

DIPLOMARBEIT

Exploration mehrdimensionaler Simulationsergebnisse für faseroptische Nachrichtennetze

Fakultät für Informatik

Eingereicht von
Nicolas Luck

Datum:
23. Juni 2008

Gutachter:
Prof. Dr. Heinrich Müller
Lehrstuhl für graphische Systeme,
Fakultät für Informatik
Dr.-Ing. Dipl.-Wirt. Ing. Stephan Pachnicke MSc.
Lehrstuhl für Hochfrequenztechnik,
Fakultät für Elektrotechnik und Informationstechnik

Inhaltsverzeichnis

I. Problembeschreibung	7
1. Einleitung	8
1.1. Motivation und Zielsetzung	8
1.2. Verwandte Arbeiten	9
1.3. Aufbau der Arbeit	9
2. Simulation faseroptischer Nachrichtennetze	11
2.1. Grundlagen	11
2.2. Modulation und Signalgüte	12
2.3. Signaldegradation	14
2.3.1. Lineare optische Effekte	15
2.3.2. Nichtlineare optische Effekte	17
2.4. Simulationstool PHOToss	21
2.4.1. Parametervariation	22
2.4.2. Architektur	23
II. Grundlagen	24
3. Optimierungsverfahren	25
3.1. Lokale Suche	25
3.2. Simulated Annealing	25
4. Versuchsplan-Erstellung	28
4.1. Latinhypercube-Design	28
4.2. Maximin-Latinhypercube-Design	30
5. Funktionsschätzer	32
5.1. Ausgleichsebene	32
5.2. Shepard-Interpolant	34
5.3. Hardy-Multiquadrik	35
6. Visualisierung multivariater Daten	38
6.1. Multivariate Datensätze	38
6.2. Multivariate Funktionen	38
6.3. Gewähltes Verfahren	39

7. Verteilte Berechnungen	40
7.1. Condor Grid-Computing Software	40
 III. Lösung	 44
8. Meta-Modell-unterstütztes Optimierungsverfahren	45
8.1. Problemstellung	45
8.2. Lösung	46
9. Entwickelte Software	49
9.1. Anwendungsfälle	49
9.2. Dateiformate	51
9.3. Visualisierung und Navigation	54
9.4. Anwendung der grundlegenden Verfahren	57
9.5. Condor- und PHOTOS-Anbindung	60
9.6. Optimierungsmodus	64
10. Implementierung	68
10.1. Benutzte Werkzeuge	68
10.2. Softwaredesign	69
 IV. Ergebnisse	 75
11. Evaluationsdaten	76
11.1. Evaluationsdatensätze	76
11.2. Nichtangegebene Parameter	82
12. Evaluation der Funktionsschätzer	84
12.1. Schätzfehler	84
12.2. Optimierung auf Schätzer	91
13. Evaluation des Optimierungsverfahrens	93
13.1. Analyse der Parameter	93
13.2. Betrachtung des Parameters <i>fac</i>	95
14. Zusammenfassung und Ausblick	102
 V. Anhang	 104
Abbildungsverzeichnis	105
Literaturverzeichnis	107

Danksagung

Zunächst möchte ich mich bei meinen Betreuern Prof. Dr. Heinrich Müller und Dr.-Ing. Dipl.-Wirt. Ing. Stephan Pachnicke MSc. für ihre Unterstützung bedanken. Beide haben mit ihren Hinweisen und der Zeit, die sie sich für Diskussionen genommen haben, maßgeblich an dieser Arbeit mitgewirkt.

Weiterer Dank geht an die Mitarbeiter des Lehrstuhls für Hochfrequenztechnik. Dipl. Ing. Martin Windmann Msc., Dipl. Phys. Matthias Westhäuser, Dipl. Phys. Christian Remmersmann und Dipl. Phys. Sinan Özdür haben im Laufe dieser Arbeit immer wieder zwischendurch ein paar Minuten ihrer Zeit gewidmet, um all die Fragen zu beantworten, die ein Informatikstudent zu dem Themengebiet der Nachrichtentechnik sowie den dafür zugrunde liegenden physikalischen Effekten haben kann. Dr. Ing. Ansgar Steinkamp danke ich für die Einführungsvorlesung in das Gebiet der optischen Datenübertragung. Allen weiteren Mitarbeitern des Lehrstuhls danke ich für eine sehr angenehme Atmosphäre.

Außerdem danke ich meinem ehemaligen Kommilitonen Dipl. Inf. Tom Paschenda für fruchtbare Gespräche und den ein oder anderen gedanklichen Anstoß.

Kurzfassung

Um die Performanz von faseroptischen Nachrichtennetzen zu optimieren, gilt es, gewisse Parameter des Netzes so zu wählen, dass die Übertragung optimal, d.h. mit möglichst wenigen Bitfehlern bei möglichst großer Datenrate bzw. bei einer möglichst hohen Reichweite erfolgt. Um einen gegebenen Parametersatz zu bewerten, können Computersimulationen anstelle von realen Systemen verwendet werden. In Abhängigkeit von mehreren Faktoren kann die Laufzeit einer solchen Simulation auf aktueller Hardware jedoch mehrere Stunden pro Parametersatz betragen, so dass es nicht mehr praktikabel ist, eine große Menge von Parametersätzen direkt auszuwerten.

In dieser Arbeit werden ein Meta-Modell basiertes Optimierungsverfahren und seine Implementierung vorgestellt, womit trotz der hohen Rechenkomplexität möglichst schnell eine optimale Parameterkombination gefunden werden kann. Basierend auf raumfüllenden Versuchsplänen, wie dem Latinhypercube-Design, werden auszuwertende Punkte des Parameterraums ausgewählt. Nach der Auswertung dieser wenigen Parametersätze, dienen die Ergebnisse als Stützstellen für Funktionsschätzer. Auf diesen schnell zu berechnenden, geschätzten Funktionen können nun bekannte Optimierungsverfahren wie z.B. Simulated Annealing angewendet werden. Regionen des Schätzers, die interessant erscheinen, können ausgewählt und mit weiteren Simulationen genauer untersucht werden. Die dabei erhaltenen Ergebnisse können nun wiederum genutzt werden, um den Schätzer zu präzisieren. So ergibt sich ein iteratives, Meta-Modell basiertes Optimierungsverfahren, das in Abhängigkeit der bekannten Daten, die als nächstes zu berechnenden Parametersätze auswählt und so automatisiert nur die essentiellen Berechnungen durchführt.

Neben der reinen Lokalisierung eines Optimums, soll die interessante Region auch auf eine einfach verständliche Weise visualisiert werden. Die im Zuge dieser Arbeit entstandene Software ermöglicht es dem Benutzer, durch den (je nach Benutzereingabe) mehrdimensionalen Parameterraum zu navigieren und zeigt dabei immer eine zweidimensionale Schnittebene der gefundenen Optima, der berechneten Daten und der Funktionsschätzer an.

Teil I.

Problembeschreibung

1. Einleitung

In diesem Kapitel wird eine Einführung und ein Überblick über die gesamte Arbeit gegeben. Abschnitt 1.1 erklärt die Hintergründe und motiviert das vorgenommene Vorgehen. Arbeiten mit einem ähnlichen Inhalt werden in 1.2 vorgestellt. 1.3 beschreibt den Aufbau der gesamten Arbeit.

1.1. Motivation und Zielsetzung

Übertragung von Telefonie- und Fernseh-Daten, verteilte Anwendungen wie Google Maps und YouTube, steigende Zahlen der Internetnutzer und sehr viele interessante bandbreitenintensive (Internet-) Anwendungen erfordern immer größere Datenübertragungsraten. Wo hingegen nahe am Nutzer die Bandbreite im Vergleich klein ausfällt und meist Kupferkabel als Übertragungsmedium dienen, bestehen Weitverkehrsnetze hauptsächlich aus Glasfaserverbindungen, die Daten optisch und mit Bandbreiten bis zu $80 \cdot 40 \text{ Gb/s}^1$ (80 Kanäle mit 40 Gb/s) übertragen. Der zu optimierende Wert einer Verbindung ist das Produkt aus Bandbreite und Entfernung. Es ist nur in Ausnahmefällen von Nutzen, eine große Bandbreite für eine sehr kurze Strecke zu erreichen, oder umgekehrt.

Um möglichst hohe Datenraten über möglichst weite Strecken hinweg zu erzielen, müssen viele Systemparameter geschickt gewählt werden. Teilweise können Parameterkombinationen analytisch berechnet und ausgeschlossen werden. Da es aber schwierig ist, alle physikalischen Effekte dabei mit einzubeziehen, müssen Systeme mit gegebenen Parametern getestet werden. Dazu können Testsysteme im Labor aufgebaut und betrieben werden. Dies ist aber nicht immer nötig. Es gibt zusätzlich die Möglichkeit, Systeme durch Software zu simulieren, was günstiger und schneller ist. Um jedoch aus einem großen Parameterraum sehr viele Kombinationen automatisiert zu simulieren (und dadurch zu bewerten), nimmt auch die Simulation, je nachdem welche physikalischen Effekte dabei mit einbezogen werden, zu viel Zeit in Anspruch.²

Nichtsdestotrotz werden Untersuchungen durchgeführt, indem optische Übertragungssysteme mit sehr vielen unterschiedlichen Parametern simuliert werden. Da die verschiedenen Simulationsläufe nicht voneinander abhängen, kann durch Parallelisierung die dafür erforderliche Zeit verkürzt werden. Da aber neben der Zeit auch die Rechenleistung bzw. die Anzahl der verfügbaren Rechner eine begrenzte Ressource ist, müssen Kompromisse gemacht werden. So ist es nicht unüblich, dass für manche Untersuchungen ein Rechner-Pool bestehend aus einer zweistelligen Anzahl an Rechnern für Wochen bis Monate belegt ist. Meist wird der Parameterraum dabei in einem vorgegebenen Bereich äquidistant abgetastet. Es werden im Vorhinein Überlegungen angestellt, wo dieser Bereich zu liegen hat und wie groß die Schrittweite sein soll.

¹ Dieser Wert gilt für aktuelle kommerziell verfügbare Netze. In Laborbedingungen werden auch höhere Übertragungsraten erreicht.

² Zur Veranschaulichung: die in 11.1 beschriebene Simulation (10x10Gb/s WDM-Verbindung mit 10 Spans, Vor-, Inline- und Nachkompensation der Dispersion und einer Augenanalyse unter Berücksichtigung nichtlinearer Effekte) benötigt auf einem Intel Core 2 Quad Q6600 mit 2GiB Ram ca. 30 Minuten.

Zunächst wäre es wünschenswert, diesen Aufwand zu reduzieren. Bei der Suche nach einem Optimum können evtl. nach der Auswertung weniger Punkte bereits Regionen ausgeschlossen werden. Man könnte von einem Optimierungsalgorithmus, wie z.B. Simulated Annealing [KGV83], die im Weiteren auszuwertenden Stellen des Parameterraums bestimmen lassen. Es ist jedenfalls in dem geschickten Auslassen bestimmter Stellen des Parameterraums Optimierungspotential zu vermuten. Außerdem ist die Visualisierung und das intuitive Erkunden des wahrscheinlich mehrdimensionalen Raumes ein weiteres wünschenswertes Ziel. Bestehende multivariate Daten zu visualisieren, ist ein Thema für sich. In diesem Fall kommen die Kosten für die Auswertung eines Punktes des mehrdimensionalen Raumes hinzu.

Ziel dieser Arbeit ist es, ein Meta-Modell [Bla75] basiertes Optimierungsverfahren zu entwickeln, welches iterativ auszuwertende Punkte auswählt, mit denen das Modell aktualisiert wird, um mit dem auf diese Weise präzisierten Modell die nächsten auszuwertenden Punkte bestimmen zu können. Die Hoffnung ist, mit einer relativ kleinen Anzahl von tatsächlichen Auswertungen das Meta-Modell soweit an die zugrunde liegende Funktion anzupassen, dass Modell und Funktion bzgl. des Optimums kongruent sind. Dieses Verfahren soll implementiert und evaluiert werden. Außerdem soll ein möglichst intuitives Benutzerinterface entwickelt werden, mit dem die erzeugten multivariaten Daten erkundet werden können. Dabei liegt der Schwerpunkt auf der Anforderung, es dem Benutzer möglichst einfach zu machen, das gefundene Optimum zu lokalisieren und die nahe Umgebung des Optimums zu untersuchen.

1.2. Verwandte Arbeiten

Meta-Modelle wurden das erste Mal in [Bla75] erwähnt. Es gibt mittlerweile viele Meta-Modell basierte Optimierungsanwendungen für Fälle, in denen die Kosten einer Funktionsauswertung zu hoch sind, um die benötigte Menge an Auswertungen vorzunehmen.

In [Hen07] wird ein iteratives Vorgehen zur Optimierung des CNC³-gesteuerten Drückprozesses beschrieben. Wie auch das in dieser Arbeit vorgestellte Verfahren, verwendet es raumfüllende Versuchspläne und ein Meta-Modell, um eine Funktion zu optimieren deren Auswertung kostspielig ist.

In [KG05] wird die Form einer Tragfläche mit Hilfe eines evolutionären Algorithmus optimiert. Da dieser pro Generation viele Individuen produziert und eine Auswertung eines Individuums kostspielig ist, werden mit Hilfe eines neuronalen Netzwerkes die Individuen bewertet, um viele schlechte auszusortieren und nur noch wenige gute tatsächlich auszuwerten.

Die Optimierung von Puffern einer Fertigungsstraße wird in [PAGN07] beschrieben. Auch hier wird ein neuronales Netz als Meta-Modell verwendet, um einen evolutionären Algorithmus einsetzen zu können.

In [JE04] wird ebenfalls ein evolutionärer Algorithmus jedoch kein neuronales Netzwerk, sondern Kriging [Kri51] als Meta-Modell verwendet.

1.3. Aufbau der Arbeit

Diese Arbeit ist unterteilt in fünf Teile. In Teil I (Problembeschreibung) wird die in dieser Arbeit gelöste Aufgabe näher beschrieben. Es wird außerdem auf die dafür nötigen Grundlagen der

³ Wikipedia: „Computerized Numerical Control: eine elektronische Methode zur Steuerung und Regelung von Werkzeugmaschinen.“

optischen Datenübertragung eingegangen.

Teil II (Grundlagen) stellt bereits bestehende Verfahren vor, auf denen sich diese Arbeit stützt bzw. die in dem vorgestellten Lösungsverfahren als Teile wiederverwendet werden. Dies sind im wesentlichen das Latinhypercube-Design sowie die Hardy Multiquadrik Methode und Simulated Annealing, aber auch Alternativen zu diesen.

In Teil III (Lösung) wird das entwickelte Verfahren zur Lösung des in Teil I geschilderten Problems dargestellt. Dies erfolgt entlang der entwickelten Software und ihrer Merkmale. Es wird eine Übersicht über ihre Funktionalität gegeben und das entwickelte, iterative Optimierungsverfahren im Detail vorgestellt. Außerdem wird auf das Softwaredesign und Implementierungsdetails eingegangen.

Teil IV (Ergebnisse) enthält statistische Untersuchungen aller relevanten Verfahren. Es wird untersucht, wie sich die bekannten Verfahren in diesem Kontext verhalten und inwiefern das neu entwickelte Verfahren sich zur Lösung des gestellten Problems eignet.

2. Simulation faseroptischer Nachrichtennetze

In diesem Kapitel werden die für diese Arbeit relevanten Gegebenheiten der optischen Übertragungstechnik angesprochen. Abschnitt 2.1 führt in das Thema ein und zeigt jene Bereiche auf, die gegenwärtig von hohem wissenschaftlichen Interesse sind. In Abschnitt 2.2 werden Modulationsformate und die *Eye Opening Penalty* angesprochen. Abschnitt 2.3 geht etwas weiter in die Tiefe und stellt die optischen Effekte vor, die beachtet werden müssen, um die Performanz optischer Nachrichtennetze optimieren zu können. In Abschnitt 2.4 wird das Simulationsprogramm PHOTOSS vorgestellt, das in dieser Arbeit zur Simulation optischer Übertragungsstrecken benutzt wurde.

2.1. Grundlagen

Faseroptische Nachrichtennetze können bezüglich ihrer Architektur in drei Kategorien unterteilt werden: Punkt-zu-Punkt Verbindungen, Metronetze und lokale Netze. Im Weiteren wird nur auf die Punkt-zu-Punkt Verbindungen eingegangen. Die Länge der Übertragungsstrecke variiert bei diesen Systemen zwischen hunderten von Metern (short haul) bis zu mehreren tausend Kilometern (long haul). Das Licht wird bei der Übertragung durch die Faser gedämpft. Da das übertragene Signal dadurch vom Empfänger evtl. nicht mehr rekonstruierbar ist, muss ab einer gewissen Übertragungslänge¹ dieser Effekt kompensiert werden. Vor 1990 war der einzig verfügbare Ansatz, das Signal mit Hilfe von 3R-Regeneratoren neu zu erzeugen. Dabei wird das Signal empfangen, interpretiert und als elektrischer Bitstrom repräsentiert, so dass daraus erneut ein optisches Signal generiert werden kann. Durch die Einführung von optischen Verstärkern gibt es seit etwa 1990² eine Alternative zu 3R-Regeneratoren. Optische Verstärker wirken der Dämpfung entgegen, ohne das Signal neu zu erzeugen, fügen dafür aber Rauschen hinzu. Optische Verstärker sind kostengünstiger als 3R-Regeneratoren und ermöglichen die Übertragung optischer Signale über 3000 km ohne Einsatz eines 3R-Regenerators. Da sie mehrere Kanäle zur gleichen Zeit verstärken, ermöglichen sie einen kostengünstigen Einsatz von wavelength-division multiplexing (WDM), also der Übertragung mehrerer Signale auf unterschiedlichen Wellenlängen, jedoch parallel innerhalb einer Faser.

Die Möglichkeit, mit Hilfe von optischen Verstärkern die Entfernung zwischen 3R-Regeneratoren und mit Hilfe von WDM die (Gesamt-) Bitrate zu erhöhen, revolutionierte die optische Übertragungstechnik und führte zu Beginn der 1990'er zur Einführung der vierten Generation von optischen Übertragungssystemen. Diese werden heute weiterentwickelt, indem einerseits die Anzahl der parallel übertragenen Kanäle und andererseits die Bitrate pro Kanal erhöht wird. Da jedoch das optische Signal von einem optischen Verstärker nicht neu aufgebaut wird, es also kein *reshaping* und *retiming* gibt, akkumulieren die Auswirkungen optischer Effekte auf das Signal über die gesamte Länge der (regeneratorfreien) Übertragungsstrecke. Dies kann dazu führen, dass das Signal nicht mehr fehlerfrei von dem Empfänger erkannt wird bzw. die Bitrate oder die Anzahl oder

¹i.A. bei 80 - 120 km [Agr02]

²Entwickelt in 1985, seit 1990 kommerziell verfügbar

Verteilung der Kanäle geändert werden muss. Diese optischen Effekte sind also die limitierenden Faktoren, die kompensiert werden müssen, um die Kapazität eines Systems zu steigern. Deshalb wird darauf im Folgenden näher eingegangen.

2.2. Modulation und Signalgüte

Das in einer optischen Faser geführte elektrische Feld hat drei Eigenschaften, die zur Übertragung von Informationen verwendet werden können: die *Intensität*, die *Phase* und die *Polarisation*. Häufig wird das Modulationsformat *On-Off-Keying* (OOK) verwendet, welches eine logische Eins mit dem Vorhandensein eines Feldes und eine Null mit der Abwesenheit des Feldes innerhalb eines Zeitfensters repräsentiert. Es kodiert die Information also in der Intensität; Phase und Polarisation werden von dem Empfänger nicht berücksichtigt. *Binary Differential Phase Shift Keying* (DPSK) kodiert Information in der Phase, indem eine logische Eins mit einer π -Phasendrehung und eine Null mit der Konstanz der Phase repräsentiert wird. Beide Modulationsformate gibt es in den Varianten *Return-to-Zero* (RZ) und *Non-Return-to-Zero* (NRZ). In der RZ-Variante kehrt die Intensität innerhalb jedes Bit-Zeitfensters zu Null zurück, während bei NRZ dies nicht der Fall ist. Abbildung 2.1 zeigt die vier Kombinationsmöglichkeiten. Als weiteres Modulationsformat sei das *Duobinär*-

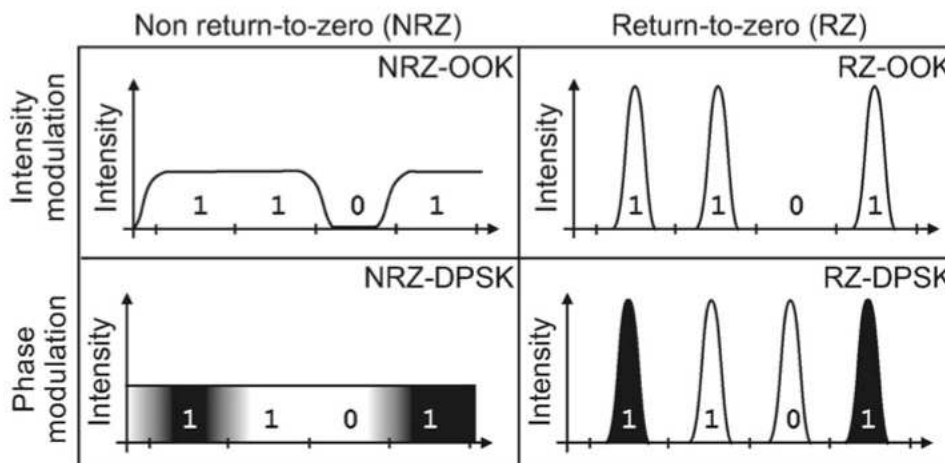


Abbildung 2.1: Verlauf der Intensität und Phase für die vier Modulationsformate NRZ-OOK, RZ-OOK, NRZ-DPSK, RZ-DPSK. Die Phase ist durch den Grauwert dargestellt. [WE06]

Verfahren erwähnt. Dieses verhält sich ähnlich zum NRZ-OOK-Format, mit dem Unterschied, dass drei logische Intensitätsniveaus des E-Feldes verwendet werden (siehe Abbildung 2.2). Zu den Niveaus Null (keine Leistung) und Eins (Leistung) wird Minus Eins hinzugefügt. Dieses wird durch eine π -Phasendrehung der Trägerwelle erreicht. Da der Empfänger nur die Leistung ($|E|^2$) detektiert, werden sowohl die Symbole Eins als auch Minus Eins als eine logische Eins interpretiert. Der Vorteil des Duobinär-Verfahrens liegt darin, dass durch die Phasenmodulation der Eins-Bits, das Signal eine geringere spektrale Breite hat. Dies hat den positiven Effekt, dass es beständiger gegenüber Dispersion (s.u.) ist. Es existieren eine Reihe weiterer Modulationsformate, auf die hier nicht weiter eingegangen wird; [WE06] gibt einen Überblick.

