{i=1}, \sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right) + \mathrm{E}\left[Z_{i=1}\right]\mathrm{E}\left[\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right]\right) + \mathrm{E}\left[\left(\sum_{i=1}^{m_f}\right.\right.$$

$$\text{MSE}\left[F\left(\vec{x}_0\right)\right] = \text{Var}\left[Z_{i=1}\right] + E\left[Z_{i=1}\right]^2 - 2Cov\left(Z_{i=1}, \sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right) - 2\text{E}\left[Z_{i=1}\right]\text{E}\left[\sum_{i=1}^{m_f}\sum_{j=1}^{n_i} w_{i,j} Z_{i,j}\right] + \text{Va}$$

$$R = \left[\begin{array}{c|ccccc} & x1_h & x2_h & x1_l & x2_l & x3_l \\ \hline x1_h & f\left(x_{1h,}x_{1h}\right) & f\left(x_{1h,}x_{2h}\right) & f\left(x_{1h,}x_{1l}\right) & f\left(x_{1h,}x_{2l}\right) & f\left(x_{1h,}x_{3l}\right) \\ x2_h & f\left(x_{2h,}x_{1h}\right) & f\left(x_{2h,}x_{2h}\right) & f\left(x_{2h,}x_{1l}\right) & f\left(x_{2h,}x_{2l}\right) & f\left(x_{2h,}x_{3l}\right) \\ x1_l & f\left(x_{1l,}x_{1h}\right) & f\left(x_{1l,}x_{2h}\right) & f\left(x_{1l,}x_{1l}\right) & f\left(x_{1l,}x_{2l}\right) & f\left(x_{1l,}x_{3l}\right) \\ x2_l & f\left(x_{2l,}x_{1h}\right) & f\left(x_{2l,}x_{2h}\right) & f\left(x_{2l,}x_{1l}\right) & f\left(x_{2l,}x_{2l}\right) & f\left(x_{2l,}x_{3l}\right) \\ x3_l & f\left(x_{3l,}x_{1h}\right) & f\left(x_{3l,}x_{2h}\right) & f\left(x_{3l,}x_{1l}\right) & f\left(x_{3l,}x_{2l}\right) & f\left(x_{3l,}x_{3l}\right) \end{array}\right]$$

A.8 Effizienz

Likelihood nach ys ableiten, um unwichtige Member rauszufiltern, ORdinary Kriging:

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

Spur schätzen

$$Spur\left(R\right) = E\left[\vec{d}^T R\vec{d}\right]$$

$$Spur\left(R\right) \approx \frac{1}{N}\sum \vec{d}^T R\vec{d}$$

$$\vec{d} = \left[\begin{array}{c} N\left(0,1\right) \\ \vdots \\ N\left(0,1\right) \end{array}\right]$$

Invertierung teilweise sparen:

Beim Quasi Newton Line-Search keine Gradienten berechnen und dadurch kann dort auf die Invertierung verzichtet werden. Stattdessen lösen wir das durch die Cholesky Zerlegung und einem LGS.

Nur bei einem äußeren QN Schritt, wird die Matrix invertiert und dann der Gradient damit berechnet.

Randomized Algorithms for Estimating the Trace of an Implicit Symmetric Positive Semi-Definite Matrix

A stochastic estimator of the trace of the influence matrix for laplacian smoothing spli-

nes M.F. Hutchinson a

Efficient Implementation of Gaussian Processes, Mark Gibbs David J.C. MacKay

Literaturverzeichnis

[1] L. Le Gratiet, “Bayesian Analysis of Hierarchical Multifidelity Codes *,” vol. 1, pp. 244–269, 2013.

[2] G. E. P. Box and M. E. Muller, “A note on the generation of random normal deviates,” *The annals of mathematical statistics*, no. 29, pp. 610–611, 1958.

[3] P. Eifinger, “Validierung eines Gradient Enhanced Kriging Verfahrens anhand einer Parameterstudie am Profilschnitt eines Fans,” *Projektarbeit*, 2013.

[4] A. I. J. Forrester and A. J. Keane, “Recent advances in surrogate-based optimization,” *Progress in Aerospace Sciences*, vol. 45, no. 1, pp. 50–79, 2009.

[5] M. Gibbs and D. J. C. MacKay, “Efficient implementation of Gaussian processes,” 1997.

[6] P. E. Gill, W. Murray, and M. H. Wright, “Practical optimization,” 1981.

[7] P. Gill, “Numerical linear algebra and optimization,” 2007.

[8] A. I. J. Forrester, A. J. Keane, and N. W. Bressloff, “Design and Analysis of "Noisy" Computer Experiments,” *AIAA Journal*, vol. 44, pp. 2331–2339, oct 2006.

[9] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient Global Optimization of Expensive Black-Box Functions,” *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998.

[10] A. J. Keane, “Statistical improvement criteria for use in multiobjective design optimization,” *AIAA journal*, vol. 44, no. 4, pp. 879–891, 2006.

[11] D. R. Jones, “A Taxonomy of Global Optimization Methods Based on Response Surfaces,” *Journal of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[12] R. Jin, W. Chen, and T. Simpson, “Comparative studies of metamodelling techniques under multiple modelling criteria,” *Structural and Multidisciplinary Optimization*, vol. 23, pp. 1–13, dec 2001.

[13] B. p.l.c., “BP Statistical Review ofWorld Energy June 2016,” tech. rep., 2016.

[14] I. Rechenberg, “Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. frommann-holzbog, Stuttgart, 1973,” *Step-Size Adaptation Based on Non-Local Use of Selection Information. In Parallel Problem Solving from Nature (PPSN3)*, 1994.