Zur Beurteilung der Güte eines Übertragungssystems bietet sich das so genannte *Augendia-*
gramm an. Dieses erhält man durch die Oszillografierung der Intensität des elektrischen Feldes
am Empfänger mit einem Vielfachen der Taktzeit als Ablenkzeit des Oszilloskops (siehe [Lük75]).
Dadurch werden die Intensitätsverläufe aller übertragenen Bits übereinander gelegt. Siehe Abbil-
dung 2.3. Für ein intensitätsmoduliertes Signal gibt die Augenöffnung ein Maß dafür an, wie groß
der Abstand der den Binärwerten Null und Eins zugeordneten Abtastwerten der Intensität ist. Die
Augenbreite gibt an, inwiefern von den genauen Abtastzeitpunkten abgewichen werden darf.

Als vergleichbares Maß, wird die *Eye Opening Penalty (EOP)*³ verwendet. Zur Berechnung der
EOP wird zunächst das höchste Rechteck innerhalb des Augendiagramms gesucht, das eine Breite
von 20% des Bit-Zeitfensters hat (siehe Abbildung 2.4). Für die bekannte Höhe des Rechtecks P_R

³auch Eye Closure Penalty genannt

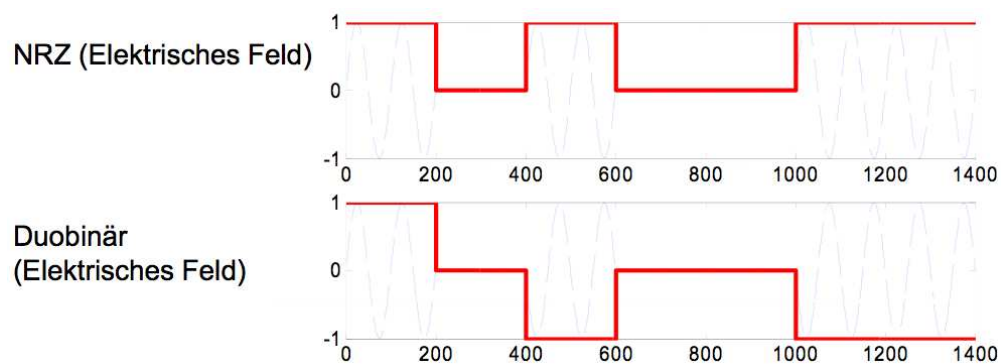


Abbildung 2.2: Prinzip des Duobinär-Verfahrens. [Pac]

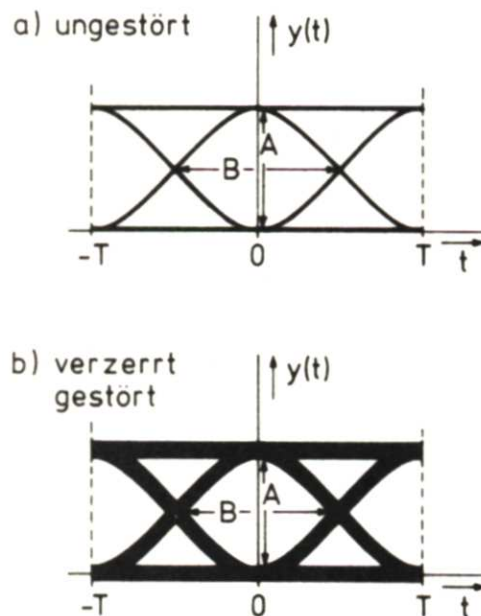


Abbildung 2.3: Darstellung eines Augendiagramms. A bezeichnet die *Augenöffnung*, B die *Augen-*
breite [Lük75]

ist die EOP gegeben durch [KL02]

$$EOP = -10 \log \frac{P_R}{2P_{ave}}, \quad (2.1)$$

wobei P_{ave} die mittlere Signalleistung ist. Negative EOP-Werte repräsentieren ein Auge, das weiter geöffnet ist als das unverzerrte Auge, wohingegen positive Werte ein Auge repräsentieren, das weiter geschlossen ist, als die Referenz. Degradation des Signals führt in der Regel zu einer Erhöhung des EOP-Wertes. Bei der Optimierung eines Übertragungssystems ist es folglich meist das Ziel, die EOP zu minimieren.

2.3. Signaldegradation

Optische Effekte, die zur Signaldegradation beitragen, können in die Kategorien *linear* und *nicht-linear* unterteilt werden. Diese Adjektive beziehen sich auf die Abhängigkeit der elektrischen Polarisation eines Materials (wie z.B. Glas) von dem umgebenden elektrischen Feld. In der nichtlinearen Optik werden Effekte betrachtet, die sich dadurch ergeben, dass der Brechungsindex eines Mediums von der Intensität des elektrischen Feldes, also dem Betragsquadrat des E-Feldes $|\vec{E}|^2$ abhängt. Jedes Medium verhält sich in diesem Sinne nichtlinear, kann aber als Näherung bei kleinen Intensitäten als ein lineares optisches Medium (also eines, bei dem der Brechungsindex für eine Wellenlänge konstant ist) betrachtet werden. Dies gilt natürlich auch für Quarzglas, woraus die meisten optischen Fasern bestehen.

Im Folgenden wird auf lineare und nichtlineare optische Effekte gesondert eingegangen. Nicht-lineare optische Effekte können für geringe Lichtintensitäten vernachlässigt werden, werden aber umso relevanter, je größer die Leistung des in die Faser eingekoppelten Lichts ist. Die Effekte

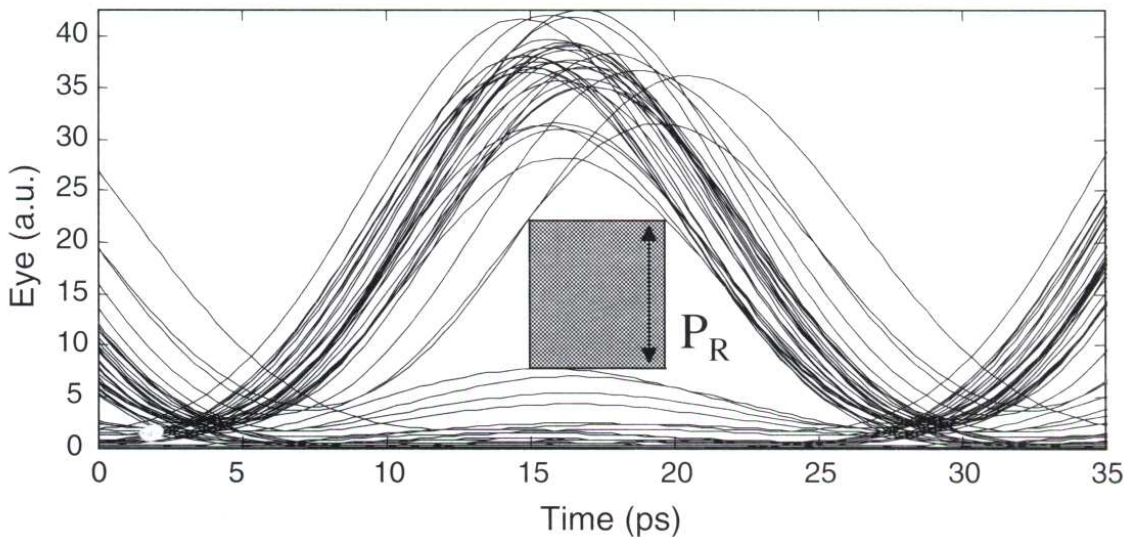


Abbildung 2.4: Augendiagramm mit eingezeichneter EOP-Box. [KL02]

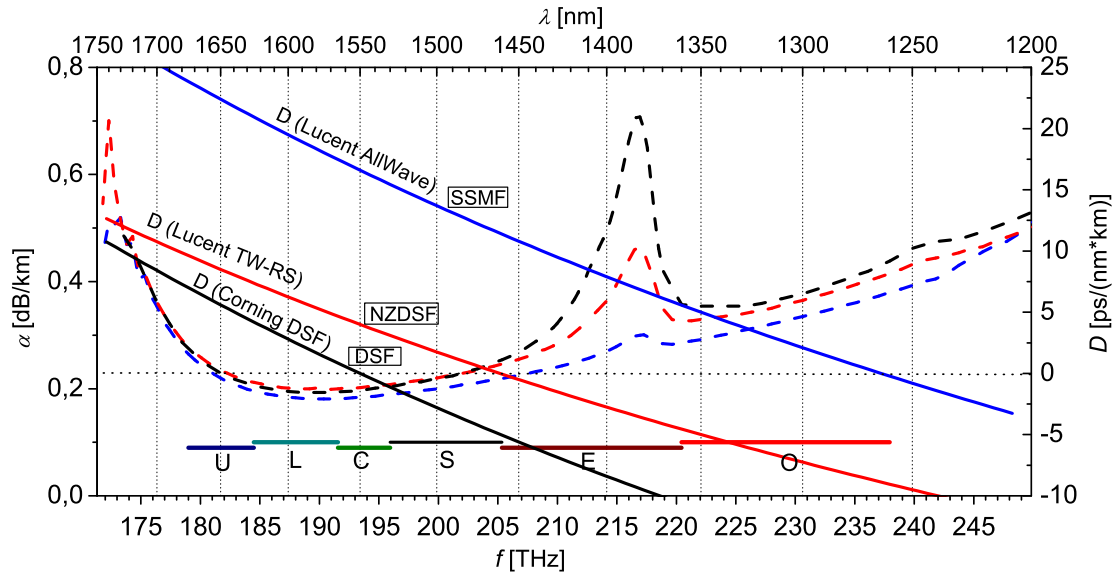


Abbildung 2.5: Dispersion und Dämpfung in Abhängigkeit der Frequenz. [Pac05]

werden kurz beschrieben, und ihre Bedeutung für den Betrieb optischer Nachrichtennetze wird dargelegt. Für eine detaillierte Behandlung siehe [Agr01] und [Agr02].

2.3.1. Lineare optische Effekte

Dämpfung

Von dem eingespeisten Licht wird nicht die gesamte Leistung durch die Faser übertragen. Durch Dämpfung geht ein Teil verloren. In Abhängigkeit des Dämpfungskoeffizienten α und der Länge L , beschreibt

$$P_{out} = P_{in} \cdot e^{-\alpha L} \quad (2.2)$$

den Zusammenhang zwischen Eingangs- und Ausgangsleistung.

Es gibt mehrere Effekte, die sich dämpfend auf unterschiedliche Spektralanteile des eingespeisten Lichts auswirken. Die *Materialabsorption* von Quarzglas erzeugt hohe Dämpfung im UV ($< 0,1\mu\text{m}$) und im IR ($> 9\mu\text{m}$) Bereich, die sich bis in den für die Datenübertragung relevanten Bereich um $1,55\mu\text{m}$ erstreckt (siehe Abbildung 2.5). Ausserdem relevant ist die OH^- Absorption um $1,37\mu\text{m}$. Mikroskopische Unebenheiten, verursacht durch herstellungsbedingte Dichtefluktuationen, führen zu geringen Brechzahlsschwankungen, die die Ursache für *Rayleigh-Streuung* darstellen. Auch makroskopische Unebenheiten, wie z.B. eine Krümmung der Faser, führen zu Leistungsverlusten.

Diese Verluste sind kritisch, da einerseits der Empfänger eine Mindestleistung erhalten muss, um überhaupt ein Signal zu erkennen und, wesentlich relevanter, da verschiedene Kompo-

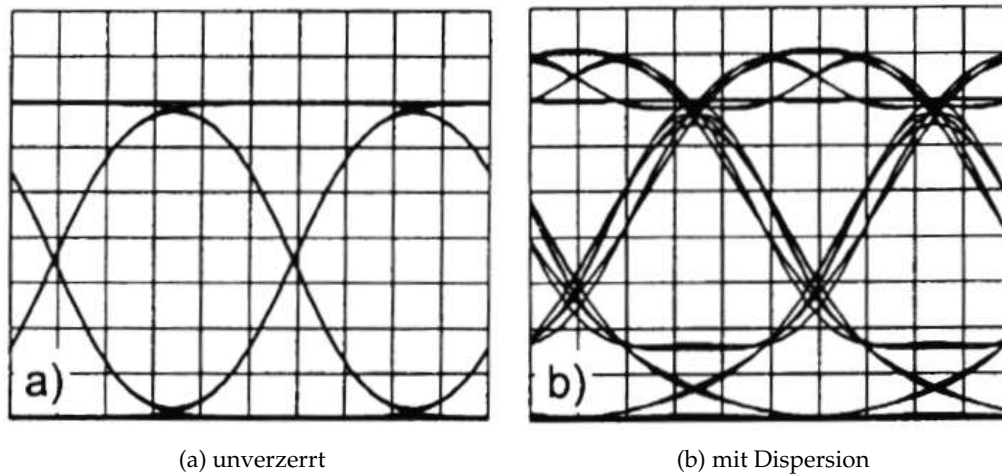


Abbildung 2.6: Auswirkung der Dispersion auf Pulsflanken eines NRZ-OOK modulierten Signals, dargestellt als Augendiagramm. [VP02]

nenten Rauschen einführen. Wird die Leistung des Signals so weit gedämpft, dass sie in der Größenordnung des Rauschen liegt, ist das Signal nicht vom Rauschen zu unterscheiden und eine Datenübertragung somit nicht mehr fehlerfrei möglich.

Der Dämpfung kann entgegengewirkt werden, indem mit möglichst hohen Leistungen gearbeitet wird. Da damit eine Verstärkung der nichtlinearen Effekte einhergeht, kann die Leistung nicht beliebig hoch gewählt werden. Aufgrund der Wellenlängenabhängigkeit der Dämpfung ist es sinnvoll, die Kanäle auf eine möglichst gering gedämpfte Wellenlänge zu legen. In der Regel werden die beiden, von der OH^- Absorption getrennten Fenster um $1,55\mu m$ und $1,3\mu m$ verwendet (siehe Abbildung 2.5). Wie bereits geschildert, bietet sich als Lösung dieses Problems der periodische Einsatz von optischen Verstärkern an.

Dispersion

In der optischen Nachrichtentechnik bezeichnet Dispersion Effekte, die zur Streuung der Signallaufzeit führen. Es gibt mehrere Ursachen für Dispersion. Gemein haben alle, dass verschiedene Anteile des eingespeisten Signals unterschiedliche Gruppengeschwindigkeiten haben. Bei *Modendispersion* bilden die verschiedenen Ausbreitungsgeschwindigkeiten der geführten Moden⁴ diese Teile. Zur Veranschaulichung: im Strahlenmodell des Lichts kann eine Mode mit einem Strahlenweg durch die Faser gleichgesetzt werden. Da verschiedene Strahlenwege eine unterschiedliche Länge haben, ist es intuitiv klar, dass ein Teil des Lichts früher und ein anderer später am Empfänger ankommt. Jedoch kann nur mit dem Wellenmodell des Lichts erklärt werden, dass Fasern konstruiert werden können, die für bestimmte Wellenlängen nur eine Mode (also einen Strahlenweg) zulassen. In solche Einmodenfasern findet Modendispersion nicht statt, so dass für weit reichende Punkt-zu-Punkt Verbindungen nur noch diese Faserart eingesetzt wird. Bei

⁴Wikipedia: „Als Moden (von engl. mode -s, dort vom lat. modus), auch Schwingungsmoden, in der Akustik überwiegend Raummoden genannt, bezeichnet man in der Physik die stationären Eigenschaften stehender Wellen (z. B. in einer Pfeife, einem Hohlraumresonator oder auf einer Saite) und auch fortlaufender Wellen (z. B. in einem Hohlleiter, Laserstrahl oder Glasfaserkabel) hinsichtlich ihrer longitudinalen oder transversalen Energieverteilung.“

Polarisationsmodendispersion sind doppelbrechende Anteile der Faser dafür verantwortlich, dass unterschiedlich polarisiertes Licht sich unterschiedlich schnell ausbreitet. Es ist nicht erwünscht, dass sich eine Faser doppelbrechend verhält. Dies kann jedoch durch Umwelteinflüsse, wie z.B. asymmetrischer Druck auf Teile der Faser, induziert werden. *Material-* und *Wellenleiterdispersion* entstehen dadurch, dass die Brechzahl eines Materials bzw. die Eindringtiefe einer Mode in den Fasermantel wellenlängenabhängig ist. Da aber ein modulierte Signal immer spektrale Breite hat, also verschiedene Wellenlängen enthält, die sich mit unterschiedlichen Geschwindigkeiten durch die Faser bewegen, erfährt dieses durch Material- und Wellenleiterdispersion eine Degradation. Pulse werden durch Dispersion verbreitert.

Dispersion ist störend, da als Modulationsverfahren meist *On-Off-Keying* eingesetzt wird. Werden durch Dispersion Pulse so weit verbreitert, dass sie in benachbarte Zeitfenster hineinragen, könnte eine Null vom Empfänger nicht mehr als solche identifiziert werden. Dies wird als *Intersymbolinterferenz* bezeichnet. Durch Verringerung der Bitrate wird das Zeitfenster eines Bits vergrößert, so dass dieser Effekt weniger Konsequenzen hat. Um jedoch die Bitrate erhöhen zu können, müssen die Auswirkungen der Dispersion minimiert werden. Dazu können besonders gefertigte Fasern eingesetzt werden. Materialdispersion wechselt im Infrarotbereich das Vorzeichen. Dies impliziert, dass jede Faser für eine bestimmte Wellenlänge eine Materialdispersion von Null hat. Dispersionsverschobene Fasern (DSF, dispersion shifted fibre) werden so konstruiert, dass das betragsmäßige Dispersionsminimum dem Dämpfungsminimum angenähert wird. Dispersionskompensierende Fasern (DCF, dispersion compensated fibre) weisen für den relevanten Frequenzbereich eine negative Dispersion auf, so dass in Kombination mit Standard-Einmodenfasern eingesetzt die akkumulierte Dispersion des gesamten Systems Null sein kann.

2.3.2. Nichtlineare optische Effekte

Verhält sich ein Material optisch nichtlinear, bedeutet dies, dass sich die Brechzahl des Materials in Abhängigkeit der Lichtintensität ändert. D.h. die Brechzahl n kann durch

$$n = n_0 + n_{NL}(I) \quad (2.3)$$

beschrieben werden, wobei I die Intensität des betrachteten Lichts ist. Dies führt zu den folgenden Effekten.

Selbstphasenmodulation

Bei der Selbstphasenmodulation erfährt ein optischer Puls durch die Ausbreitung in einer Faser eine Phasendrehung. Durch die Abhängigkeit des Brechungsindex von der Intensität, erfährt ein intensitätsmoduliertes Signal eine ständige Änderung der Phase. Diese ist gegeben durch [Agr02]

$$\Phi_{NL} = \gamma P_{in} L_{eff}, \quad (2.4)$$

wobei P_{in} die optische Leistung und L_{eff} durch

$$L_{eff} = \frac{1 - e^{-\alpha L}}{\alpha} \quad (2.5)$$

gegeben ist (mit der Faserlänge L und dem Dämpfungskoeffizienten α).

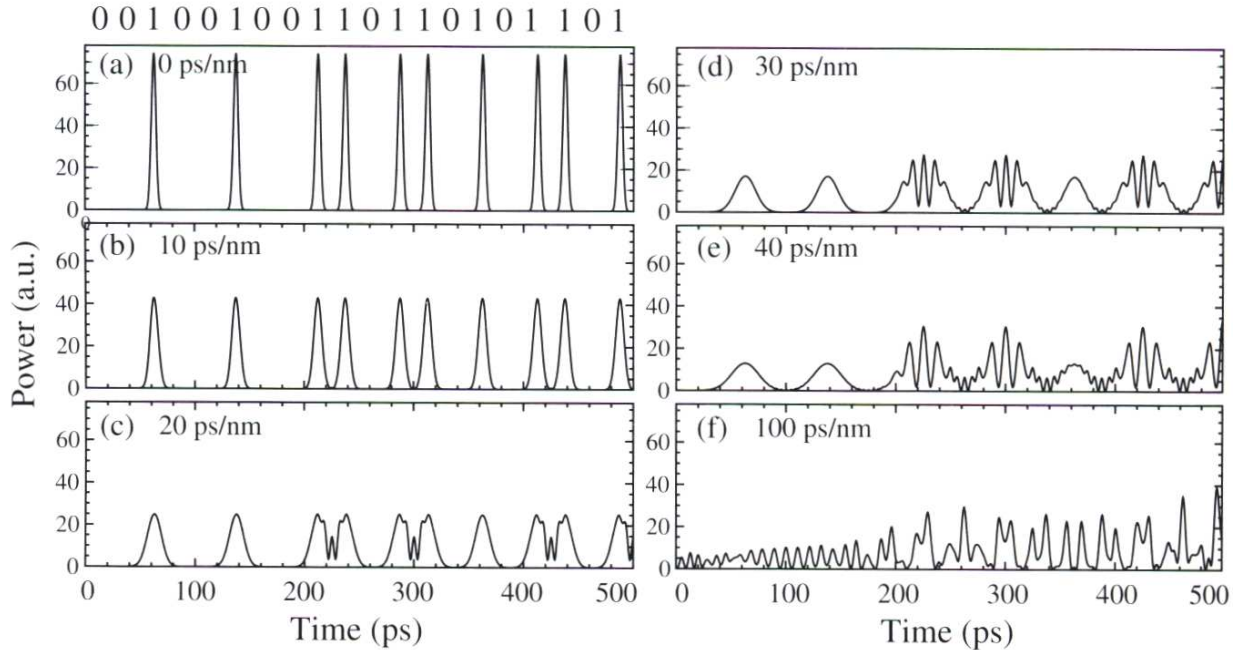


Abbildung 2.7: Auswirkung der Dispersion auf ein RZ-OOK modulierte Signal. Bei $10 \frac{\text{ps}}{\text{nm}}$ ist eine Verbreiterung der Pulse zu erkennen. Für höhere Dispersionswerte überlappen sich die Pulse zeitlich, bis sie nicht mehr zu unterscheiden sind. [KL02]

Da die Ursache dafür das Signal selbst ist, heißt der Effekt *Selbstphasenmodulation* (SPM). Da sich diese Phasendrehungen mit einer Intensitätsänderung des Signals über die Zeit ändern, resultieren sie in einer Änderung der Frequenz einzelner Signalteile verschiedener Intensität, so dass neue Frequenzen zu dem Spektrum des Signals hinzukommen. Steigende Impulsflanken erfahren eine Verschiebung zu niedrigen Frequenzen, fallende Impulsflanken zu höheren Frequenzen (siehe Abbildung 2.8). In Verbindung mit Dispersion führt dies zu einer Pulsverbreiterung, also zu Degradation des Signals.

Kreuzphasenmodulation

WDM-Systeme sind sensitiver gegenüber leistungsabhängigen Effekten, da nicht die Leistung eines einzelnen Kanals, sondern die Summe der Leistungen aller Kanäle, die zu einem gegebenen Zeitpunkt an einer Stelle der Faser wirken, die Intensität nichtlinearer Effekte bestimmt. D.h. dass der Betrag, der im letzten Abschnitt beschriebenen Phasendrehung, eines Kanals in einem WDM-System auch von den anderen Kanälen bzw. von deren genauen Leistungen zu einem Zeitpunkt und damit bei intensitätsmodulierten Signalen von deren übertragenen Bitmustern abhängt. D.h. die Phasendrehung des j -ten Kanals in einem Mehrkanalsystem ist gegeben durch [Agr02]

$$\Phi_{NL}^j = \gamma L_{eff} \left(P_j + 2 \sum_{m \neq j} P_m \right), \quad (2.6)$$

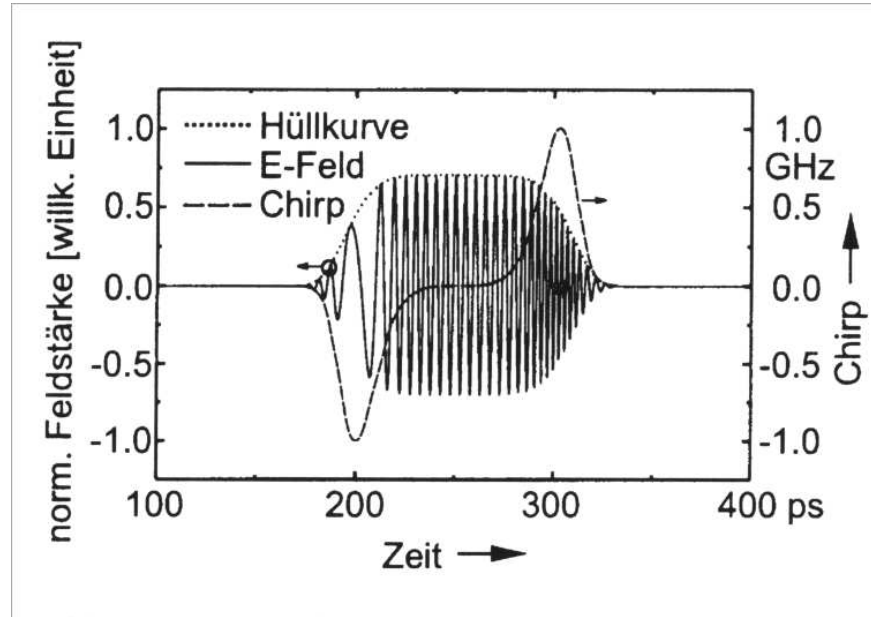


Abbildung 2.8: Einfluss von SPM auf ein intensitätsmoduliertes Signal. [VP02]

wobei über die Leistungen P_m der übrigen Kanäle summiert wird. Dies macht es schwer, die genaue Phasendrehung und damit die negativen Konsequenzen der Kreuzphasenmodulation abzuschätzen. Zu vermeiden ist die zeitliche Überlappung zweier Pulse unterschiedlicher Kanäle.

Wie bei SPM führt die durch die Phasendrehung induzierte Frequenzverschiebung in Verbindung mit Dispersion zu einer Amplitudenverzerrung und somit zur Degradation des Signals.

Dispersion führt jedoch weiterhin dazu, dass sich die Pulse unterschiedlicher Kanäle mit unterschiedlichen Geschwindigkeiten ausbreiten. Durch die Wahl der spektralen Platzierung der Kanäle, kann also Einfluss darauf genommen werden, wie groß der Geschwindigkeitsunterschied der Signale der verschiedenen Kanäle und damit deren potentielle Überlappung ist.

Vier-Wellen-Mischung

Die Leistungsabhängigkeit der Brechzahl führt in WDM-Systemen zu einem weiteren Effekt. Drei Wellen (Kanäle) mit unterschiedlichen Frequenzen ω_1 , ω_2 und ω_3 erzeugen eine weitere Welle (transferieren Leistung in einen weiteren Kanal) mit der Frequenz [Agr01]

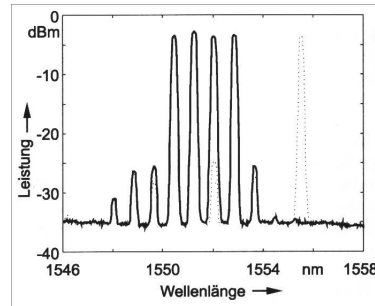
$$\omega_4 = \omega_1 \pm \omega_2 \pm \omega_3, \quad (2.7)$$

wobei alle \pm Kombinationen möglich sind. Damit dies eintritt, müssen jedoch die Beteiligten Wellen möglichst phasengleich sein. Dies kann ausgedrückt werden durch [Agr01]

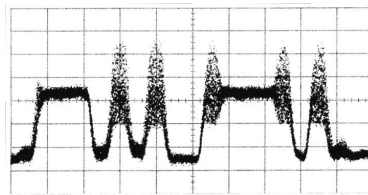
$$\Delta k = k_3 + k_4 - k_1 - k_2 \quad (2.8)$$

wobei $k_i = n \frac{\omega_i}{c}$ und bei $\Delta k = 0$ die Auswirkung der Vier-Wellen-Mischung (Four Wave Mixing, FWM) maximal wird.

Dabei geht Leistung aus den drei ursprünglichen Kanälen verloren. Ist ω_4 eine von dem System bereits belegte Frequenz, entsteht *Übersprechen* (siehe Abbildung 2.9), was die fehlerfreie Demodulation dieses Kanals u.U. unmöglich macht. Das Ausmaß dieses Effektes hängt von dem spektralen



(a) Ausgangsspektrum nach 11 km dispersionsverschobener Faser. Zu den vier eingespeisten Kanälen werden durch FWM weitere Wellen erzeugt.



(b) Typisches durch FWM hervorgerufenes Übersprechen.

Abbildung 2.9: Auswirkungen der FWM. [VP02]

Abstand der Kanäle und von der Dispersion der Faser ab. Ein geringer Abstand sowie eine geringe Dispersion verstärken die Auswirkung der Vier-Wellen-Mischung.

In einem WDM-System mit spektral gleich verteilten Kanälen, haben die durch FWM entstandenen Wellen Frequenzen von benachbarten Kanälen. Um die durch Übersprechen hervorgerufene Signaldegradation gering zu halten, müssen u.U. Frequenzen ausgelassen werden. Das bedeutet, FWM erzwingt eine Abwägung zwischen Übertragungskapazität und Übertragungsdistanz, da viele Kanäle sowohl die Kapazität erhöhen, jedoch auch zur Signaldegradation beitragen. Durch ein geeignetes Dispersionsmanagement und einer geringen Eingangsleistung kann der Einfluss von FWM gering gehalten werden.

Stimulierte Brillouin Streuung

Brillouin Streuung ist eine optische Streuung, die auf der Wechselwirkung einer elektromagnetischen Welle mit einer akustischen Welle beruht. Das durch Brillouin Streuung gestreute Licht ist in der Frequenz um etwa 10 GHz verschoben und hat ein sehr enges Spektrum von unter 100 MHz Bandbreite. Es breitet sich in entgegengesetzter Richtung zu der eintreffenden optischen Welle aus.

Stimulierte Brillouin Streuung (SBS) entsteht erst bei großen Intensitäten. Durch Anwesenheit eines elektrischen Feldes werden Materialien komprimiert. Dies führt dazu, dass durch eine elektromagnetische Welle, also durch ein oszillierendes elektrisches Feld, innerhalb eines Materials (wie z.B. einer Glasfaser) eine akustische Welle erzeugt wird. Die optische Welle wird an der akustischen Welle gestreut, wodurch eine neue optische Welle entsteht, die Stokes-Welle genannt wird, und sich in entgegengesetzter Richtung ausbreitet. Durch Interferenz zwischen dieser und

der ursprünglichen Welle wird die Amplitude der akustischen Welle erhöht, wodurch wiederum die Amplitude der Stokes-Welle erhöht wird.

Da die Stokes-Welle rückwärts gerichtet ist und ihre Leistung exponentiell in der Leistung der ursprünglichen Welle ansteigt, ist eine Konsequenz von SBS, dass für höhere Eingangsleistungen ein immer kleinerer Teil die Faser komplett passiert und ein immer größerer Teil zurück gestreut wird.

Die Entstehung von SBS kann vermieden werden, indem das Signal mit einer Frequenz im kHz Bereich moduliert wird. Dadurch wird die eigentliche Übertragung nicht beeinträchtigt, es wird jedoch die Intensitätsschwelle, die zur Entstehung von SBS überschritten werden muss, nach oben verschoben.

Stimulierte Raman Streuung

Raman Streuung basiert, ähnlich wie Brillouin Streuung, auf Vibrationen des Mediums, durch das sich eine elektromagnetische Welle bewegt; jedoch nicht auf sich ausbreitenden akustischen Wellen, sondern auf Vibrationen einzelner Moleküle. Bei der Wechselwirkung eines Photons mit einem Molekül, kann es zu einer bleibenden Energieübertragung kommen, wobei sich die Schwingungsenergie des Moleküls und die Energie, also die Wellenlänge, des Photons ändert. Es sind beide Fälle, Energieübertragung vom Photon auf das Molekül und anders herum, möglich. Es entsteht also Streulicht von niedriger, als auch von höherer Frequenz.

Das gestreute Licht ist in der Frequenz um etwa 13 THz verschoben und weist ein weites Spektrum von bis zu 30 THz Bandbreite auf. Es breitet sich (anders als SBS) homogen in alle Richtungen aus.

Bei stimulierter Raman Streuung (SRS) sorgt die Interferenz der ursprünglichen Welle mit den Streuwellen für eine Verstärkung der Vibration der Moleküle, die zu einer Erhöhung der Amplitude der Streuwellen führt, wodurch wiederum die Vibration weiter verstärkt wird.

Problematisch ist SRS, weil es in WDM-Systemen zu Übersprechen führt. SRS kann jedoch auch gewinnbringend eingesetzt werden, um durch Energieübertragung von einem *Pump-Strahl* auf das optische Signal der Dämpfung entgegenzuwirken. Die Frequenz des Pump-Strahls wird dabei so gewählt, dass das Streulicht die Frequenz des zu verstärkenden Lichts aufweist. Dabei ist SRS besonders wegen der Einsetzbarkeit in allen Frequenzbereichen interessant.

2.4. Simulationstool PHOTOSS

Der Lehrstuhl für Hochfrequenztechnik entwickelt eine Software zur Simulation optischer Netzwerke namens PHOTOSS. Diese wird bereits seit mehreren Jahren von namhaften Unternehmen der Telekommunikationsbranche eingesetzt. Der PHOTOSS-Benutzer hat die Möglichkeit, über eine grafische Oberfläche Komponenten der optischen Datenübertragung zu Netzwerken zu verschalten. Jede Komponente hat ihre eigenen Parameter, die das Verhalten der Komponente bei der Simulation beeinflussen. Die wesentliche Aufgabe von PHOTOSS ist es, die Übertragung eines Signals durch ein solches, im voraus aufgebautes Netzwerk, zu simulieren, so dass der Benutzer nach der Simulation die Möglichkeit hat, vor sowie nach jeder Komponente das Signal zu analysieren. Dies wird durch dedizierte Komponenten unterstützt, die meist nur einen Eingang aber keinen Ausgang haben. Diese Empfänger-Komponenten bieten dafür die Möglichkeit, das Signal auf eine bestimmte Weise zu visualisieren oder bestimmte Merkmale des Signals zu berechnen. Die Kern-Komponente von PHOTOSS stellt die Faser dar. Alle oben beschriebenen linearen und nichtlinearen

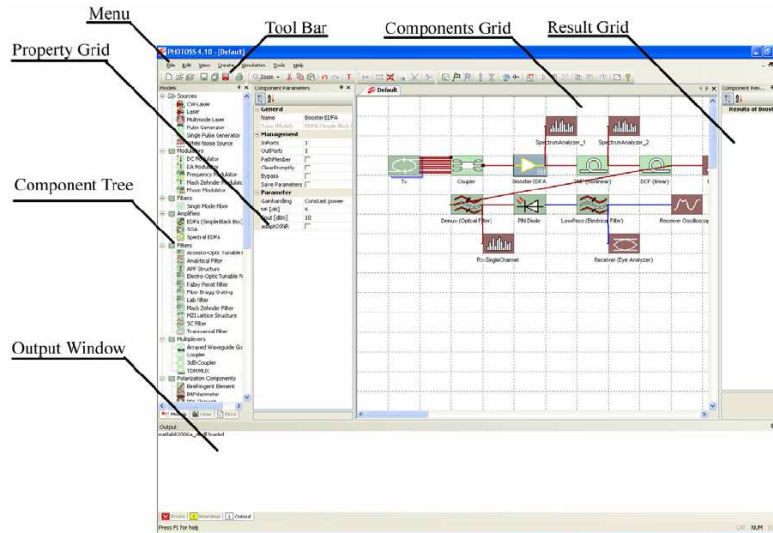


Abbildung 2.10: PHOTOSS-GUI mit Beschreibungen. In der Mitte des Bildes sind mehrere, zu einem Netzwerk verschaltete Komponenten in dem *Components Grid* zu sehen. Die Spalte am linken Bildrand zeigt den *Components Tree* mit einer Reihe von verfügbaren Komponenten.

Effekte werden durch die Berechnungen dieser Komponente simuliert. Dabei können Fasereigenschaften als Parameter gesetzt werden und es kann z.B. bestimmt werden, ob nichtlineare Effekte dabei berücksichtigt werden sollen.

2.4.1. Parametervariation

Um genügend Informationen über das Verhalten eines Netzwerkes zu erlangen, ist es oft nötig, eine Simulation sehr viele Male mit jeweils unterschiedlichen Komponentenparametern zu simulieren. PHOTOSS kann eine solche *Parametervariation* automatisiert ausführen, wenn der Benutzer zuvor die zu variierenden Parameter sowie die gewünschten Ausprägungen der Parameter eingegeben hat. Es wird dann, für den Fall dass mehrere Parameter variiert werden, die Simulation für alle Kombinationen der Parameterausprägungen durchgeführt. Für den Benutzer von Interesse sind die von den Empfänger-Komponenten berechneten Ergebnisse und bei einer Parametervariation die Auswirkung der verschiedenen Parameterausprägungen darauf. Auf diese Weise ließen sich z.B. optimale Werte für Filterbandbreiten der Kanäle eines WDM-Systems und der Dispersioneigenschaften der eingesetzten Faser bestimmen, wenn diese als Parameter gewählt werden, eine sinnvolle Region und Schrittweite für alle Parameter eingestellt wird und auf Empfängerseite eine Analyse-Komponente die Güte des empfangen Signals bestimmt.

Bei mehreren variierten Parametern und vielen Parameterausprägungen entstehen jedoch auch viele Ergebniswerte, so dass es sehr schnell nicht mehr praktikabel ist, Parameterausprägungen mit zugehörigem Ergebniswert in Tabellenform auszugeben (abgesehen von der evtl. unpraktikablen Laufzeit). PHOTOSS bietet zur Visualisierung die Möglichkeit, alle Parameter bis auf einen festzuhalten und den Ergebniswert als Funktion des übrigen Parameters zu plotten.

2.4.2. Architektur

PHOTOSS ist eine monolithische Anwendung, die sowohl Benutzerschnittstelle als auch fachliche Teile, insbesondere die zugrunde liegenden physikalischen Berechnungen, in einem (Betriebs-system-) Prozess vereint. Im normalen Gebrauch, wird nach dem Start von PHOTOSS ein neuer Simulationsaufbau erstellt, dieser mit Komponenten gefüllt und vernetzt und die Simulation gestartet. Ist diese abgeschlossen (was je nach Simulationsart Sekunden bis Stunden dauern kann), kann der Benutzer über das gleiche Fenster die Ergebnisse der Simulation einsehen.

Es gibt aber auch die Möglichkeit, einen Simulationsaufbau im Vorhinein anzufertigen, diesen in einer Datei zu speichern und zu einem späteren Zeitpunkt PHOTOSS mit der Datei als Argument, zu starten. Es wird dann die Simulation sofort durchgeführt, ohne dass eine weitere Benutzerinteraktion erforderlich ist. Wurde zuvor bei der Erstellung des Simulationsaufbaus Entsprechendes eingestellt, werden nun alle Simulationsergebnisse automatisch in eine Ausgabedatei geschrieben. Wurden zudem gewisse Simulationsparameter als variabel deklariert, können die Werte für diese Parameter ebenfalls beim Start des PHOTOSS-Prozesses als Argument übergeben werden. Die Simulation wird dann für diese Parameterausprägungen berechnet. Dies ermöglicht es, PHOTOSS als eine *Black-Box* bzw. als eine Abbildung (von einem Simulationsaufbau mit zugehörigen Parametern auf die Simulationsergebnisse) zu betrachten. Diese Sichtweise ist für die restlichen Teile dieser Arbeit von Bedeutung.

Teil II.

Grundlagen

3. Optimierungsverfahren

Die mathematische Bedeutung der Optimierung ist das Auffinden von optimalen Parametern eines Systems. Optimal bedeutet, dass eine (Bewertungs-) Funktion des Systems für diese Parameter maximal oder minimal wird. Ein konkretes Optimierungsproblem kann wie folgt definiert werden.

Definition 3.0.1 Ein Optimierungsproblem kann beschrieben werden durch ein Tupel (L, f) , wobei L die Menge aller möglichen Lösungen (Systemparametern) und f eine Funktion $f : L \rightarrow \mathbb{R}$ bezeichnet, die jeder Lösung einen Güte- bzw. Kostenwert zuordnet. Ziel ist es, eine Lösung $o \in L$ zu finden, so dass o.B.d.A. gilt

$$\forall l \in L : f(o) \leq f(l). \quad (3.1)$$

Die Aufgabe von Optimierungsverfahren ist, dieses Problem zu lösen. Da dies für allgemeine Bewertungsfunktionen f kaum möglich ist, ohne alle Lösungen zu betrachten, existieren viele Heuristiken, die entweder die betrachteten Funktionen oder die Gültigkeit der Lösung einschränken, dafür aber mit einer praktikablen Anzahl von Auswertungen auskommen.

3.1. Lokale Suche

Lokale Suche [AL97] ist eine Metaheuristik, beschreibt also das Gerüst einer Reihe von Optimierungsheuristiken. Hierbei werden ausgehend von einer gegebenen Initiallösung benachbarte Lösungen gesucht, von denen die Beste ausgewählt wird.

Definition 3.1.1 Eine Optimierungsheuristik gehört zu der Kategorie der lokalen Suche, wenn sie nach folgendem Schema vorgeht:

1. Wähle Initiallösung $l \in L$
2. Bestimme Nachbarschaft $N \subset L$ zu l
3. Suche nach der besten Lösung $o \in N$ der Nachbarschaft

Die Art und Weise, wie die Initiallösung und die Nachbarschaft bestimmt wird und inwiefern (und mit welcher Abbruchbedingung) diese Schritte wiederholt werden, bestimmt die genau Art der lokalen Suche.

3.2. Simulated Annealing

Das Optimierungsverfahren *Simulated Annealing* (simulierte Abkühlung) ist ein heuristisches Verfahren, zum Auffinden von Näherungslösungen eines Optimierungsproblems. Es wurde 1983 von Kirkpatrick et al. vorgestellt [KGV83]. Simulated Annealing wird eingesetzt, wenn es aufgrund der

Komplexität des Problems nicht möglich ist, alle Möglichkeiten auszuprobieren oder einfachere Verfahren anzuwenden.

Die Idee dabei ist, den Abkühlungsprozess eines Materials zu simulieren. Bei einer hohen Temperatur sind die Moleküle kaum an ihre initiale Position gebunden. Eine bestimmte Anordnung von Molekülen ist mit einer Energie gleichzusetzen. Höhere energetische Anordnungen *fallen*, falls dies möglich ist, von alleine in einen Zustand niedriger Energie. Die temperaturabhängige Molekülbewegung jedoch ermöglicht es, dass sich Moleküle aus einem energetischen lokalen Minimum heraus bewegen in einen Zustand höherer Energie, von dem aus Anordnungen erreichbar sind, die evtl. eine geringere Energie aufweisen, als dieses lokale Minimum. In der Metallurgie wird auf diese Weise durch Erhitzung und kontrolliertes Abkühlen die Anzahl und Größe der Kristalle eines Metalls erhöht.

Simulated Annealing überträgt diesen Vorgang auf Optimierungsprobleme im Allgemeinen. Im Wesentlichen ist es eine lokale Suche, bei der jedoch eine benachbarte Lösung auch dann gewählt werden kann, wenn diese nicht die Beste der Nachbarschaft oder sogar schlechter, als die aktuelle Lösung ist. Aus der Nachbarschaft wird per Zufall eine Lösung gewählt. Ist diese besser als die aktuelle, wird die neue Lösung zur aktuellen und es wird von dort aus weiter gesucht. Ist die neue Lösung jedoch nicht besser als die aktuelle, wird sie mit einer gewissen Wahrscheinlichkeit trotzdem akzeptiert. Diese Wahrscheinlichkeit hängt ab von der *Temperatur*, die während der Optimierung stetig sinkt. Damit besteht die Möglichkeit, ein lokales Optimum wieder zu verlassen. Da die Wahrscheinlichkeit, eine schlechtere Lösung zu akzeptieren, mit der Temperatur sinkt, verhält sich das Verfahren gegen Ende wie eine lokale Suche und endet in einem lokalen Optimum. Die Wahrscheinlichkeit, dass dieses das globale (oder zumindest ein *gutes*) Optimum ist, ist jedoch höher, als bei einer lokalen Suche, bei der durch die Wahl der Initiallösung das Ziel-Optimum bereits festgelegt ist.

Algorithmus 3.1 implementiert Simulated Annealing. Als Eingaben werden benötigt:

- eine Initiallösung,
- eine Initialtemperatur
- und die maximale Anzahl erfolgloser Schritte, nach denen die Temperatur gesenkt wird.

Folgende Funktionen werden darin verwendet:

- $neighbour(l)$ liefert einen zufällig gewählten Nachbar der Lösung l
- $accept(f(l_{try}), f(l))$ liefert **true** wenn die neue Lösung l_{try} akzeptiert werden soll. Dies hängt ab von den Werten der aktuellen und der neuen Lösung und vom Zufall. Die Wahrscheinlichkeit, dass akzeptiert wird, ist 1 für den Fall $f(l_{try}) < f(l)$ und

$$P(f(l_{try}), f(l)) = e^{-\frac{f(l_{try}) - f(l)}{t}} \quad (3.2)$$

sonst. Dabei bezeichnet t die aktuelle Temperatur. Die Akzeptanzwahrscheinlichkeit ist also größer, für eine hohe Temperatur und kleiner für schlechtere l_{try} .