[15] C. Voß, *Automatische Optimierung von Verdichterschaufeln*. Abschlussbericht zum AG-Turbo COOREFF-T Teilvorhaben 1.1.1 des Verbundprojektes CO2-Reduktion durch Effizienz, 2008.

[16] C. Voß, “Multi-objective automated compressor optimization using a combines CFD-FEM process chain,” *Eccomas*, 2010.

[17] D. J. C. Mackkay, *Bayesian Methods for Adaptive Models*. PhD thesis, Citeseer, 1991.

[18] M. Aulich and U. Siller, “High-Dimensional Constrained Multiobjective Optimization of a Fan Stage,” in *ASME GT2011-45618*, 2011.

[19] D. J. J. Toal and A. J. Keane, “Efficient multipoint aerodynamic design optimization via cokriging,” *Journal of Aircraft*, vol. 48, no. 5, pp. 1685–1695, 2011.

[20] D. Huang, T. T. Allen, W. I. Notz, and R. A. Miller, “Sequential kriging optimization using multiple-fidelity evaluations,” *Structural and Multidisciplinary Optimization*, vol. 32, no. 5, pp. 369–382, 2006.

[21] J. Maurer, “Boost.Random.”

[22] D. G. Krige, “A statistical approach to some basic mine valuation problems on the witwatersrand,” 1953.

[23] G. Matheron, “Principles of geostatistics,” *Economic geology*, vol. 58, no. 8, pp. 1246–1266, 1963.

[24] D. Shepard, “A two-dimensional interpolation function for irregularly-spaced data,” in *Proceedings of the 1968 23rd ACM national conference*, pp. 517–524, ACM, 1968.

[25] R. L. Plackett, “Some theorems in least squares,” *Biometrika*, vol. 37, no. 1/2, pp. 149–157, 1950.

[26] F. Krüger, *Entwicklung von parallelisierbaren Gradienten-basierten Verfahren zur automatisierten, Ersatzmodell-gest{ü}tzten Optimierung unter Nebenbedinungen f{ü}r CFD-FEM-Verdichterdesign*. PhD thesis, 2012.

[27] S. G. Z.-H. Han, R. Zimmermann, “A New Cokriging Method for Variable-Fidelity Surrogate Modeling of Aerodynamic Data,” *48th AIAA Aerospace Sciences Meeting*, no. January, pp. 1–22, 2010.

[28] Z. Han, R. Zimmerman, and S. Görtz, “Alternative Cokriging Method for Variable-Fidelity Surrogate Modeling,” *AIAA journal*, vol. 50, no. 5, pp. 1205–1210, 2012.

[29] M. C. Kennedy and A. O’Hagan, “Predicting the output from a complex computer code when fast approximations are available,” *Biometrika*, vol. 87, pp. 1–13, 2000.

[30] I. N. Bronstejn and K. A. Semendjajew, *Taschenbuch der Mathematik*. Harri Deutsch, 2008.

[31] H. Thornburg, “Autoregressive Modeling: Elementary Least-Squares Methods,” *Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University Stanford, California 94305*, 2006.

[32] A. I. J. Forrester, A. Sóbester, and A. J. Keane, “Multi-fidelity optimization via surrogate modelling,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, vol. 463, no. 2088, pp. 3251–3269, 2007.

[33] R. Zimmermann and Z.-h. Han, “Simplified cross-correlation estimation for Cokriging predictors as multi-fidelity surrogate models,” no. 1984, pp. 1–24, 2009.

[34] P. Steimann, M. Frenkel, and D. Keller, “Moderne Programmiertechniken und Methoden,” 2012.

[35] A. Fog, “Software optimization resources,” 2013.

[36] K.A., “Processors - Define SSE2,SSE3 and SSE4,” 2011.

[37] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The RPROP algorithm,” in *Neural Networks, 1993., IEEE International Conference on*, pp. 586–591, IEEE, 1993.

[38] H. Helbig and A. Scherer, “Neuronale Netze,” *Vorlesungsskript*, 2011.

[39] G. Douglas, “Boost Function,” 2004.

[40] H. Avron and S. Toledo, “Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix,” *Journal of the ACM*, vol. 58, pp. 1–34, apr 2011.

[41] M. F. Hutchinson, “A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines,” *Communications in Statistics-Simulation and Computation*, vol. 18, no. 3, pp. 1059–1076, 1989.

[42] D. Toal, A. Forrester, N. Bressloff, A. Keane, and C. Holden, “An adjoint for likelihood maximization,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 465, no. 2111, pp. 3267–3287, 2009.

[43] C. A. Mader, J. R. R. A. Martins, J. J. Alonso, and E. Van Der Weide, “ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers,” 2008.

[44] A. Griewank and A. Walther, *Evaluating derivatives : principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics, 2008.

[45] D. J. Toal, N. Bressloff, A. Keane, and C. Holden, “The development of a hybridized particle swarm for kriginghyperparameter tuning,” 2011.

[46] S. P. Smith, “Differentiation of the Cholesky Algorithm,” *Journal of Computational and Graphical Statistics*, 1995.

[47] I. Murray, “Differentiation of the Cholesky decomposition,” 2016.

[48] S. Särkkä, *BAYESIAN FILTERING AND SMOOTHING*. Cambridge University Press, 2013.

[49] M. B. Giles, “Collected matrix derivative results for forward and reverse mode AD,” *Lecture Notes in Computational Science and Engineering*, vol. 64, pp. 34–44, 2008.

[50] S. Cook, *CUDA Programming*. Morgan Kaufmann, 2012.

[51] A. Dworak, F. Ehm, W. Sliwinski, and M. Sobczak, “MIDDLEWARE TRENDS AND MARKET LEADERS 2011,” *CERN*, 2011.