- $temperatureFactor(t)$ liefert in Abhängigkeit der aktuellen Temperatur ein $f \in [0, 1]$, mit dem die aktuelle Temperatur multipliziert wird, um die nächste Temperatur zu bilden.

Algorithm 3.1 Simulated Annealing (analog zu dem Algorithmus in [MM92])

```
1: Wähle Initiallösung  $i \in L$ 
2: Wähle Initialtemperatur  $t_0 \in \mathbb{R}$ 
3: Wähle Schrittzahl  $i_{max} \in \mathbb{N}$ 
4:  $l_{best} = l$ 
5:  $t = t_0$ 
6: repeat
7:    $jumped = \text{false}$ 
8:    $i = 1$ 
9:   repeat
10:     $l_{try} = \text{neighbour}(l)$ 
11:    if  $\text{accept}(f(l_{try}), f(l))$  then
12:       $l = l_{try}$ 
13:       $jumped = \text{true}$ 
14:    end if
15:    if  $f(l_{try}) < f(l_{best})$  then
16:       $l_{best} = l_{try}$ 
17:       $i = 1$ 
18:    else
19:       $i = i + 1$ 
20:    end if
21:  until  $i \geq i_{max}$ 
22:   $t = t * \text{temperatureFactor}(t)$ 
23: until  $jumped == \text{false}$ 
```

4. Versuchsplan-Erstellung

Steht man vor dem Problem, Informationen über eine unbekannte und (in irgendeiner Form) teure Funktion, erlangen zu müssen, stellt sich die Frage, an welchen Stellen (für welche Parameterkombinationen) die Funktion ausgewertet werden soll. Dieses Problem kann unabhängig davon betrachtet werden, ob die Funktion von einem Rechner berechnet wird, oder ob reale Experimente nötig sind, um den Wert der Funktion an einer gegebenen Stelle zu ermitteln. Als anschauliches Beispiel kann die Verteilung der Konzentration eines bestimmten Stoffes über einem Grundstück als zweidimensionale Funktion angesehen werden. Die Auswertung kann immer nur an einem Punkt erfolgen. Wird das Grundstück als mathematisch perfekte zwei-dimensionale Fläche angesehen, existieren jedoch unendlich viele Punkte, die ausgewertet werden müssten, um die komplette Funktion zu kennen. Da jede Auswertung mit Kosten verbunden ist, kann eine maximale Anzahl an Auswertungen n angegeben werden, die für eine konkrete, praktische Aufgabenstellung *bezahlbar* ist. Die entscheidende Frage ist, wie müssen diese n auszuwertenden Parameterkombinationen gewählt werden, so dass dadurch möglichst viel Information über die unbekannte Funktion erlangt wird.

Eine solche Menge an auszuwertenden Punkten wird Versuchsplan oder auch Design (eines Experimentes) bezeichnet. Der naive Ansatz, ein solches Design zu erstellen, wäre, die Punkte zufällig zu wählen oder äquidistant über den Parameterraum zu verteilen. Da noch nichts über die Funktion bekannt ist, scheint zunächst kein anderes Vorgehen denkbar. Es existieren jedoch einige statistische Verfahren, die intelligenter vorgehen. Zwei davon werden in diesem Kapitel vorgestellt.

4.1. Latinhypercube-Design

Das *Latinhypercube-Sampling* (LHS) oder auch *Latinhypercube-Design* (LHD) gehört zu den so genannten raumfüllenden Versuchsplänen (*space-filling designs*). LHS wurde 1979 von McKay, Conover und Beckman vorgestellt ([MCB79]). Latinhypercube ist die Verallgemeinerung des Latin-square. Ein quadratisches Raster, das Abtast-Punkte enthält, wird Latin-square genannt, genau dann wenn in jeder Spalte und in jeder Reihe genau ein Punkt enthalten ist. Wird dieses Konzept auf eine beliebige Anzahl an Dimensionen erweitert, heißt es Latinhypercube und die Voraussetzung ist, dass jeder Punkt der einzige auf allen achsen-parallelen Hyperebenen ist, auf denen er liegt. Anders ausgedrückt, für jede Variable gibt es für jeden Bereich genau einen Sample-Punkt in dem Design. Ein LHD kann als Matrix repräsentiert werden:

Definition 4.1.1 [FLS06] Ein *Latinhypercube-Design* mit n Auswertungen und s Eingabe-Variablen, bezeichnet als $LHD(n,s)$, ist eine $n \times s$ Matrix, deren Spalten zufällige Permutationen von $\{1, 2, \dots, n\}$ sind.

Als Beispiel sei folgendes LHD(5,2) angegeben.

$$\begin{pmatrix} 1 & 4 \\ 5 & 5 \\ 3 & 3 \\ 4 & 1 \\ 2 & 2 \end{pmatrix}$$

Jede Zeile repräsentiert eine Auswertung und jede Spalte eine Variable. Der Definitionsbereich jeder Variable wird in n (Anzahl der Samples) Teile unterteilt. Dadurch ergeben sich n^s Hyperwürfel im Parameterraum. Die Ausdehnung eines Hyperwürfels ist durch die Anzahl der Auswertungen bestimmt. Abbildung 4.1 zeigt visuell, wie die Design-Punkte des Beispiel-Designs im Parameterraum verteilt liegen.

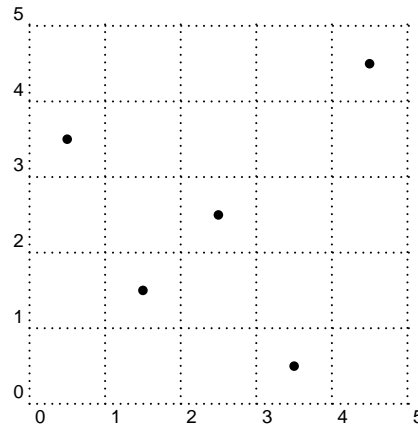


Abbildung 4.1: Visuelle Darstellung des LHD(5,2) Beispieldesigns.

Die Samples müssen nicht unbedingt im Zentrum eines Hyperwürfels liegen. Mit folgendem Algorithmus werden sie zufällig innerhalb ihres Hyperwürfels angeordnet. Der Parameterraum wird auf den Einheitswürfel abgebildet.

Algorithmus 4.1.1 zur Erstellung eines LHS (ebenfalls aus [FLS06]):

1. Erzeuge s unabhängige Permutationen $\pi_j(1) \dots \pi_j(n)$ der Zahlen $1 \dots n$ ($j = 1 \dots s$). D.h. erzeuge ein LHD(n, s)
2. Erzeuge $n \cdot s$ zwischen 0 und 1 unabhängig gleichverteilte Zufallszahlen U_k^j ($k = 1 \dots n$ und $j = 1 \dots s$). Sei $x_k = (x_k^1 \dots x_k^s)$, mit

$$x_k^j = \frac{\pi_j(k) - U_k^j}{n} \quad (k = 1 \dots n, j = 1 \dots s).$$

Dann ist das Design $D_n = \{x_1, \dots, x_n\}$ ein LHS.

Der Vorteil dieses Verfahrens gegenüber einer zufälligen Anordnung der Samples besteht darin, dass die Lage der Samples zueinander mit in Betracht gezogen wird. LHDs sind aufgrund ihrer einfachen Erzeugung und ihrer guten Projektionseigenschaften sehr beliebt. Bei einem LHD ist

sichergestellt, dass sich die Sample-Punkte für jede Variable einzeln betrachtet über den gesamten Definitionsbereich (nahezu) äquidistant verteilen und dass keine Häufung von Sample-Punkten auftritt.

4.2. Maximin-Latinhypercube-Design

Für zwei gegebene Werte n und s gibt es viele mögliche LHD(n,s) Designs. Der in 4.1 vorgestellte Algorithmus ist randomisiert und wählt also zufällig eins aus der Menge aller LHD(n,s) aus. Es gibt jedoch noch weitere Eigenschaften, die diese Designs voneinander unterscheiden. Morris und Mitchell schlagen vor [MM92], aus allen möglichen LHDs ein *Maximin distance design* auszuwählen. Maximin-Designs gehören (wie auch Minimax Designs) zu der Klasse der distanzbasierten Versuchspläne. Ein Maximin-Design maximiert den minimalen Abstand zwischen jeweils zwei Versuchspunkten und stellt damit sicher, dass zwei Punkte nicht zu nahe beieinander liegen.

Definition 4.2.1 [Hen07] Ein Versuchsplan $\mathcal{D} = \{x_1 \dots x_n\}$ aus der Menge χ aller Versuchspläne mit n Punkten ist ein Maximin-Design, falls

$$\min_{\substack{u,v \in \mathcal{D} \\ u \neq v}} \delta(u,v) = \max_{\mathcal{D} \in \chi} \min_{\substack{u,v \in \mathcal{D} \\ u \neq v}} \delta(u,v). \quad (4.1)$$

Während Maximin-Designs die Abstände zwischen den Sample-Punkten betrachten, minimieren Minimax-Designs den maximalen Abstand eines unbeobachteten Punktes zu seinem nächsten Sample-Punkt. D.h. es wird versucht, für jeden möglichen Punkt des Parameterraumes ein Sample möglichst nahe an diesen zu platzieren.

Definition 4.2.2 [Hen07] Ein Versuchsplan $\mathcal{D} = \{x_1 \dots x_n\}$ aus der Menge χ aller Versuchspläne mit n Punkten ist ein Minimax-Design, falls

$$\max_{x \in \Theta} \delta(x, \mathcal{D}) = \min_{\mathcal{D} \in \chi} \max_{x \in \Theta} \delta(x, \mathcal{D}), \quad (4.2)$$

wobei $\delta(x, \mathcal{D}) = \min_{y \in \mathcal{D}} \delta(x, y)$ und Θ den Parameterraum darstellt.

Maximin-Designs sind (im Gegensatz zu Minimax-Designs) einfach zu konstruieren. Beide zeigen aber nicht die, von den LHDs bekannten, wünschenswerten Projektionseigenschaften. Morris und Mitchell kombinieren Maximin- mit Latinhypercube-Designs [MM92], um Versuchspläne zu generieren, die sowohl den Definitionsbereich jedes Parameters für sich genommen äquidistant abdecken, deren Punkte aber auch bezüglich aller Dimensionen gut verteilt liegen.

Bei diesen *Maximin-Latinhypercube-Designs* werden Latinhypercube-Designs sukzessive verändert, bis sie Maximin-Design Eigenschaften aufweisen. Dies geschieht mit Hilfe des Optimierungsverfahrens *Simulated Annealing* (beschrieben in 3.2) und einer Fitness-Funktion, die die Maximin-Eigenschaft eines (Latinhypercube-) Designs berechnet.

Für ein gegebenes (Latinhypercube-) Design \mathcal{D} können die beiden folgenden Listen berechnet werden. Zunächst eine Distanzliste $d = (d_1 \dots d_m)$, die die unterschiedlichen Distanzen zwischen den Design-Punkten in aufsteigend sortierter Reihenfolge enthält. Die zweite Liste $J = (J_1 \dots J_m)$ enthält für jede Distanz der ersten Liste die Anzahl der Punkt-Paare in \mathcal{D} , die durch die entsprechende Distanz separiert sind. Johnson et. al. nannten Designs, die d_1 maximieren und gleichzeitig

J_1 minimieren, *maximin (Mm) designs of minimum index* [MM92][JMY90]. Morris und Mitchell stellen eine Funktionsschar

$$\Phi_p(\mathcal{D}) = \left(\sum_{j=1}^m J_j d_j^{-1} \right)^{\frac{1}{p}} \quad (4.3)$$

vor, so dass Designs einer bestimmten Klasse, die eine solche Funktion minimieren, Mm-Designs dieser Klasse sind. Für die gegebene Klasse der LHD(n,s) Designs (für gegebene n und s), lässt sich mit Hilfe einer solchen *Maximin-Bewertungsfunktion* das Latinhypercube-Design finden, das einem Maximin-Design am ehesten entspricht.

Dazu wird zunächst ein LHD randomisiert erzeugt. Mithilfe von Simulated Annealing wird dieses Design modifiziert und bzgl. einer Funktion aus o.g. Schar optimiert. Als Modifikation wird hier das Vertauschen von zufälligen Elementen einer zufälligen Spalte der Design-Matrix gewählt. Ist das Design vor dieser Modifikation ein LHD, so ist es dies auch danach. Simulated Annealing vergleicht den Wert der Fitness-Funktion für das so gewonnene, neue Design mit dem für das alte Design. Ist das neue Design besser, wird damit weiter gearbeitet, ist es schlechter, wird es mit einer gewissen Wahrscheinlichkeit entweder auch behalten oder es wird verworfen und eine andere Modifikation des ursprünglichen Designs erzeugt. Scheint keine Verbesserung mehr möglich, bricht Simulated Annealing ab und liefert das (bezüglich der Fitness-Funktion) gefundene Optimum; in diesem Fall das LHD mit optimalen Maximin-Eigenschaften.

5. Funktionsschätzer

Nachdem eine unbekannte Funktion an wenigen Stellen ausgewertet wurde, besteht die Möglichkeit, mit Hilfe der gewonnenen Information die Funktionswerte an nicht ausgewerteten Stellen zu schätzen. Dazu kann aus einer Fülle an möglichen Schätzverfahren gewählt werden. Grob können diese in *Approximations-* und *Interpolationsverfahren* unterteilt werden. Interpolanten nehmen an den bekannten Stellen die korrekten (durch die Auswertung der Funktion erhaltenen) Werte an. Bei der Approximation werden die Parameter einer parameterisierten Funktion so gewählt, dass der aufsummierte Abstand der Funktion zu den bekannten Werten minimal wird. D.h., dass ein Approximant nicht unbedingt durch die bekannten Punkte laufen muss. Bei den meisten Interpolationsverfahren geht man allerdings von einer bestimmten räumlichen Verteilung der Interpolationspunkte aus. Für die Interpolation zwischen unregelmäßig gestreuten Datenpunkten bietet sich die so genannte *Streudateninterpolation* (*scattered data interpolation*) an. In dieser Arbeit kommt aber nur die Approximation oder die Streudateninterpolation zum Einsatz, um keine Einschränkungen für die Platzierung der auszuwertenden Punkte einführen zu müssen.

In 5.1 wird die Approximation am Beispiel einer Hyperebene erklärt. 5.3 und 5.2 stellen (Streudaten-) Interpolationsverfahren vor. Hauptquelle für dieses Kapitel ist [HL87]. Kapitel 9 in [HL87] befasst sich ausführlich mit der Streudateninterpolation.

5.1. Ausgleichsebene

Zur groben Schätzung der unbekannten Funktionswerte kann eine Hyperebene berechnet werden, die möglichst nah an den bekannten Punkten liegt. Auch wenn die Abschätzung über eine Ebene recht grob ist, kann evtl. der Trend der Funktion daran abgelesen werden.

Definition 5.1.1 Eine (Hyper-) Ebene im d -dimensionalen Raum ist (analog zur Geradengleichung des \mathbb{R}^2) gegeben durch

$$E(\vec{x}) = a_1x_1 + a_2x_2 + \cdots + a_{d-1}x_{d-1} + a_d, \quad (5.1)$$

wobei das Argument \vec{x} ein $d - 1$ dimensionaler Vektor ist (dessen Komponenten die x_i sind) und $E(\vec{x})$ den Wert der Ebene in der übrigen Dimension angibt.

Sind genau d d -dimensionale Punkte gegeben, kann ein Gleichungssystem aufgestellt werden, das Lösungen für die a_i und damit eine Hyperebene liefert, die durch diese Punkte verläuft. Mehr als d Punkte im d -dimensionalen Raum müssen nicht auf einer Hyperebene liegen. Es kann aber eine Hyperebene berechnet werden, die möglichst nahe an allen gegebenen Punkten liegt.

Der $(d - 1)$ dimensionale Parameterraum der unbekannten Funktion zusammen mit dem (ein-dimensionalen) Wertebereich bilden den d -dimensionalen Raum der Approximationsebene. Für eine gegebene Auswertung der Funktion, bestehend aus dem Parametervektor \vec{x} und seinem Funktionswert y , kann die zu minimierende Differenz zu einer gegebenen Approximationsebene

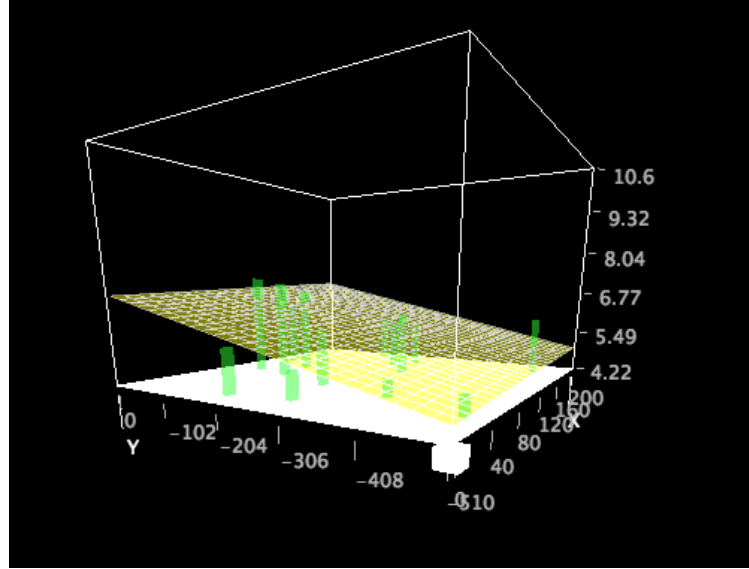


Abbildung 5.1: Ausgleichsebene, die die gegebenen Punkte (grüne Säulen) approximiert. Die zu schätzende Funktion hängt von zwei Parametern ab, die hier durch die beiden horizontalen Dimensionen repräsentiert werden. Die Höhe spiegelt den Funktionswert an der entsprechenden Stelle wider. Die Ebene zeigt den groben Trend der Funktion an; es ist zu erkennen, dass kleine Funktionswerte eher in der rechten, vorderen Ecke zu finden sind.

mit

$$d(\vec{x}, \vec{a}) = \left| \begin{pmatrix} x_1 & x_2 & \dots & x_{d-1} & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{d-1} \\ a_d \end{pmatrix} - y \right| \quad (5.2)$$

berechnet werden. Wobei $x_1 \dots x_{d-1}$ die Elemente des Vektors \vec{x} und $a_1 \dots a_d$ (Elemente von \vec{a}) die Parameter der Ebene sind und damit das Produkt der beiden Vektoren der Funktionswert der Ebenengleichung 5.1 an der Stelle \vec{x} ist.

Um die Ebenenparameter so zu wählen, dass die resultierende Ebene möglichst gut die vorhandenen Daten approximiert, muss das Minimierungsproblem

$$\min_{\vec{a}} \left\| \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d-1} & 1 \\ x_{2,1} & x_{2,2} & \dots & x_{2,d-1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,d-1} & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{d-1} \\ a_d \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \right\| \quad (5.3)$$

gelöst werden. Wobei $\|\cdot\|$ die euklidische Norm bezeichnet und $x_{i,j}$ und y_i die Daten darstellen, die durch Auswertung der Funktion gewonnen wurden. $x_{i,j}$ ist das j -te Element und y_i der Funktionswert des i -ten Parametervektors. Die a_p sind die, durch die Minimierung zu berechnenden, Ebenenparameter.

Abbildung 5.1 zeigt als Beispiel eine Ebene, die gegebene Punkte in einem dreidimensionalen Raum approximiert. Zwei Dimensionen repräsentieren die Parameter der zu schätzenden Funktion; die dritte Dimension ist der Funktionswert.

Eine solche Ausgleichsebene ist einfach zu berechnen. Sie gibt den groben Trend der, durch die Abtastpunkte repräsentierten, Funktion an. D.h. es ist an der Ausgleichsebene abzulesen, in welche Raumrichtung steigende und in welche Richtung fallende Funktionswerte zu erwarten sind. Jedoch ist es nicht möglich, mit der Ausgleichsebene Extremwerte zu finden. Eine Optimumsuche auf der Ebene wird das Optimum immer am Rand bzw. in einer Ecke des betrachteten Bereiches finden (es sei denn, $a_1 \dots a_{d-1}$ sind 0, so dass die Ebene überall den gleichen Funktionswert annimmt).

5.2. Shepard-Interpolant

Ein ebenfalls sehr bekanntes Verfahren ist die *Shepard-Methode*, auch bekannt als *Shepard-Interpolant* oder *inverse Distanzgewichtung*. Dies ist im Wesentlichen ein gewichtetes Mittel der bekannten Funktionswerte f_i .

Definition 5.2.1 [HL87] Ein Shepard-Interpolant hat die Gestalt

$$f(x) = \sum_{i=1}^N \omega_i f_i, \quad (5.4)$$

mit Gewichtsfunktionen

$$\omega_i(x) = \frac{\sigma_i(x)}{\sum_{j=1}^N \sigma_j(x)} \quad (5.5)$$

und

$$\sigma_i(x) = \frac{1}{d_i(x)^{\mu_i}} = \frac{1}{(\sum_{j=1}^d (x_j - x_{ij})^2)^{\frac{\mu_i}{2}}}, \quad (5.6)$$

wobei N die Anzahl der Interpolationspunkte, f_i der i -te Interpolationspunkt und $d_i(x)$ der Abstand des Punktes x zum i -ten Interpolationspunkt ist.

Die Parameter μ_i sind vom Benutzer zu wählen. Es lässt sich zeigen, dass der Interpolant

- für $0 < \mu_i < 1$ an dem i -ten Interpolationspunkt eine Spitze,
- für $\mu_i = 1$ an dem i -ten Interpolationspunkt eine Ecke und
- für $\mu_i > 1$ an dem i -ten Interpolationspunkt einen Flachpunkt (Punkt mit Steigung Null) hat.

Bei der Wahl $\mu_i = 2$ kürzt sich in Gleichung 5.6 der Exponent zu 1, was günstig für die Rechenzeit ist. Gleichung 5.5 kann zur Berechnung durch den numerisch stabileren Ausdruck

$$\omega_i(x) = \frac{\prod_{j \neq i} d_j(x)^{\mu_i}}{\sum_{k=1}^N \prod_{j \neq k} d_j(x)^{\mu_i}} \quad (5.7)$$

ersetzt werden.

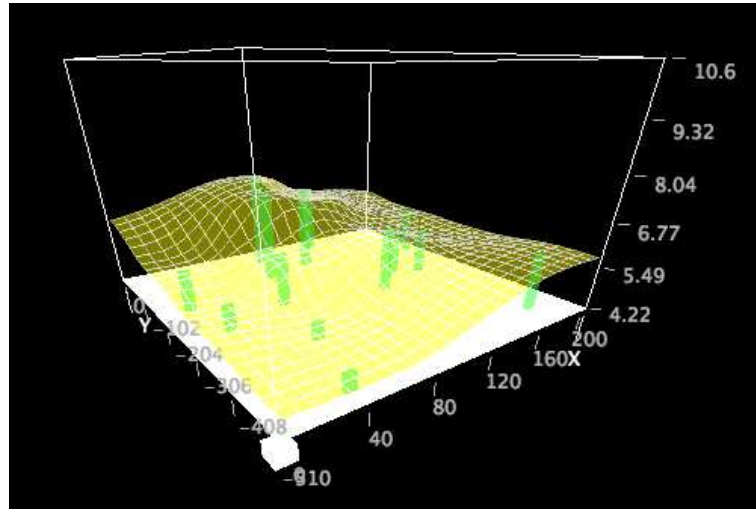


Abbildung 5.2: Shepard-Interpolant für die gegebenen Punkte (Parameter $\mu = 4$).

In [HL87] wird vorgestellt, wie durch eine rekursive Repräsentation dieses Verfahren zu einem lokalen Interpolationsverfahren gemacht werden kann. Dann wäre es möglich, Stützstellen hinzuzufügen, ohne alle Punkte neu berechnen zu müssen. Dies ist jedoch für diese Arbeit nicht relevant, da die Berechnung der Simulation schon wesentlich länger dauert, als die Berechnung der relevanten Shepard-Interpolanten.

Da immer

$$\sum_{i=1}^N \omega_i = 1 \quad (5.8)$$

gilt, ist jeder geschätzte Wert $f(x)$ eine konvexe Kombination der bekannten Stützstellen. Dies bedeutet, dass kein $f(x)$ kleiner (oder größer) als der kleinste (oder größte) bekannte Funktionswert der Stützstellen ist. In weiter Entfernung von allen Stützstellen nähert sich der Shepard-Interpolant dem Mittelwert aller bekannten Funktionswerte an. Die Extremwerte des Shepard-Interpolanten liegen immer auf den Stützstellen und haben deren Funktionswert. Um also die Extremwerte des Shepard-Interpolanten zu finden, reicht es aus, die kleinste oder größte Stützstelle zu ermitteln. D.h. wenn es nur um das auffinden von Extremwerten geht, bringt die Shepard-Methode keinen Gewinn gegenüber der Betrachtung der Stützstellen allein. Trotzdem macht die Implementierung dieses Verfahrens für die bearbeitete Aufgabe Sinn. Die Funktionswerte des Shepard-Interpolanten in der Umgebung eines Extremwertes geben einen Hinweis auf die Ausdehnung dieser potentiell interessanten Region. Ausserdem gibt die Visualisierung einer geschlossenen Fläche einen wesentlich besseren Eindruck, als einzelne Stützstellen.

5.3. Hardy-Multiquadrik

Multiquadriken (MQ) sind Interpolanten für Streudaten und gehören zu der Klasse der radialen Basisfunktionsmethoden. Diese heißen so, da dabei der Interpolant aus Funktionen aufgebaut wird, die nur vom Abstand des auszuwertenden Punktes zu den Interpolationspunkten abhängen.

Definition 5.3.1 [HL87] Ein Streudaten-Interpolant basierend auf radialen Basisfunktionen hat die Gestalt

$$f(x) = \sum_{i=1}^N \alpha_i R(d_i(x)) + p_m(x) \quad (5.9)$$

mit

$$p_m(x) = \sum_{j=1}^m \beta_j \rho_j(x), \quad (5.10)$$

wobei $d_i(x)$ der Abstand des Punktes x zum i -ten Interpolationspunkt und $R(d_i(x))$ eine radiale positive Funktion ist. Die ρ_j sind Polynome mit Grad kleiner gleich m . N ist die Anzahl der Interpolationspunkte, und m muss vom Benutzer gewählt werden.

Ein solcher Interpolant wird durch die Koeffizienten α_i $i = 1 \dots N$ und β_j $j = 1 \dots m$ bestimmt. Mit der Forderung, dass der Interpolant durch die Interpolationspunkte verlaufen soll (also $f(x_i) = f_i$ für bekannte Punkte (x_i, f_i) , $i = 1 \dots N$ gelten soll) und den m Nebenbedingungen

$$\sum_{i=1}^N \alpha_i \rho_j(x_i) = 0, \quad j = 1 \dots m \quad (5.11)$$

ergibt sich ein Gleichungssystem, mit dem diese Koeffizienten berechnet werden können. Die Lösbarkeit dieses Gleichungssystems hängt von den vorliegenden Interpolationspunkten und den gewählten Parametern ab. Ab einer gewissen Anzahl an Interpolationspunkten, könnte es schlecht konditioniert sein, so dass evtl. besondere Techniken angewendet werden müssen [HL87].

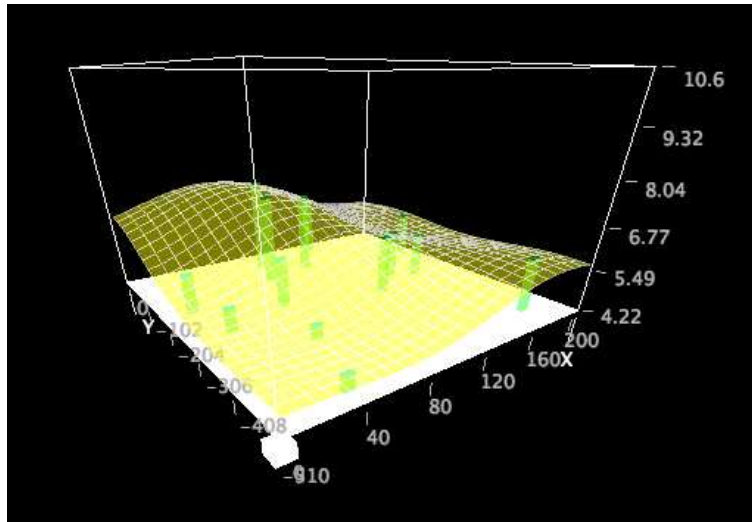


Abbildung 5.3: Hardy-Multiquadrik angepasst an die gegebenen Punkte. Mit Parameter $\mu = 1$ und R berechnet mit Algorithmus 5.3.1.

Für Multiquadriken ist jedoch $m = 0$ zu setzen. D.h. Gleichung 5.9 vereinfacht sich zu

$$f(x) = \sum_{i=1}^N \alpha_i R(d_i(x)). \quad (5.12)$$

Für die (nach dem Erfinder Hardy benannte) *Hardy-Multiquadrik* ist die Basisfunktion R gegeben durch

$$R(r_i) = (r_i^2 + R_i^2)^{\frac{\mu_i}{2}}, \quad \mu \neq 0, \quad (5.13)$$

wobei r_i , R_i und μ vom Benutzer gewählt werden müssen. Für μ wählte Hardy $\mu_i = \mu = 1$. Micchelli zeigte, dass für $\mu \leq 1$ das korrespondierende Gleichungssystem immer lösbar ist [Mic86]. r_i wird in der Regel als $d_i(x)$ und R_i als eine Konstante R gewählt. Wobei es hier mehrere sinnvolle Möglichkeiten gibt. Hardy verwendete $R = 0,815d$ mit d als mittleren Abstand eines Datenpunktes zu seinem nächsten Nachbarn. Ein an die Daten angepasstes R liefert folgender Algorithmus.

Algorithmus 5.3.1 [HL87]

1. Skalieren Datenpunkte auf Einheitswürfel.
2. Berechne ein biquadratisches Ausgleichspolynom $Q(x)$ der Datenpunkte (x_i, f_i) .
3. Berechne $R^2 = \frac{1}{1+120V}$, wobei V die Varianz $V = \frac{1}{N} \sum_{i=1}^N (f_i - Q(x_i))^2$, also die gemittelte Abweichung des Ausgleichspolynoms ist.
4. Berechne den Interpolanten für $\mu = 1$ oder $\mu = -1$ und dem zuvor bestimmten R .
5. Skalieren den erhaltenen Interpolanten auf den ursprünglichen Bereich.

In [HL87] finden sich die hier beschriebenen Beispiele für R und noch weitere.

Anders als bei der Shepard-Methode müssen die Extremwerte der Hardy-Multiquadrik nicht auf den Stützstellen liegen. D.h. die Hardy-Multiquadrik kann einen Trend der Stützstellen fortführen und Werte annehmen, die größer oder kleiner als die größte bzw. kleinste Stützstelle ist.

Obwohl das zu lösende Gleichungssystem für $\mu \leq 1$ immer lösbar ist, kann es numerisch bedingt bei der Lösung des Systems durch eine Software zu Problemen kommen. Bei der im Zuge dieser Arbeit erstellten Implementierung wurde stattdessen die Aufgabe als Minimierungsproblem

$$\min_{\vec{\alpha}} \left\| f(x) - \sum_{i=1}^N \alpha_i R(d_i(x)) \right\| \quad (5.14)$$

formuliert. Da $\mu = \frac{1}{2}$ gewählt wurde, ist sichergestellt, dass $f(x)$ und die Summe immer den (bis auf numerisch bedingte Schwankungen) gleichen Wert annehmen, so dass

$$f(x) \approx \sum_{i=1}^N \alpha_i R(d_i(x)). \quad (5.15)$$

gilt.

6. Visualisierung multivariater Daten

Sobald Daten von mehr als drei Dimensionen abhängen, erfordert es Überlegungen, wie diese darzustellen sind. Zunächst ist zu unterscheiden, welche Art von zu visualisierenden Daten vorliegen.

6.1. Multivariate Datensätze

Es wurden viele Verfahren entwickelt, die es ermöglichen, *multivariate Datensätze* darzustellen. Dabei wird ein Datum als Vektor $x = (x_1, x_2, \dots, x_n)$ eines n -dimensionalen Raumes und alle Datensätze als eine Menge solcher Vektoren aufgefasst. Ein grundlegender Ansatz ist, diese Vektoren auf eine zweidimensionale Ebene zu projizieren. Da es jedoch sehr viele Möglichkeiten gibt, dies zu tun, ist die Aufgabe, eine möglichst aussagekräftige Projektion zu finden. Dafür werden z.B. die *Hauptachsentransformation* oder das *Grand Tour System* [Asi85] eingesetzt. Einen weiteren Ansatz bilden die Pixeldiagramme (Iconic Displays), wie z.B. die *Chernoff'schen Gesichter* [Che73] oder die *Strichmännchen-Technik* (Stick Figures) [Pic70]. Auch dabei werden die Daten auf eine zweidimensionale Ebene projiziert, jedoch werden die Werte der übrigen Dimensionen ebenfalls visualisiert, indem anstatt eines Punktes ein Icon gezeichnet wird, dessen Form von den Werten der übrigen Dimensionen abhängt.

6.2. Multivariate Funktionen

Diese Verfahren können jedoch nicht eingesetzt werden, wenn, wie in dieser Arbeit, eine *multivariate, skalare Funktion*, visualisiert werden soll (also eine Funktion, die mehrere Argumente auf ein Skalar abbildet). In diesem Fall ist ein Datum als ein Vektor $x = (x_1, x_2, \dots, x_n, y)$ zu verstehen, wobei eine Funktion $f : \mathcal{D} \rightarrow \mathcal{R}$ existiert und $(x_1, x_2, \dots, x_n) \in \mathcal{D}$ sowie $f(x_1, x_2, \dots, x_n) = y$ gilt. Ein weiterer Unterschied zu dem ersten Fall ist, dass für alle $d \in \mathcal{D}$ ein Datum $x = (d_1, d_2, \dots, d_n, y)$ existiert. Die Daten enthalten also nicht die Information, ob eine gewisse Merkmalskombination vorliegt, sondern auf welchen Skalar eine Merkmalskombination abgebildet wird, wobei immer alle Merkmalskombinationen vorliegen. Dies erfordert andere Visualisierungsverfahren. Beim *dimensionalen Stapeln* (dimensional stacking) [LWW90] werden möglichst alle Dimensionen hierarchisch auf ein zweidimensionales Display abgebildet. Zunächst werden zwei Dimensionen ausgewählt, die das Display grob unterteilen. Zu jeder Wertekombination dieser zwei Dimensionen korrespondiert eine Region des Displays. Diese Regionen werden von den nächsten zwei Dimensionen unterteilt und die dabei entstandenen Regionen von den nächsten usw. Sind alle Dimensionen untergebracht, kann jedem Element des Definitionsbereichs eine Region des Displays (ein Pixel) zugeordnet werden. Der Wert der Funktion an dieser Stelle wird durch die Farbe, die diese Region bekommt, dargestellt. Eine ähnliche Technik wird in [MTS91] beschrieben.

Bekannte Visualisierungstechniken für ein-, zwei- und dreidimensionale Funktionen, wie ein- und zweidimensionale Plots oder Isolinien bzw. Isoflächen, lassen sich anwenden, sobald ein

Teilraum des Definitionsbereichs festgelegt ist. Wie bei der Projektion multivariater Datensätze auf eine zweidimensionale Ebene ist hier die Wahl des Teilraumes entscheidend.

6.3. Gewähltes Verfahren

Die in dieser Arbeit erstellte Software hat die Aufgabe, multivariate Funktionen zu visualisieren, wobei die Anzahl an Dimensionen nicht eingeschränkt sein soll. Der dazu gewählte Ansatz ist die Visualisierung einer Funktion über einen zweidimensionalen Unterraum des Definitionsbereiches. Diese zweidimensionale Teilfunktion wird visualisiert, indem in einer dreidimensionalen Szene der Funktionswert auf der dritten Dimension abgetragen wird (siehe Abbildung 6.1).

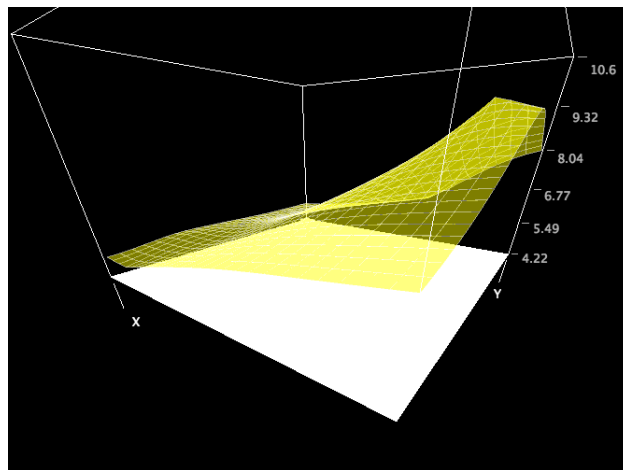


Abbildung 6.1: Zweidimensionale Funktion in einer dreidimensionalen Szene dargestellt. Funktionswert auf der dritten Dimension abgetragen.

Das bedeutet, es wird nie die gesamte Funktion visualisiert, sondern immer nur ein Teilbereich. Damit die angezeigte zweidimensionale Teilfunktion immer einfach interpretierbar ist, werden nur achsenparallele Ebenen als Unterraum des Definitionsbereiches ausgewählt. Dadurch sind die Achsen der visualisierten zweidimensionalen Funktion immer eine Auswahl der Achsen der mehrdimensionalen Funktion, und keine Kombinationen dieser.

Der Ort und die Ausrichtung dieser Schnittebenen sind variabel und sollen interaktiv modifiziert werden können, um durch die zeitliche Abfolge verschiedener dadurch entstandener Bilder einen Eindruck der mehrdimensionalen Funktion zu bekommen.

7. Verteilte Berechnungen

Wenn es darum geht, möglichst viel Gesamtrechenleistung zu erreichen, gibt es kaum (wirtschaftlich relevante) Alternativen zu dem Einsatz vieler günstiger Rechensysteme. Aufgrund ihrer Verbreitung sind IBM-PC kompatible Rechner, wie sie heutzutage als (Desktop-) Standardhardware eingesetzt werden, günstig. Für die Durchführung aufwendiger Berechnungen ist es in den meisten Fällen wirtschaftlich vorteilhaft, viele solcher gängigen Computer einzusetzen, anstatt Spezialhardware einzukaufen.

Es gibt verschiedene Verfahren, Rechner zu vernetzen, so dass sie gemeinsam eine Aufgabe lösen können. Diese können in die beiden Klassen *Cluster Computing* und *Grid Computing* unterteilt werden. Ein Cluster kann (meistens) von außen als ein Computer betrachtet werden. Auszuführender Programmcode wird über schnelle Netzwerkverbindungen an die einzelnen Elemente des Clusters verteilt, die diesen parallel ausführen. Der Benutzer eines Clusters muss davon jedoch nichts mitbekommen, da das gesamte System sich wie ein Rechner mit einer großen Anzahl CPU-Kerne verhält.

Ein Computing Grid hingegen hat im Wesentlichen die Aufgabe, Berechnungsanfragen (*Jobs*) an freie Rechner des Grids zu verteilen. D.h. das Grid hilft dabei, Ressourcen zugänglich zu machen und zu verwalten. Dabei läuft ein Job aber nur auf einem Rechner. Innerhalb eines Computing Grids laufen aber viele Jobs gleichzeitig auf vielen verschiedenen Rechnern.

Die für diese Arbeit relevanten Berechnungen können leicht in viele kleine Jobs aufgeteilt werden. Die Simulationen eines optischen Netzes mit unterschiedlichen Parametern hängen nicht voneinander ab, so dass jede Parameterkombination als einzelner Job berechnet werden kann. Da zur Simulation das Programm PHOTOSS benutzt wurde, dessen Code schlecht parallelisierbar ist, ist es sinnvoll eine PHOTOSS-Instanz pro CPU-Kern zu starten. Beides sind Anforderungen, die sowohl von Cluster- als auch von Grid-Verfahren erfüllt werden können. Da jedoch am Lehrstuhl für Hochfrequenztechnik viele IBM-PC kompatible Rechner zur Verfügung stehen, die weiterhin als Desktop-Arbeitsplätze nutzbar bleiben sollen, bietet sich ein Grid-Verfahren an. Die Software *Condor* wurde an dem Lehrstuhl bereits erfolgreich in Verbindung mit PHOTOSS eingesetzt. Diese Software ermöglicht es, ein Computing Grid aufzubauen, in dem die Rechner sich bei Nichtbenutzung verfügbar melden, um Jobs anzunehmen, die sie im Hintergrund berechnen, ansonsten aber vollwertige Desktop-Arbeitsplätze bleiben. Da an dem Lehrstuhl schon Erfahrung in der Benutzung mit Condor besteht, und es kostenfrei verfügbar ist, baut auch diese Arbeit darauf auf. Condor wird im Folgenden genauer beschrieben.

7.1. Condor Grid-Computing Software

Condor wird seit 1988 von der University of Wisconsin in Madison entwickelt und ist frei verfügbar [Con][TTL05]. Es wurde ursprünglich für Unix entwickelt, läuft nun aber auch auf Linux, MacOSX, FreeBSD und neueren Windows-Systemen. Es kann dazu benutzt werden, Rechenlast innerhalb eines Computing Grids zu verteilen. Dabei können innerhalb eines *Pools* dedizierte Ressourcen (Rechner, die nur diesen einen Zweck erfüllen) mit nicht-dedizierten (z.B. Arbeitsplatz-Rechner)

gemischt werden. Für jeden Rechner kann einzeln entschieden werden, wann Jobs angenommen werden und welche Jobs angenommen werden. Jeder Rechner eines Condor-Pools kann potentiell Jobs in Auftrag geben und jeder kann potenziell Jobs annehmen. Auf diese Weise können ungenutzte Ressourcen, wie z.B. die Rechenzeit von Arbeitsplatz-Rechnern während der Mittagspause oder nachts, einfach nutzbar gemacht werden.

Jeder Condor-Pool benötigt einen Condor-Master, der für die Zuordnung der Jobs zu den Ressourcen zuständig ist. Wird ein Job von einem Rechner akzeptiert, werden die nötigen Dateien (die ausführbare Datei, die das zu berechnende Programm enthält und evtl. Datendateien) von dem Rechner, der den Job in Auftrag gibt, zu dem Rechner, der den Job berechnet, kopiert. Auf dem ausführenden Rechner wird ein temporärer Benutzer angelegt, der das zu berechnende Programm ausführt. Potentielle Ausgabedateien werden danach zu dem Rechner, von dem aus der Job aufgegeben wurde, kopiert.

Job-Dateien

Um einen Job in Auftrag zu geben, muss eine Condor-Job-Datei geschrieben werden, und diese dem Programm *condor_submit* als Argument übergeben werden. Daraufhin wird ein neuer Job generiert, der von dem Condor-Master einem Rechner zugewiesen und wie oben beschrieben berechnet wird.

Listing 7.1: Condor-Job-Datei zur Erzeugung eines PHOTOSS-Jobs

```

1 EXECUTABLE           = C:/Programme/Photoss/Photoss.exe
2 UNIVERSE             = VANILLA
3 SHOULD_TRANSFER_FILES = YES
4 WHEN_TO_TRANSFER_OUTPUT = ON_EXIT
5 TRANSFER_INPUT_FILES = sim.pho,\
6 C:/Programme/Photoss/ApplicationBasics.dll,\
7 C:/Programme/Photoss/DefaultModelTree.txt,\
8 C:/Programme/Photoss/DefaultUserTree.txt,\
9 C:/Programme/Photoss/MFC71.dll,\
10 C:/Programme/Photoss/condor_exec.exe.manifest,\
11 C:/Programme/Photoss/ToolkitPro1010vc80.dll,\
12 C:/Programme/Photoss/component_dllsdev_kit.dll,\
13 C:/Programme/Photoss/matlabR12.dll,\
14 C:/Programme/Photoss/matlabR13.dll,\
15 C:/Programme/Photoss/matlabR14.dll,\
16 C:/Programme/Photoss/matlabR2006a.dll,\
17 C:/Programme/Photoss/msvc71.dll,\
18 C:/Programme/Photoss/msvcr71.dll,\
19 C:/Programme/Photoss/photoss_dll.dll,\
20 C:/Programme/Photoss/visualizer_dll.dll
21 ARGUMENTS = -r sim.pho
22 QUEUE

```

Listing 7.1 zeigt eine solche Job-Datei. Dieses Beispiel erzeugt einen Job, der eine PHOTOSS-Simulation ausführt. In der ersten Zeile wird das ausführbare Programm angegeben. Da PHOTOSS aber eine Reihe von DLL-Dateien benötigt, müssen diese ebenfalls angegeben werden, um zusammen mit dem Programm auf den ausführenden Rechner übertragen zu werden (Zeilen 6-20).

Außerdem wird die Datei *sim.pho* übertragen. Dies ist eine PHOTOSS-Simulationsdatei, die von PHOTOSS geladen und simuliert werden kann. Zeile 21 lässt Condor PHOTOSS mit den Argumenten

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>condor_status

Name           OpSys      Arch    State    Activity    LoadAv  Mem    ActvtyTime
Curie_hft.e-t  WINNT51    INTEL   Unclaimed Idle         0.000   1023   0+00:10:04
Fraunhofer_hf  WINNT51    INTEL   Unclaimed Idle         0.980   1023   0+01:00:06
simulator05.h  WINNT51    INTEL   Unclaimed Idle         0.040    959   0+01:30:05

          Total Owner Claimed Unclaimed Matched Preempting Backfill
          INTEL/WINNT51      3      0      0      3      0      0      0
          Total      3      0      0      3      0      0      0

H:\>condor_q

-- Submitter: simulator05_hft.e-technik.uni-dortmund.de : <129.217.188.164:1483>
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
0 jobs; 0 idle, 0 running, 0 held
H:\>

```

Abbildung 7.1: Bildschirmfoto von *condor_status* und *condor_q*.

-r *sim.pho* ausführen, die PHOToss dazu veranlassen, sofort die Datei *sim.pho* zu laden und zu simulieren (anstatt auf eine Benutzereingabe zu warten). Dadurch ist erst möglich, PHOToss als Berechnungs-Prozess zu verwenden, der ohne Benutzereingabe auskommt, und so über Condor auf einem beliebigen Rechner ausgeführt werden kann.

Die letzte Zeile sorgt dafür, dass tatsächlich ein Job mit den zuvor eingestellten Eigenschaften erzeugt wird. (Um zwei identische Jobs zu erzeugen, könnte einfach eine weitere Zeile mit *queue* angehängt werden.)

Wegen Zeile vier werden alle von dem Programm erzeugten Dateien nach erfolgreicher Ausführung von Condor auf den Rechner kopiert, von dem aus der Job mit Hilfe von *condor_submit* aufgegeben wurde. Die Dateien werden in das Verzeichnis abgelegt, das das aktuelle Arbeitsverzeichnis beim Aufruf von *condor_submit* war.

Tools

Zur Verwaltung von Jobs bringt Condor mehrere Programme mit. *condor_submit* wurde schon im letzten Abschnitt angesprochen. Mit *condor_status* kann zunächst eine Übersicht über den hiesigen Pool verschafft werden. Es werden alle Rechner des Pools und deren Status (*verfügbar, belegt...*) angezeigt (Abbildung 7.1).

Um eine Übersicht über zuvor aufgegebene Jobs zu erhalten, kann *condor_q* verwendet werden. Ohne Argumente gibt *condor_q* eine Liste aller in Auftrag gegebenen und noch nicht beendeten Jobs des Benutzers. *condor_q* kann aber auch benutzt werden, um Informationen zu einem bestimmten Job zu erlangen. Z.B. kann mit diesem Tool herausgefunden werden, warum ein Job noch nicht ausgeführt wurde.

Bestehende Jobs können mit *condor_rm* und der ID-Nummer des Jobs (die von *condor_submit* und *condor_q* angezeigt wird) entfernt werden.

Mit *condor_hold* und *condor_release* können Jobs (vor der Ausführung) zurückgehalten bzw. wieder freigegeben werden.

condor_prio setzt die Priorität eines Jobs (die bestimmt, welcher Job eines Benutzers eine freiwerdende Ressource erhalten soll).

Für eine detaillierte Beschreibung aller Programme siehe [Con].

Logs

Konnte ein Job erfolgreich dem Pool hinzugefügt werden, legt *condor_submit* Log-Dateien an, in denen Informationen zu dem Job gesammelt werden. Die Namen dieser Dateien können über Einträge der Job-Datei bestimmt werden.

Listing 7.2: Variablen einer Condor-Job-Datei zum festlegen der Log-Dateien.

```
1 OUTPUT = job.out
2 ERROR  = job.err
3 LOG    = job.log
```

Sind diese Zeilen in einer Job-Datei enthalten, wird in *job.out* die Standardausgabe und in *job.err* die Fehlerausgabe des über Condor gestarteten Programms geschrieben. In *job.log* werden dem Job zugehörige Ereignisse festgehalten, wie z.B. der Beginn, der Abbruch oder die erfolgreiche Beendigung der Ausführung oder das Zurückhalten oder Freigeben des Jobs. Sobald ein solches Ereignis eintritt, schreibt der Condor-Dämon neue Zeilen in diese Datei, beginnend mit der Nummer des Ereignisses. Dies macht es möglich, durch die Überwachung dieser Datei Statuswechsel eines Jobs, wie die erfolgreiche Beendigung, mitzubekommen.

Eine ausführliche Beschreibung dieser Dateien und der möglichen Ereignisse ist in der Condor-Anleitung und [Con] zu finden.

Teil III.

Lösung

8. Meta-Modell-unterstütztes Optimierungsverfahren

Den Kern dieser Arbeit stellt das im Folgenden vorgestellte Verfahren dar. In Abschnitt 8.1 wird der Einsatzbereich des Verfahrens genau dargestellt. Das Verfahren selbst wird in Abschnitt 8.2 erläutert.

8.1. Problemstellung

Das Ziel ist die Optimierung eines simulierten optischen Nachrichtennetzes; oder abstrakter, einer unbekannten und teuer auszuwertenden Funktion. Die Simulation eines solchen Netzes sei dafür als gegeben anzunehmen, so dass als Abstraktion eine Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$ betrachtet werden kann, die von Parametern des Netzes auf die o.B.d.A. zu minimierende Güte des Systems abbildet. Da die Funktion *teuer auszuwerten* ist, sollen bei der Optimierung möglichst wenige Auswertungen gemacht werden.

Es gibt jedoch einen Unterschied zwischen realen und simulierten optischen Nachrichtennetzen, der hierfür relevant ist. Bei der Simulation können die Parameter beliebig genau gewählt werden. Ein reales System unterliegt jedoch physikalisch bedingten Schwankungen, so dass für die Parameter nur ein Bereich angegeben werden kann, in dem diese liegen werden. Dies hat Konsequenzen auf die Bewertung einer Parameterkombination. Liegt z.B. ein optimaler Wert so nahe an einem relativ schlechten Wert, dass durch die zu erwartende Schwankung eines realen Systems dieser schlechte Wert angenommen werden kann, ist das Optimum (für ein System mit diesen Schwankungen) von wenig Wert. Das bedeutet also, dass nicht ein einzelner optimaler Wert gesucht wird (wie dies in Kapitel 3 definiert wurde), sondern ein optimaler Bereich. Naheliegende Kriterien für die Bewertung eines solchen Bereiches wären der Mittelwert aller Werte innerhalb des Bereiches oder das Maximum (bzw. Minimum, falls der Gütwert maximiert werden soll). Ist es nötig, eine bestimmte Performanz des optischen Systems zuzusichern, bestimmt das Maximum der Bewertungsfunktion innerhalb des gewählten Bereiches was zugesichert werden kann. Deshalb wird das Maximum für die folgende Definition des gesuchten Bereiches herangezogen.

Da die Schwankungen der Parameter voneinander nicht abhängen, sollte die Ausdehnung des gesuchten Bereiches für jeden Parameter einzeln gewählt werden. D.h. es ist ein Bereich von Interesse, der in jeder Dimension (d.h. für jeden Parameter) durch eine obere und eine untere Schranke begrenzt wird, die unabhängig von den anderen Parametern sind.

Definition 8.1.1 Eine hyperquaderförmige Teilmenge des d -dimensionalen Parameterraums P ist eine durch die d -dimensionalen Vektoren \vec{u} und \vec{o} festgelegte Menge

$$L(\vec{u}, \vec{o}) = \{p \in P \mid \forall i : (u_i \leq p_i \leq o_i)\}, \quad (8.1)$$

wobei die Elemente von \vec{u} die unteren Schranken jeder Dimension und die von \vec{o} die oberen Schranken jeder Dimension darstellen.

Definition 8.1.2 Die Menge aller hyperquaderförmigen Teilmengen einer bestimmten, durch den Vektor \vec{s} festgelegten, Ausdehnung hat die Gestalt

$$\mathcal{L}(\vec{s}) = \{L(\vec{u}, \vec{o}) \mid \forall i : (o_i - u_o = s_i)\}, \quad (8.2)$$

wobei die Elemente von \vec{s} die Differenz zwischen der oberen und unteren Schranke einer Dimension angeben.

$\mathcal{L}(\vec{s})$ ist also die Menge aller Teilmengen des Parameterraumes mit der gewünschten, durch \vec{s} beschriebenen, Ausdehnung. Gesucht ist nun die Teilmenge $L \in \mathcal{L}(\vec{s})$, für die der o.B.d.A. maximal annehmbare Gütewert minimiert wird.

Definition 8.1.3 Gegeben sei eine zu optimierende Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$ und für jede Dimension des Definitionsbereiches eine Intervallgröße s_i ($i = 1 \dots n$); also ein Vektor \vec{s} . Gesucht ist eine Teilmenge L des d -dimensionalen Parameterraums P , so dass

$$L = \operatorname{argmin}_{L(\vec{u}, \vec{o}) \in \mathcal{L}(\vec{s})} \left(\max_{p \in L(\vec{u}, \vec{o})} f(p) \right) \quad (8.3)$$

8.2. Lösung

Das im Folgenden beschriebene Verfahren zum Auffinden einer in Definition 8.1.3 beschriebenen Teilmenge des Parameterraums ist eine Heuristik. Um immer das korrekte Ergebnis zu liefern, müssten alle möglichen Teilmengen der entsprechenden Größe überprüft werden. Ziel dieses Verfahrens ist es, ein brauchbares Ergebnis, mit möglichst wenig Funktionsauswertungen zu liefern.

Im Folgenden wird es als Algorithmus beschrieben. Die Eingabe besteht aus der zu optimierenden Funktion f , den Größen s_i des gesuchten Bereiches und einer Initialregion beschrieben durch die Vektoren \vec{o} und \vec{u} (die für jeden Parameter die obere bzw. untere Schranke enthalten), wobei $o_i - u_i \geq s_i$ gelten muss. Die Initialregion ist der Bereich, in dem die Suche nach dem optimalen Bereich begonnen wird. Iterativ werden die Schranken \vec{o} und \vec{u} dieses Bereiches verschoben und deren Abstand verringert, solange bis die Abbruchbedingung $\forall i, 1 \leq i \leq d : (o_i - u_i = s_i)$ gilt. Der so gefundene, durch \vec{o} und \vec{u} beschriebene Bereich bildet die Ausgabe.

Die Verschiebung der Schranken erfolgt in Abhängigkeit eines Modells der Funktion f . Dieses Modell wird in Abhängigkeit des aktuellen, durch \vec{o} und \vec{u} beschriebenen, Bereiches an f angepasst. Zu Beginn eines Schritts wird eine konstante Anzahl an noch nicht ausgewerteten Punkten innerhalb des aktuellen Bereiches ausgewählt. Die Funktion f wird an diesen Punkten ausgewertet und das Modell von f wird mit diesen Funktionswerten aktualisiert und dadurch präzisiert. Auf diesem nun weiter angepassten Modell wird ein optimaler Ort gesucht. Da dies nur auf dem Modell erfolgt, ist keine Funktionsauswertung dafür nötig. Der so gefundene Ort wird als Zentrum des, für den nächsten Schritt zu wählenden, Suchbereiches \vec{o}, \vec{u} gesetzt. Die Ausdehnung des Bereiches wird in jeder Dimension durch Multiplikation mit einem Faktor fac_i ($0 < fac_i < 1$) dabei reduziert. Dieser Vorgang wiederholt sich, bis $\forall i, 1 \leq i \leq d : (o_i - u_i = s_i)$ gilt.

aktualisiereSuchBereich(\vec{v}) legt den Suchbereich um den optimalen Ort \vec{v} herum, wobei der Abstand zwischen \vec{o} und \vec{u} verringert wird.

Für die weiteren in Algorithmus 8.1 benutzten Funktionen sind verschiedene Implementierungen möglich. Für *optimalerOrt*(Modell) wird *Simulated Annealing* (Abschnitt 3.2) eingesetzt, das als *optimalen Ort* das Minimum des Modells liefert.

Algorithm 8.1 Meta-Modell basiertes Optimierungsverfahren

```

1: Setze  $\vec{o}, \vec{u}$  auf Werte der Initialregion
2: repeat
3:    $\vec{d} = \text{abtastpunkteInRegion}(\vec{o}, \vec{u})$ 
4:    $\vec{y} = f(\vec{d})$ 
5:   Modell = aktualisiereModellMitNeuenWerten(Modell,  $\vec{d}, \vec{y}$ )
6:    $\vec{v} = \text{optimalerOrt}(\text{Modell})$ 
7:    $\vec{o}, \vec{u} = \text{aktualisiereSuchBereich}(\vec{v})$ 
8: until  $\forall i : o_i - u_i = s_i$ 

```

Algorithm 8.2 aktualisiereSuchBereich(\vec{v})

```

1:  $t_i = o_i - u_i$  (für alle  $i$ )
2:  $t_i = t_i \cdot fac_i$  (für alle  $i$ )
3: if  $t_i < s_i$  then
4:    $t_i = s_i$ 
5: end if
6:  $\vec{o} = \vec{v} + \vec{t}/2$ 
7:  $\vec{u} = \vec{v} - \vec{t}/2$ 

```

$\text{abtastpunkteInRegion}(\vec{o}, \vec{u})$ liefert eine festgelegte Anzahl an Abtastpunkten, die alle innerhalb des durch \vec{o} und \vec{u} gegebenen Bereiches liegen. Dazu werden die beiden vorgestellten Verfahren *Latinhypercube-Design* (LHD, Abschnitt 4.1) und *Maximin Latinhypercube-Design* (Maximin LHD, Abschnitt 4.2) eingesetzt. Die Anzahl der Abtastpunkte bleibt ein zu evaluierender Parameter des Verfahrens.

Als Schätzer bzw. Modell der zu optimierenden Funktion können ebenfalls verschiedene Verfahren eingesetzt werden. Die vorgestellten Interpolationsverfahren *Shepard-Methode* (Abschnitt 5.2) und *Hardy-Multiquadrik* (Abschnitt 5.3) werden hier eingesetzt und in Teil IV evaluiert.

Ebenfalls ausschlaggebende Parameter sind die Faktoren fac_i , mit denen in jedem Schritt die Größe des Suchfensters multipliziert wird.

Tabelle 8.1 fasst die Parameter des Verfahrens zusammen, die in Teil IV evaluiert werden.

Tabelle 8.1: Parameter des Verfahrens

Parameter	Beschreibung
Anzahl Abtastpunkte pro Schritt	Die Funktion $abtastpunkteInRegion(\vec{\sigma}, \vec{u})$ liefert eine gewisse Anzahl an zu evaluierenden Punkten. Je mehr dies sind, umso genauer schätzt der Funktionsschätzer die unbekannte Funktion, jedoch müssen auch mehr Funktionsauswertungen vorgenommen werden. Dieser Parameter sollte also möglichst klein gewählt werden, um die Zahl der gesamten Funktionsauswertungen gering zu halten. Jedoch ist damit zu rechnen, dass die Wahrscheinlichkeit, dass echte Optimum zu finden, mit einer kleineren Anzahl an Abtastpunkten sinkt.
fac_i	In jedem Schritt wird das Suchfenster verkleinert, indem die Größe des Fensters in der i -ten Dimension mit fac_i multipliziert wird. Es gilt $0 < fac_i < 1$. Da das Verfahren abbricht, wenn das Suchfenster die gewünschte Größe angenommen hat, steigt die Anzahl der Schritte mit wachsendem fac_i . Es gilt $\lim_{fac_i \rightarrow 1} AnzahlSchritte \rightarrow \infty$ sowie $\lim_{fac_i \rightarrow 0} AnzahlSchritte \rightarrow 1$.
Funktionsschätzer	Als Schätzer werden die vorgestellten Verfahren <i>Hardy Multiquadrik</i> und die <i>Shepard Methode</i> verwendet.

9. Entwickelte Software

Im Zuge dieser Arbeit wurde eine Software erstellt, deren hauptsächlicher Zweck die Implementierung des in Kapitel 8.2 vorgestellten Verfahrens ist. Die in Abschnitt 8.1 getroffene Annahme, es kann von einer zu optimierenden Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ausgegangen werden, muss allerdings erst noch erfüllt werden. Zunächst liegen nur mit der Software PHOTOSS erstellte Simulationen vor, die ebenfalls mit PHOTOSS simuliert werden müssen. Die Lücke zu diesem Abstraktionsniveau einer auswertbaren Funktion muss erst geschlossen werden.

Die hier vorgestellte Software hat weiterhin die Aufgabe, die mehrdimensionalen Funktionen und Schätzer und den Ablauf der Optimierung zu visualisieren. Sie soll als Werkzeug bei der Planung von optischen Nachrichtennetzen dienen, indem nach dem Aufbau eines solchen Netzes mit PHOTOSS der Parameterraum untersucht und angezeigt werden kann.

Ausserdem wurden die in Teil IV vorgestellten Ergebnisse mit dieser Software erstellt. Sie diente während der Entstehung dieser Arbeit dazu, potentiell relevante Verfahren zu testen und zu evaluieren.

Abbildung 9.1 zeigt ein Bildschirmfoto der Benutzerschnittstelle der Software.

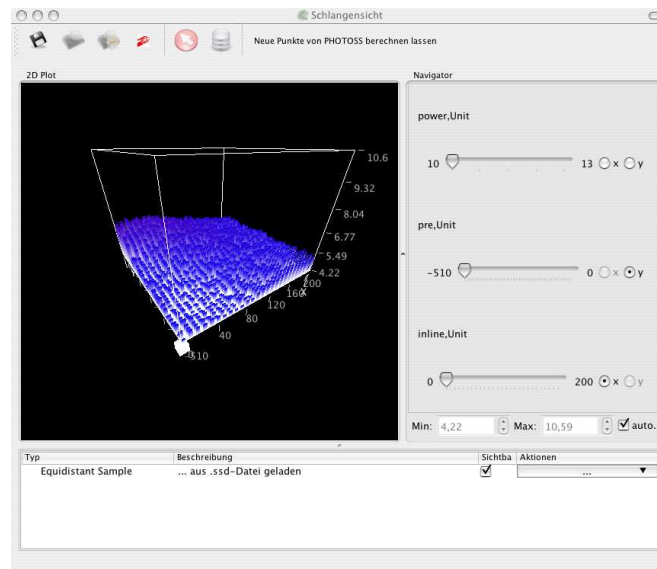


Abbildung 9.1: Bildschirmfoto der erstellten Software.

9.1. Anwendungsfälle

Im Folgenden wird beschrieben, welche Tätigkeiten durch die Software ermöglicht werden.

Laden von PHOToss-Ergebnissen

Der bisherige Ansatz zur Untersuchung des Parameterraums einer Simulation optischer Netze bestand zunächst in der Berechnung vieler äquidistant verteilter Parameterkombinationen. Die Ergebnisse solcher *Parametervariationen* (Abschnitt 2.4.1) werden von PHOToss in einer *Comma Separated Values* Datei (CSV-Datei) abgelegt. Solche Dateien sollen eingelesen werden können. Dadurch wird es möglich, die implementierten Verfahren auf bestehende, bekannte Daten anzuwenden und die bestehenden Daten zu visualisieren.

Speichern von zuvor geladenen CSV-Dateien

Für sehr viele Abtastungen belegt eine entsprechende CSV-Datei sehr viel Speicher und für eine nicht optimierte Laderoutine dauert das Einlesen relativ lange. Aus diesen Gründen soll die Software eine einmal aus einer CSV-Datei eingelesene äquidistant abgetastete Funktion in einem anderen, spezialisierten und effizienteren Format speichern und daraus wieder einlesen können.

Visualisierung von und Navigation durch mehrdimensionale Funktionen

Sowohl die importierten, abgetasteten Funktionen, als auch die berechneten Funktionsschätzer hängen meistens von mehreren (zumindest mehr als zwei) Variablen ab, so dass diese nicht auf einfache Weise komplett dargestellt werden können. Die Software soll es ermöglichen, Teile dieser zuvor geladenen oder erzeugten Funktionen zu visualisieren und auf simple Weise den angezeigten Teil durch den Parameterraum zu navigieren.

Anwenden der grundlegenden Verfahren auf beliebige Daten

Die in Teil II vorgestellten Verfahren sollen einzeln, d.h. nicht als Teil des Optimierungsprozesses, auf vorliegenden Daten angewendet werden. Aus einer eingelesenen äquidistant abgetasteten Funktion sollen mit Hilfe der LHD- und MaximinLHD-Verfahren Punkte ausgesucht werden können, die zusammen als neue Funktion repräsentiert werden, die nur für diese Punkte Funktionswerte aufweist. Dadurch kann die realistischere Situation simuliert werden, in der nur wenige Punkte tatsächlich ausgewertet wurden. Es soll weiterhin möglich sein, wahlweise nur noch diese Teil-Funktion, und nicht die Ausgangsfunktion, anzuzeigen.

Ausgehend von solchen Teil-Funktionen soll es möglich sein, Funktionsschätzer zu generieren, die die Werte der Teil-Funktion als Stützstellen verwenden. Da die Ausgangsfunktion noch vorliegt, soll diese parallel zu dem Funktionsschätzer angezeigt werden können, so dass es möglich ist, einen Eindruck davon zu gewinnen, wie gut der erzeugte Funktionsschätzer die Ausgangsfunktion schätzt und wo er evtl. versagt.

Auswerten von PHOToss-Simulationen

Es soll möglich sein, eine PHOToss-Simulationsdatei und die Parameter der darin enthaltenen Simulation anzugeben, so dass daraufhin eine Funktion angelegt wird, die zu Beginn für keinen Punkt des (durch die angegebenen Parameter bestimmten) Definitionsbereichs einen Funktionswert bereit hält, die jedoch an einer beliebigen Stelle ausgewertet werden kann. Der Benutzer soll die Möglichkeit haben, manuell Punkte auszuwählen, die daraufhin ausgewertet werden. Die Auswertung erfolgt unter Verwendung von PHOToss, welches als eigenständiger Prozess gestartet

wird, um die in der angegebenen Datei enthaltene Simulation mit den entsprechenden Parametern zu simulieren.

Wie bei Teil-Funktionen von importierten äquidistant abgetasteten Funktionen können, nachdem mehrere Punkte ausgewertet wurden, Funktionsschätzer mit diesen Punkten als Stützstellen erzeugt und angezeigt werden.

Optimierung

Als der wesentliche Nutzen der Software soll es möglich sein, von einer Funktion, ob komplett bekannt oder nur als PHOTOSS-Simulationsdatei vorliegend, mit Hilfe von dem in Kapitel 8 vorgestellten Verfahren den optimalen Parameterbereich suchen zu lassen. Dazu muss der Benutzer zunächst eine Funktion anlegen, indem er entweder eine zuvor äquidistant abgetastete Funktion einlesen lässt oder eine PHOTOSS-Simulationsdatei angibt. Anschließend bestimmt er die Größe des gesuchten Bereiches (die von den Schwankungen der Parameter abhängt) und legt die Parameter des Optimierungsverfahrens fest (siehe Abschnitt 8.2).

Während die Suche läuft, soll es dem Benutzer stets möglich sein, die bisherigen Auswertungen sowie den aktuellen Status des Funktionsschätzers einsehen zu können. Wird das Verfahren auf eine bereits bekannte Funktion angewendet, soll es möglich sein, diese parallel dazu anzuzeigen. In diesem Fall soll es ausserdem möglich sein, durch Betrachtung aller möglichen Bereiche der gewünschten Größe den absolut optimalen Bereich zu finden und anzuzeigen. Dadurch wird es möglich abzuschätzen, wie verlässlich die Lösung des vorgestellten Verfahrens ist.

9.2. Dateiformate

PHOTOSS speichert die Ergebnisse einer Simulation in einer CSV-Datei, die für jeden Parameter und für jedes Ergebnis eine durch Tabulatoren getrennte Spalte enthält. Bei lediglich einer Simulation enthält die Datei nur eine Zeile. Bei einer *Parametervariation* (Abschnitt 2.4.1) wird jede Parameterkombination mit dem dazu gehörigen Ergebniswert in jeweils einer Zeile abgelegt. Als Kopf der Datei dienen zwei mit % als Kommentar gekennzeichnete Zeilen. Die erste gibt für jede Spalte den Namen des entsprechenden Parameters bzw. des entsprechenden Ergebnisses an, die zweite gibt für Ergebnisse an, welche Komponente der Simulation das entsprechende Ergebnis berechnet hat.

Listing 9.1: Beispiel einer von PHOTOSS erstellten Ergebnisdatei

%power	pre	inline	Eye opening penalty Receiver (Eye Analyzer)
10	-10	0	6.161613807231003
10	-10	8	6.479711139957114
10	-10	16	6.537707010024554
10	-10	24	6.464482109704168
10	-10	32	6.274806677619703
10	-10	40	6.491104542746009
10	-10	48	6.345213457438072
10	-10	56	6.575585624120618
10	-10	64	6.295195222715233
10	-10	72	6.283374135831772
10	-10	80	6.076316478172775
10	-10	88	5.859395098598858

10	-10	96	5.659361797749032
10	-10	104	5.936875836930421
10	-10	112	5.873849057151439
10	-10	120	5.794348413079675
10	-10	128	5.487820808885495
10	-10	136	5.696963657922264
10	-10	144	5.53868204303262
10	-10	152	5.595231855472288
10	-10	160	5.356486110100896
10	-10	168	5.310671323595344
10	-10	176	5.273695168019445
10	-10	184	5.157651754915905
10	-10	192	5.226897359847578
10	-10	200	5.254126464023034

Listing 9.1 zeigt den Anfang einer solchen PHOToss-Ergebnisdatei. Bei der dazugehörigen Simulation wurden die drei Parameter *power*, *pre* und *inline* variiert, um in Abhängigkeit davon das Ergebnis, die *Eye opening penalty* (EOP), von der Komponente mit dem Namen *Receiver* (die ein *Augen Analysator* ist) berechnen zu lassen.

Diese Dateien werden von der Software eingelesen, indem in einem ersten zeilenweisen Durchlauf der Parameterraum bestimmt wird. D.h. es wird für jeden Parameter Minimum, Maximum und Schrittweite¹ bestimmt, so dass es möglich ist, den nötigen Speicherplatz zu berechnen und zu allokalieren. Dann wird die Datei ein zweites Mal zeilenweise durchlaufen, wobei die Ergebniswerte in den zuvor allokierten Speicherbereich geschrieben werden. Der Speicherbereich wird als mehrdimensionales Feld interpretiert, so dass eine Parameterkombination mit einer Speicherstelle korrespondiert, an der der Ergebniswert abgelegt wird. Listing 9.2 zeigt die (gekürzte) Methode, mit der der Index des Feldes berechnet wird, der einer gegebenen Parameterkombination zugeordnet ist. Die Variablen *m_variables* und *m_domains* wurden zuvor, bei dem ersten Durchlauf der Datei beschrieben. *m_variables* enthält die Namen der Parameter (Variablen) und *m_domains* für jeden Parameter sein Minimum, Maximum und seine Schrittweite.

Listing 9.2: Berechnung des Index des mehrdimensionalen Feldes (C++)

```
1 inline unsigned int EquidistantSample :: offset_for(vector<float> key){
2   unsigned int pos = 0;
3   for(unsigned int i = 0; i < m_variables.size(); ++i){
4     Domain d = m_domains[i];
5     pos += int(int((key[i] - d.min)/d.step + 0.5) * mult(i));
6   }
7   return pos;
8 }

1 inline unsigned int EquidistantSample :: mult(unsigned int n){
2     if(n<=0) return 1;
3     else {
4         Domain d = m_domains[n-1];
5         return d.size * mult(n-1);
6     }
7 }
```

¹ Die Schrittweite wird für einen Parameter als konstant vorausgesetzt. D.h. es werden nur äquidistant abgetastete Funktionen betrachtet.

Auf eine auf diese Weise eingelesene und repräsentierte Funktion kann schnell zugegriffen werden. Jedoch dauert das Einlesen, insbesondere das Parsen der CSV-Datei (bei großen Dateien) relativ lange. Dies liegt daran, dass immer nur wenige Bytes eingelesen und interpretiert werden. Liegt die Funktion jedoch in Form eines zusammenhängenden Speicherbereiches vor, kann dieser in einem Stück in eine Datei geschrieben, und wieder heraus gelesen werden. Um diese Daten interpretieren zu können, müssen die Daten über den Parameterraum mitgeliefert werden; also die Variablen *m_variables* und *m_domains* ebenfalls in die Datei geschrieben werden.

Das zum Speichern von äquidistant abgetasteten mehrdimensionalen Funktionen entwickelte Dateiformat hat die folgende Gestalt. Das erste Byte gibt die Anzahl der Parameter *p* an. Als nächstes folgen $p \cdot 4 \text{ Words}^2$, wovon jeweils 4 *Words* eine Datenstruktur der Form

```
1 struct Domain{
2     float min, max, step;
3     unsigned int size;
4 };
```

repräsentieren. Dies sind die nötigen Daten eines Parameters; Minimum, Maximum und Schrittweite. Es gilt $size = \frac{max-min}{step}$. Die erste Struktur beschreibt den ersten Parameter, die zweite den zweiten usw. Danach folgen *p* Zeichenketten, die mit einem Zeilenende begrenzt werden; also *p* Zeilen, die jeweils den Namen eines Parameters enthalten. Die erste Zeile enthält den Namen des ersten Parameters usw. Darauf folgt das mehrdimensionale Feld, das die Funktionswerte enthält. Die Größe davon ist durch den zuvor gelesenen Parameterraum bestimmt: $\prod_{i=1}^p Domain.size_i \text{ Words}$, wobei ein Funktionswert als *float* repräsentiert wird, also ein *Word* belegt. Die Byte-Reihenfolge ist *Little-Endian* für alle Werte.

Listing 9.3 zeigt den Quellcode, mit dem eine solche Datei geschrieben wird und Listing 9.4 das lesende Gegenstück. Dabei ist anzumerken, dass geprüft werden muss, welche Byte-Reihenfolge der Rechner benutzt, auf dem das Programm ausgeführt wird. Für den Fall von *Big-Endian* müssen vor dem Schreiben und nach dem Lesen die Bytes vertauscht werden.

Listing 9.3: Schreib-Routine für das eingeführte Dateiformat (C++)

```
1 void EquidistantSample::writeFile(string filename){
2     ofstream file(filename.c_str(), ios::binary);
3     char variableCount = m_domains.size();
4     file.write(&variableCount, 1);
5     vector<Domain> domains = m_domains;
6     for(unsigned int i=0; i<domains.size(); ++i)
7         domains[i].convertIfBigEndian();
8     for(unsigned int i=0; i<domains.size(); ++i)
9         file.write(reinterpret_cast<const char*>(&domains[i]), sizeof(Domain));
10    for(unsigned int i=0; i<m_variables.size(); ++i)
11        file << m_variables[i] << std::endl;
12
13    if(EndianHelper::getInstance().isBigEndianSystem())
14        for(unsigned int i=0; i<m_size; ++i)
15            m_data[i] = EndianHelper::getInstance().swapIfBigEndianSystem(m_data[i]);
16
17    file.write(reinterpret_cast<const char*>(m_data), m_size * sizeof(float));
18}
```

² gemeint sind *Words* einer 32-Bit Maschine, also entspricht ein *Word* 4 Byte.

```
19 if (EndianHelper::getInstance().isBigEndianSystem())
20     for(unsigned int i=0;i<m_size;++i)
21         m_data[i] = EndianHelper::getInstance().swapIfBigEndianSystem(m_data[i]);
22 }
```

Listing 9.4: Lese-Routine für das eingeführte Dateiformat (C++)

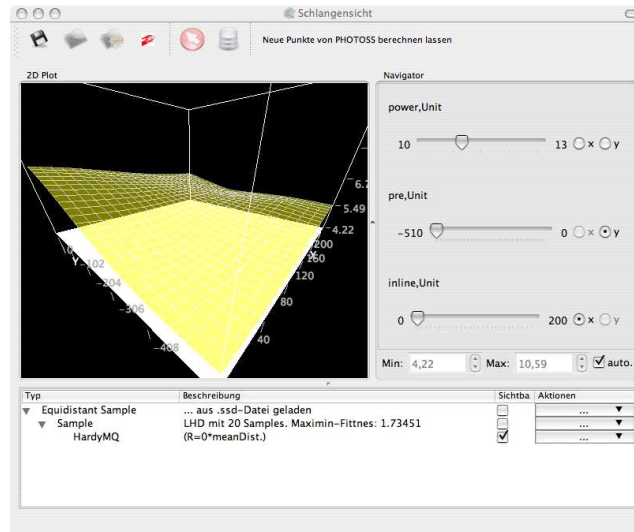
```
1 void EquidistantSample::readFile(string filename){
2     ifstream file(filename.c_str(), ios::binary);
3     vector<string> variables;
4     vector<Domain> domains;
5     char variableCount;
6     file.read(&variableCount, 1);
7     domains.resize(static_cast<int>(variableCount));
8     for(unsigned int i=0;i<domains.size();++i)
9         file.read(reinterpret_cast<char*>(&domains[i]), sizeof(Domain));
10    for(unsigned int i=0;i<domains.size();++i)
11        domains[i].convertIfBigEndian();
12    variables.resize(static_cast<int>(variableCount));
13    for(unsigned int i=0;i<variables.size();++i)
14        getline(file, variables[i]);
15
16    initialize(variables, domains);
17
18    file.read(reinterpret_cast<char*>(m_data), m_size * sizeof(float));
19
20    if (EndianHelper::getInstance().isBigEndianSystem())
21        for(unsigned int i=0;i<m_size;++i)
22            m_data[i] = EndianHelper::getInstance().swapIfBigEndianSystem(m_data[i]);
23
24    m_min = m_max = m_data[0];
25    for(unsigned long i=0; i < m_size; ++i){
26        if(m_data[i] < m_min) m_min = m_data[i];
27        if(m_data[i] > m_max) m_max = m_data[i];
28    }
29 }
```

9.3. Visualisierung und Navigation

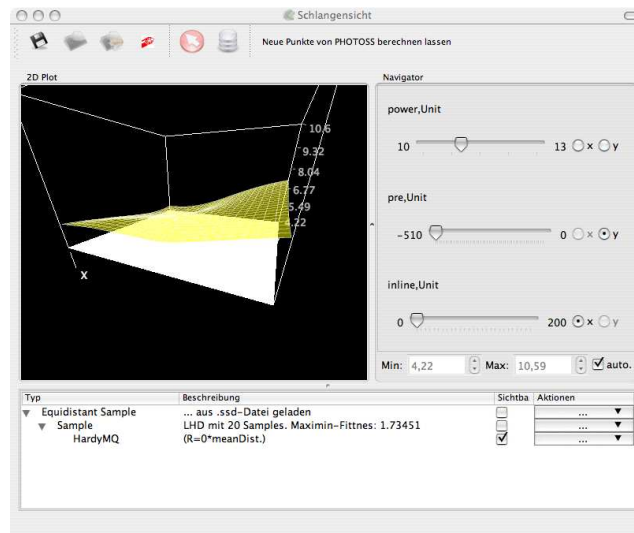
Die Software bietet eine Visualisierungsmöglichkeit für mehrdimensionale Funktionen. Es werden die Funktionen jedoch nicht komplett, in einem Bild visualisiert. Stattdessen werden Schnittebenen des Definitionsbereiches der Funktion ausgewählt und die Funktionswerte dieses Bereiches dargestellt.

Eine Schnittebene ist ein zweidimensionaler Teilraum des Definitionsbereiches. Eine Funktion, die auf einem zweidimensionalen Raum definiert ist, kann mit einem bekannten Verfahren visualisiert werden. Dazu wird der Funktionswert als dritte Dimension interpretiert, so dass sich eine (zweidimensionale) Fläche in einem dreidimensionalen Raum ergibt. Dies ist anschaulich begreifbar und es kann mit Hilfe von (perspektivischer) Projektion eine solche dreidimensionale Szene auf ein zweidimensionales Computerdisplay abgebildet werden.

Abbildung 9.2 zeigt eine solche von der Software berechnete Projektion eines zweidimensionalen Teilbereiches einer dreidimensionalen Funktion. In beiden Bildern wird der gleiche Teilbereich der gleichen Funktion aber auf unterschiedliche Punkte des dreidimensionalen Raumes projiziert, also aus unterschiedlichen Blickwinkeln gesehen.



(a) Blickwinkel 1



(b) Blickwinkel 2

Abbildung 9.2: Das linke Teilfenster mit dem Namen *2D Plot* (hauptsächlich schwarz und gelb, in beiden Bildern zu sehen) zeigt eine zweidimensionale Funktion, deren Funktionswert auf der dritten Achse (die Höhe) aufgetragen wird. Bei beiden Bildern ist die gleiche Funktion jedoch aus unterschiedlichen (gegenüber liegenden) Blickwinkeln zu sehen

Die Software bietet dem Benutzer also die Möglichkeit, die Parameter der Projektion dieses dreidimensionalen Raumes zu verändern. Er kann durch klicken und ziehen der Maus die be-

trachtete zweidimensionale (Teil-) Funktion drehen und aus jedem Blickwinkel betrachten und er kann hinein und heraus zoomen.

Es wird jedoch immer nur ein Teil, eine Schnittebene des Definitionsbereiches der eigentlich mehrdimensionalen Funktion angezeigt. Wo diese Schnittebene in dem Definitionsbereich liegt, muss vom Benutzer festgelegt werden. Als Beschränkung gilt dafür, dass die Ebene immer parallel zu zwei der Dimensionen des Definitionsbereiches liegen soll. Dies stellt sicher, dass jede der beiden Achsen des angezeigten Schnitts immer einer Dimension des Definitionsbereiches der Ausgangsfunktion und somit einem variierten Parameter der optischen Simulation entspricht. Der Benutzer kann wählen, welcher Parameter auf die x - und welcher auf die y -Achse abgebildet werden soll. Für alle weiteren Parameter muss ein Wert gewählt werden, an dem die Ebene in dieser Dimension liegt.

Diese Einstellungen werden über das *Navigator*-Fenster (rechts neben dem *2D Plot*-Fenster oder Abbildung 9.3) vorgenommen. Durch anklicken der Radio-Buttons werden die Parameter festgelegt, zu denen die Ebene parallel liegt. Durch verschieben der Slider der übrigen Parameter wird die Ebene in den entsprechenden Dimensionen verschoben. Für jede Konfiguration dieser Steuerelemente existiert also eine zweidimensionale Teilfunktion, die in dem linken Fenster angezeigt wird, wenn der Navigator entsprechend eingestellt ist.

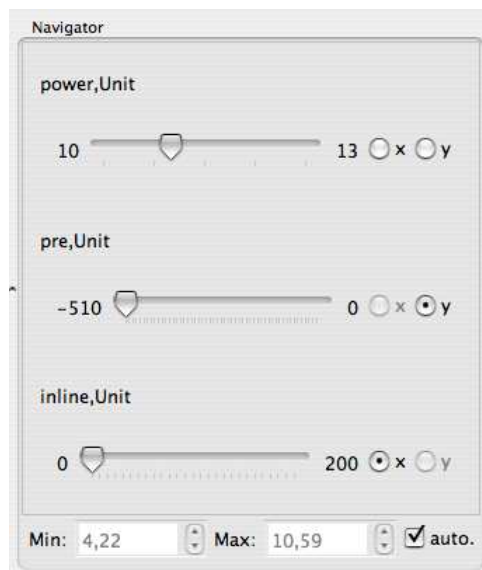


Abbildung 9.3: Navigator-Fenster der Software: hiermit wird festgelegt, welchen Teil der mehrdimensionalen Funktion das Plot-Fenster zeigt. Die Abbildung zeigt eine Beispielkonfiguration anhand einer dreidimensionalen Funktion.

In Abbildung 9.4 sind vier verschiedene Teilbereiche einer dreidimensionalen Funktion zu sehen. Zwei Dimensionen wurden wie beschrieben ausgewählt, um die Schnittebene auszurichten; eine Dimension, die den Ort der Schnittebene angibt, bleibt übrig. Bei den vier Bildern wurde der Ort der Schnittebene in dieser Dimension variiert.

Alle bisher gezeigten Bilder demonstrieren, wie die Software einen Interpolanten bzw. Funktionsschätzer visualisiert. Diese werden dafür nur an einer endlichen Anzahl an Stellen ausgewertet. Da sie jedoch an jeder Stelle schnell berechnet werden können und im Normalfall stetig verlau-

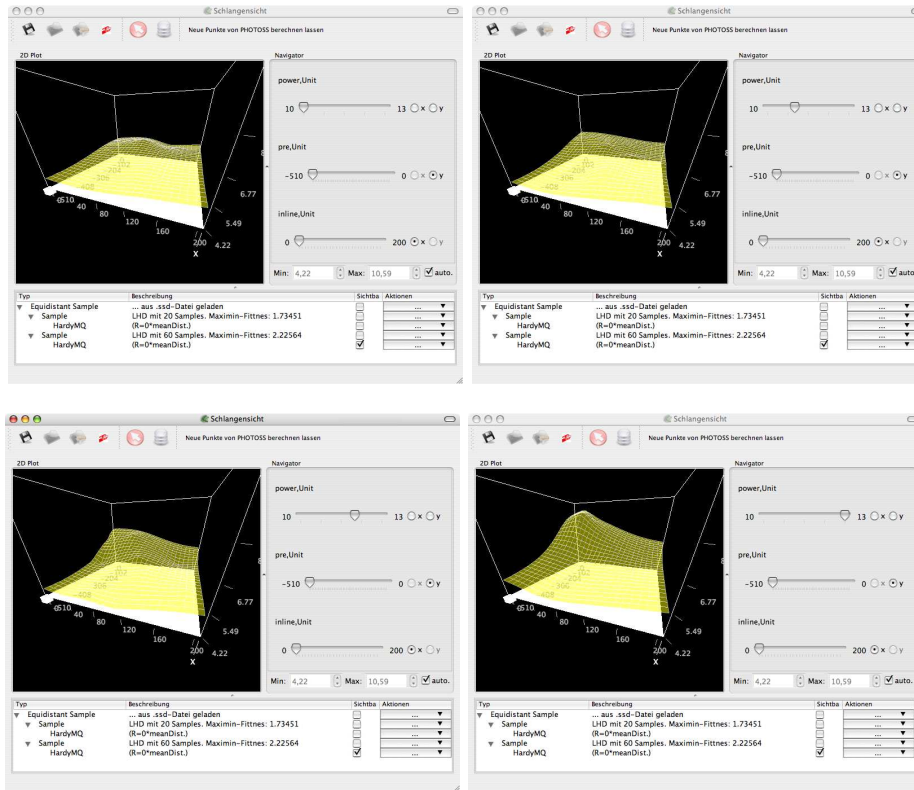


Abbildung 9.4: Mehrere zweidimensionale Schnitte einer dreidimensionalen Funktion an unterschiedlichen Stellen der dritten Dimension.

fen, werden zwischen den ausgewerteten Punkten viereckige Polygone gezeichnet, so dass der Eindruck einer geschlossenen Fläche entsteht.

Äquidistant abgetastete Funktionen werden jedoch anders gezeichnet, um zu verdeutlichen, dass Funktionswerte nur für einzelne Punkte vorliegen. Jeder Funktionswert wird als *Säule* dargestellt, die zu ihren Nachbarn eine Lücke aufweist. Die Farbe der Säulen wird in Abhängigkeit vom Funktionswert zwischen blau und rot interpoliert; das Minimum ist blau, das Maximum rot. (Siehe Abbildung 9.5.)

9.4. Anwendung der grundlegenden Verfahren

Die Software bietet die Möglichkeit, die in Teil II beschriebenen Verfahren auf bestehende Daten anzuwenden. Dies sind im Wesentlichen die beiden Verfahren zur Erstellung von Versuchsplänen, Latinhypercube-Design (Abschnitt 4.1) und Maximin Latinhypercube-Design (Abschnitt 4.2), sowie die drei vorgestellten Funktionsschätzer, die Ausgleichsebene (Abschnitt 5.1), die Shepard-Methode (Abschnitt 5.2) und die Hardy-Multiquadrik (Abschnitt 5.3).

Dazu ist es zunächst erforderlich, einen bestehenden Datensatz, also eine bestehende äquidistant Abgetastete Funktion zu laden, die als Ausgangspunkt aller Anwendungen dient. Die Idee dabei ist, als nächstes einige wenige bekannte Punkte der Funktion auszuwählen, um die Situation zu

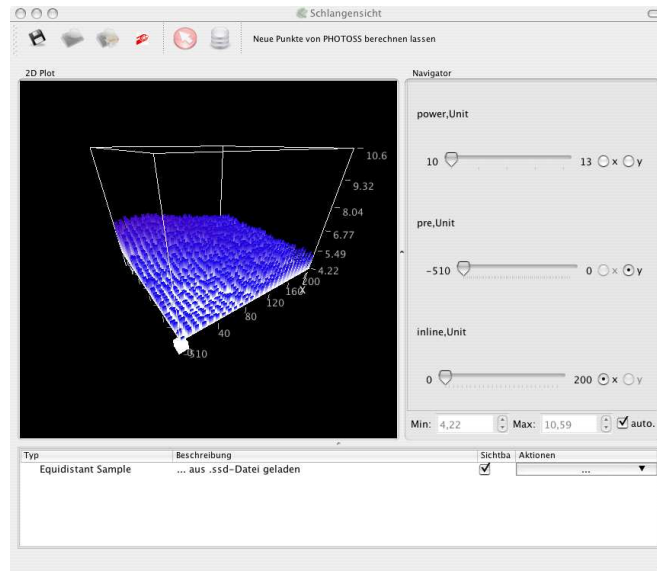


Abbildung 9.5: Visualisierung einer äquidistant abgetasteten Funktion.

simulieren, es wären nur diese Punkte bekannt. Dazu können mit Hilfe der Verfahren LHD und MaximinLHD Versuchspläne erstellt werden, die innerhalb des Definitionsbereiches liegen, und deren Design-Punkte an dem Raster der bekannten äquidistant verteilten Punkte der Funktion ausgerichtet werden. Es muss lediglich die Anzahl der Punkte und das Verfahren (LHD oder MaximinLHD) gewählt werden. Daraufhin wird ein neues Objekt erstellt, das in der Objekt-Liste (das untere der drei Teil-Fenster der Benutzerschnittstelle, Abbildung 9.6), angezeigt wird.

Typ	Beschreibung	Sichtba	Aktionen
▼ Equidistant Sample	... aus .ssd-Datei geladen	<input checked="" type="checkbox"/>	...
Sample	MaximinLHD mit 30 Samples Maximin-Fitness: 1.88717	<input checked="" type="checkbox"/>	...

Abbildung 9.6: Die Objekt-Liste zeigt alle anzeigbaren Objekte. Hier zu sehen, die eingeleseene äquidistant abgetastete Funktion und als Kindobjekt ein Maximin Latinhypercube-Design in dem Definitionsbereich der Funktion.

Die Sichtbarkeit dieser Objekte kann für jedes Objekt einzeln gewählt werden. Die Objekte können gelöscht werden und es können je nach Typ des Objekts verschiedene Aktionen darauf ausgeführt werden. Eine äquidistant abgetastete Funktion erlaubt es, solche Versuchspläne innerhalb des Definitionsbereiches anzulegen, wobei die Funktionswerte an den Design-Punkten ebenfalls in dem neuen Objekt gespeichert sind, so dass das neue Objekt eine Menge von Punkten mit Funktionswerten darstellt. Gezeichnet werden diese Punkte als durchsichtige grüne Säulen, die etwas größer sind, als die Säulen einer äquidistant abgetasteten Funktion, also deren Säulen einhüllen. (Abb. 9.7.)

Solche *Samples* erlauben es, Funktionsschätzer zu erzeugen, die die Punkte des Samples als Stützstellen verwenden. Je nachdem welches Schätzverfahren gewählt wird, müssen dazu etwaige

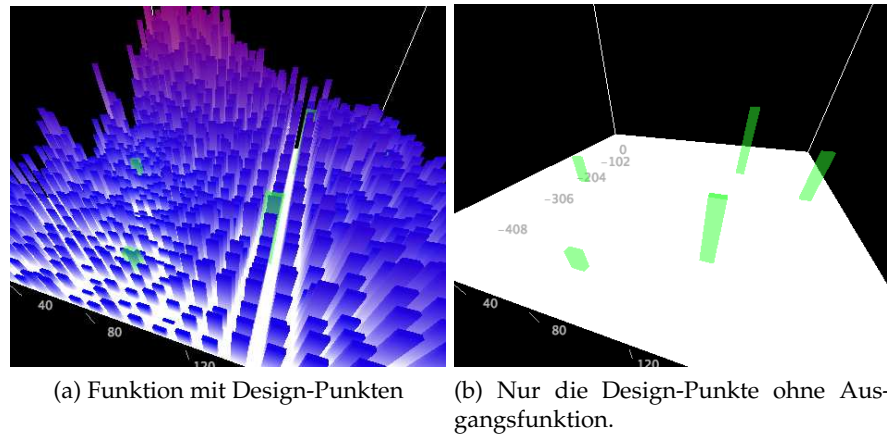


Abbildung 9.7: Es können alle Objekte einzeln sichtbar oder nicht sichtbar geschaltet werden. So kann nach Auswahl einiger weniger Punkte einer Funktion, die Ausgangsfunktion versteckt werden.

Parameter des Verfahrens angegeben werden. Die Funktionsschätzer sind eigenständige Objekte, die angezeigt, versteckt und gelöscht werden können. Abbildung 9.8 zeigt die verschiedenen vorgestellten Schätzverfahren, alle angewandt auf die gleiche Menge von Design-Punkten.

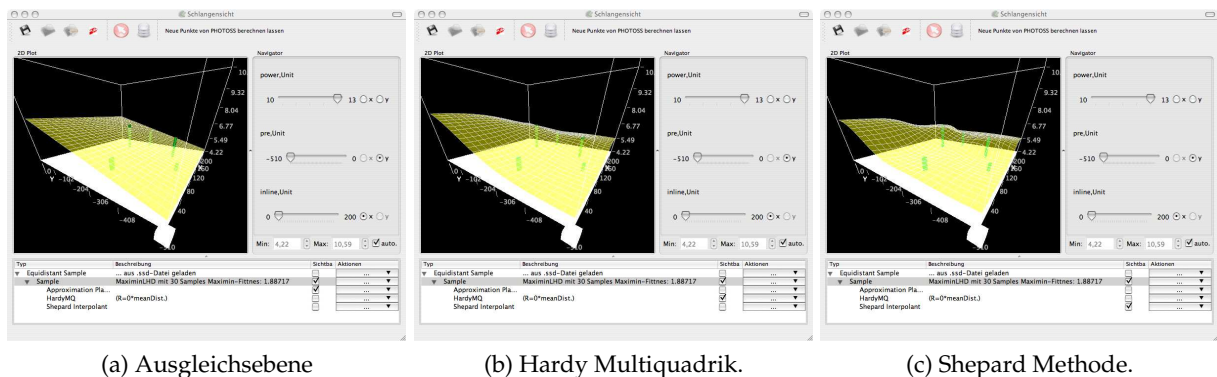


Abbildung 9.8: Verschiedene Funktionsschätzer mit den gleichen Stützstellen.

Um Aussagen über die Güte eines Funktionsschätzers treffen zu können, müsste dieser mit der echten Funktion verglichen werden. Dies kann zunächst optisch getan werden, indem die Ausgangsfunktion und der Funktionsschätzer gleichzeitig angezeigt werden. Abbildung 9.9 zeigt ein solches Szenario. Es ist zu erkennen, wie der Funktionsschätzer den Trend der Funktion nachbildet, jedoch besonders in dem Bereich der relativ hohen Funktionswerte größere Fehler macht. Die Erklärung dafür ist auch ersichtlich: es fehlen Stützstellen in dieser Region.

Um die Güte statistisch zu bewerten, bieten die Funktionsschätzer-Objekte die Aktion *RMS berechnen* an. Dabei wird der *root mean square error*, also die Wurzel des Mittelwerts der Fehlerquadrate berechnet. Jeder Funktionswert der zugrunde liegenden äquidistant abgetasteten Funktion

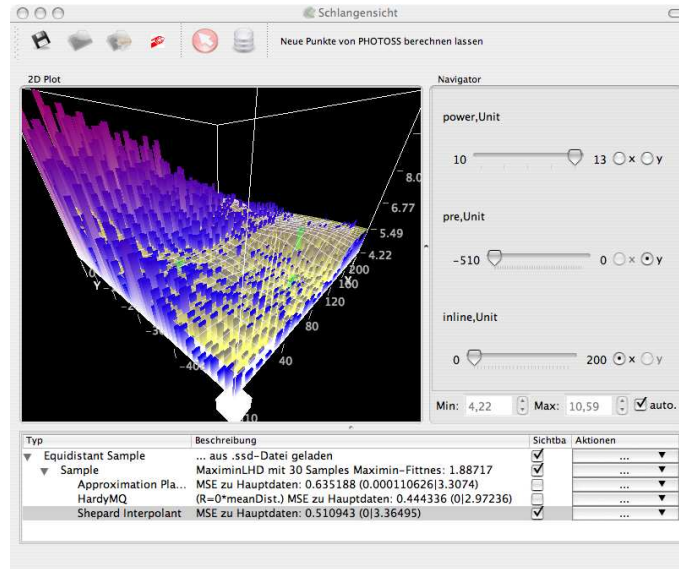


Abbildung 9.9: Sowohl die abgetastete Funktion als auch der Funktionsschätzer mit seinen Stützstellen ist in dieser Abbildung zu sehen. Man erhält einen optischen Eindruck von der Güte des Schätzers.

wird dabei mit dem Funktionswert des Schätzers an der entsprechenden Stelle verglichen. Von den Quadraten dieser Differenzen wird das arithmetische Mittel gebildet und davon die Wurzel gezogen. Dies entspricht der Gleichung

$$RMS(f, F, X) = \sqrt{\frac{1}{|X|} \sum_{x \in X} (f(x) - F(x))^2} \quad (9.1)$$

wobei f die abgetastete Funktion, F der Funktionsschätzer und X die Menge der Punkte ist, an denen abgetastet wurde.

Das Ergebnis wird in der Objekt-Liste in der Spalte *Beschreibung* angezeigt. In dem Beispiel in Abbildung 9.9 ist zu erkennen, dass die Hardy Multiquadrik einen kleineren Fehler als die Ausgleichsebene oder der Shepard Interpolant macht. Die Zahlen in Klammern sind der kleinste und der größte gemessene Fehler.

9.5. Condor- und Photoss-Anbindung

Bisher war der Ausgangspunkt aller Anwendungen der Software eine bereits äquidistant abgetastete Funktion. Die wesentliche Motivation zur Entwicklung der Software ist aber, eine solche zu optimierende Funktion eben nicht mehr äquidistant (und damit an vielen Stellen) abtasten zu müssen. Vielmehr soll die Software, aufbauend auf den vorgestellten Verfahren, die auszuwertenden Punkte online bestimmen und mit den Ergebnissen der Auswertung weiterarbeiten. Dazu ist es erforderlich, dass die Software Photoss-Prozesse starten und die Ergebnisse deren Berechnung einlesen kann.

Die Software bietet dem Benutzer also die Möglichkeit, eine noch unbekannte Funktion in Form einer Photoss-Simulationsdatei zu laden. Dazu muss lediglich die PHO-Datei, ein Arbeitsverzeichnis und die Parameter der Simulation angegeben werden. Dies erfolgt über den in Abbildung 9.10 gezeigten Dialog.

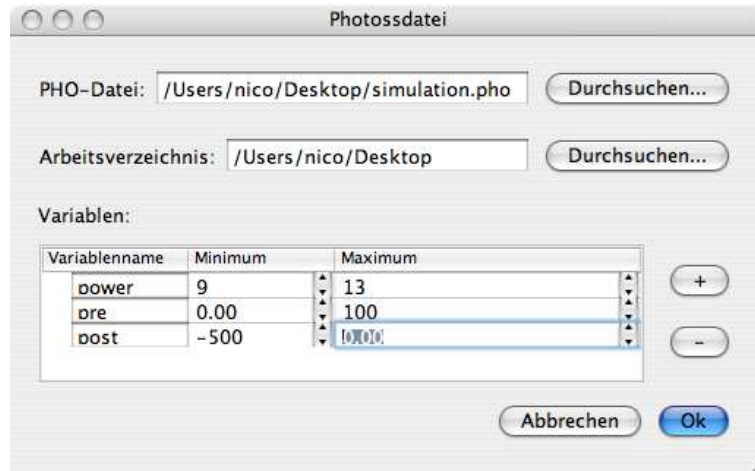


Abbildung 9.10: Dialog zum erstellen einer unbekannten Funktion anhand einer vorhandenen PHO-Datei.

Navigation und Visualisierung erfolgen dann wie bereits beschrieben. Jedoch ist zu Beginn noch kein Funktionswert bekannt, so dass ein leeres Plot-Fenster zu sehen ist (Abbildung 9.11). Es können jedoch beliebige Punkte (Parameterkombinationen) gewählt werden, die daraufhin berechnet und angezeigt werden.

Da eventuell viele Punkte gleichzeitig ausgewertet werden sollen, also viele Photoss-Prozesse gestartet werden müssen, verwendet die Software Condor (Kapitel 7.1), um die Berechnungen auf mehrere Rechner zu verteilen. Dazu ist es im Wesentlichen nötig, eine Condor-Jobdatei zu erzeugen, die nötigen Dateien in ein Arbeitsverzeichnis zu kopieren, den Job an Condor zu übergeben und nach seiner Beendigung die von Photoss produzierte Ergebnisdatei einzulesen.

Erzeugen der Jobdatei

In einer Condor-Jobdatei wird im Wesentlichen festgelegt, welche ausführbare Datei von Condor ausgeführt werden soll, mit welchen Argumenten das Programm gestartet werden soll und welche Dateien zusammen mit der Programmdatei übertragen werden müssen. Die auszuführende Datei ist in diesem Fall immer die *Photoss.exe*. Dateien die benötigt werden sind alle zu Photoss gehörigen DLL- und Ressourcen-Dateien sowie die zu simulierende PHO-Datei. Die an Photoss zu übergebenden Argumente sind zum einen die Zeichenkette „-r <Name der PHO-Datei>“, was Photoss veranlasst, die Berechnung der angegebenen Simulation automatisch zu starten, sowie eine Zeichenkette „-p <Parameter Name>=<Wert>“ für jeden Parameter, um die Parameterausprägungen Photoss mitzuteilen. Eine automatisch generierte Jobdatei zeigt Listing 9.5. Dort fällt ein weiteres Argument auf. -m none sorgt dafür, dass Photoss die *Matlab* Unterstützung deaktiviert. Dies ist nötig, da in dem für diese Arbeit verwendeten Rechner-Pool nicht auf jedem Rechner die Software *Matlab* installiert ist, die mit Photoss zusammen verwendet werden kann,

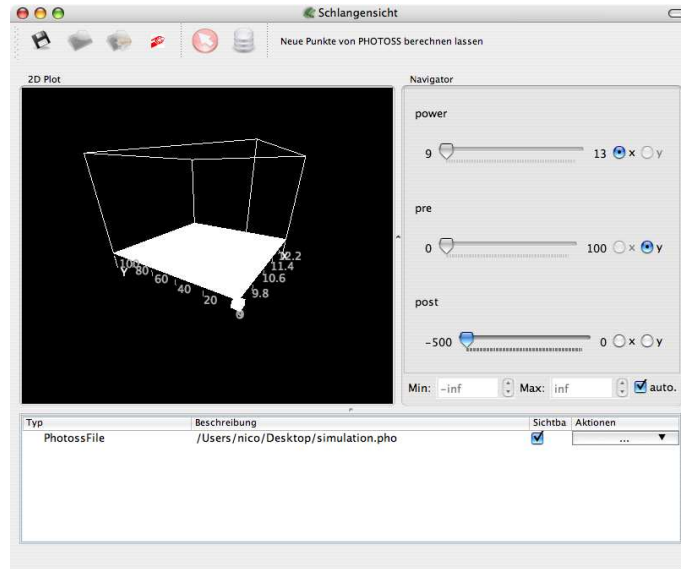


Abbildung 9.11: Leeres Plot-Fenster bei unbekannter Funktion.

um Berechnungen durchzuführen.

Listing 9.5: Von der Software erzeugte Condor-Jobdatei.

```

1 EXECUTABLE = D:/ users /Nico/PHOTOSS/ photoss_release /Photoss .exe
2 UNIVERSE = vanilla
3 SHOULD_TRANSFER_FILES = YES
4 WHEN_TO_TRANSFER_OUTPUT = ON_EXIT
5 TRANSFER_INPUT_FILES = D:/ users /Nico/PHOTOSS/ photoss_release / ApplicationBasics_dll.dll , \
6 D:/ users /Nico/PHOTOSS/ photoss_release /DefaultModelTree.txt , \
7 D:/ users /Nico/PHOTOSS/ photoss_release /DefaultUserTree.txt , \
8 D:/ users /Nico/PHOTOSS/ photoss_release /condor_exec.exe.manifest , \
9 D:/ users /Nico/PHOTOSS/ photoss_release /ToolkitPro1113vc80.dll , \
10 D:/ users /Nico/PHOTOSS/ photoss_release /component_dlls/dev_kit.dll , \
11 D:/ users /Nico/PHOTOSS/ photoss_release /matlabR12_dll.dll , \
12 D:/ users /Nico/PHOTOSS/ photoss_release /matlabR13_dll.dll , \
13 D:/ users /Nico/PHOTOSS/ photoss_release /matlabR14_dll.dll , \
14 D:/ users /Nico/PHOTOSS/ photoss_release /matlabR2006a_dll.dll , \
15 D:/ users /Nico/PHOTOSS/ photoss_release /msvc80.dll , \
16 D:/ users /Nico/PHOTOSS/ photoss_release /msvcr80.dll , \
17 D:/ users /Nico/PHOTOSS/ photoss_release /photoss_dll.dll , \
18 D:/ users /Nico/PHOTOSS/ photoss_release /visualizer_dll.dll , \
19 D:/ users /Nico/PHOTOSS/ photoss_release /QtCore4.dll , \
20 D:/ users /Nico/PHOTOSS/ photoss_release /QtGui4.dll , \
21 D:/ users /Nico/DA/ condor.test/condor.test.pho
22 ARGUMENTS = -m none -r condor.test.pho -p bw_el=50 -p bw_opt=50 -p d=0
23 ERROR = err.txt
24 OUTPUT = out.txt
25 LOG = log.txt
26 QUEUE

```

Der durch so eine Jobdatei beschriebene Job wird daraufhin an Condor übergeben, indem das Programm *condor_submit* von der Software gestartet wird, wobei der Pfad zu der Jobdatei als Argument übergeben wird.

Überwachung des Status des Jobs

Condor legt nach erfolgreicher Annahme eines neuen Jobs die Dateien für die Standardausgabe, Fehlerausgabe und das Log an. In der oben angegebenen Jobdatei wurden die Namen für diese Dateien in den Zeilen 24, 25 und 26 festgelegt. In der Log-Datei fügt Condor bei jedem den Job betreffenden Ereignis einen Absatz hinzu. Darin steht in einer menschenlesbaren Form, was mit dem Job geschehen ist. Die ersten Zeichen der ersten Zeile eines solchen Absatzes bilden eine dreistellige Zahl, die das Ereignis kodiert. Z.B. gibt *005* an, dass der Job beendet, also erfolgreich durchgeführt wurde. Listing 9.6 zeigt eine solche von Condor produzierte Log-Datei.

Listing 9.6: Condor-Logdatei.

```

1 000 (280.000.000) 03/27 20:19:42 Job submitted from host:
2      <129.217.188.159:1050>
3 ...
4 022 (280.000.000) 03/27 20:19:47 Job disconnected, attempting to reconnect
5      Socket between submit and execute hosts closed unexpectedly
6      Trying to reconnect to vm1@GABOR.hft.e-technik.uni-dortmund.de
7      <129.217.188.174:1060>
8 ...
9 024 (280.000.000) 03/27 20:20:25 Job reconnection failed
10     Job not found at execution machine
11     Can not reconnect to vm1@GABOR.hft.e-technik.uni-dortmund.de,
12     rescheduling job
13 ...
14 001 (280.000.000) 03/27 20:20:35 Job executing on host:
15     <129.217.188.174:1060>
16 ...
17 005 (280.000.000) 03/27 20:20:40 Job terminated.
18     (1) Normal termination (return value 0)
19         Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
20         Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
21         Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
22         Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage
23     92704 - Run Bytes Sent By Job
24     21885820 - Run Bytes Received By Job
25     92704 - Total Bytes Sent By Job
26     21885820 - Total Bytes Received By Job
27 ...

```

Damit nun die Software feststellen kann, wann die Berechnung beendet wurde, überwacht sie die Log-Datei eines jeden Jobs und liest immer ganze, neu hinzugekommene Absätze, die ein Ereignis repräsentieren. Sobald das Ereignis *005 - Job terminated* aus der Datei gelesen wurde, wird die Log-Datei nicht mehr beachtet und es werden die Ergebnisse eingelesen.

Einlesen der PHOToss-Ergebnisse

PHOToss schreibt seine Ergebnisse, wie bereits vorgestellt (Abschnitt 9.2, Listing 9.1), in eine neu angelegte CSV-Datei. Durch die Zeilen 3 und 4 in Listing 9.5 werden alle Dateien, die von PHOToss angelegt werden, von Condor auf den Rechner kopiert, von dem aus der Auftrag aufgegeben wurde. PHOToss benennt diese Datei nach der berechneten PHO-Datei „<Name der PHO-Datei ohne Endung>_ParameterVariationResults.txt“. Nachdem Condor also die erfolgreiche Ausführung des Jobs meldet, erwartet die Software eine entsprechende Datei in dem Arbeitsverzeichnis und liest diese ein.

Anders als in Listing 9.1 enthält diese Datei jedoch nur eine Datenzeile, da alle Parameter auf einen Wert festgelegt wurden. (Siehe Listing 9.7)

Listing 9.7: Von PHOToss erzeugte Ergebnisdatei.

```
1 %bw_opt,ghz      d,ps/nm bw_el,GHz      Eye opening penalty
2 %               EyeAnalyzer
3 50              0        50        0.1265921155781947
```

Dieser Funktionswert wird dem Sample hinzugefügt und in dem Plot-Fenster angezeigt.

9.6. Optimierungsmodus

Die Software kennt zwei Modi, in denen Sie betrieben werden kann. Diese können über zwei Knöpfe der Toolbar umgeschaltet werden. Alle bisher vorgestellte Funktionen sind Teil des *Datenmodus*. Die Hauptaufgabe der Software wird aber durch den *Optimierungsmodus* erfüllt, durch den das in Kapitel 8 vorgestellte Verfahren implementiert wird.

Um in den Optimierungsmodus wechseln zu können, muss eine zu optimierende Funktion geladen und ausgewählt werden. Dies kann eine äquidistant abgetastete Funktion oder eine noch unbekannte Funktion in Form einer PHO-Datei sein. Es muss festgelegt werden, in welchem Bereich des (unendlichen großen) Definitionsbereiches die Suche erfolgen soll. Des Weiteren muss die Ausdehnung des gesuchten optimalen Bereiches angegeben werden. Beides ermöglicht der in Abbildung 9.12 gezeigte Dialog.

Nachdem dieser Dialog ausgefüllt wurde, wechselt die Software in den Optimierungsmodus, wobei das Benutzerinterface wie in Abbildung 9.13 erscheint. Die Visualisierung und Navigation funktioniert weiterhin wie vorgestellt. Die Optimierung beginnt erst nach Benutzung des *Optimierung starten*-Knopfes.

Während der Optimierung werden Objekte angelegt, die in der Liste, in der unteren rechten Ecke des Hauptfensters, angezeigt werden. Für jede Iteration der Optimierungsschleife wird ein Sample der neu ausgewerteten Punkte und, als Kind-Objekt, der Funktionsschätzer für diese Iteration angelegt. Diese Objekte werden im Plot-Fenster angezeigt, wenn sie in der Liste vom Benutzer markiert werden. Der aktuelle Suchbereich wird zusammen mit dem Sample der neu ausgewerteten Punkte als rote Fläche im Plot-Fenster angezeigt³.

Es können die Funktionsschätzer zu jeder Iteration betrachtet werden. Dabei wird das per Simulated Annealing gefundene Minimum des Schätzer durch eine weiße Säule markiert. (Siehe Abbildung 9.14.)

³ Der Suchbereich ist ein Hyperquader in dem mehrdimensionalen Definitionsbereich, dessen zweidimensionale Schnitte immer ein Rechteck bilden.

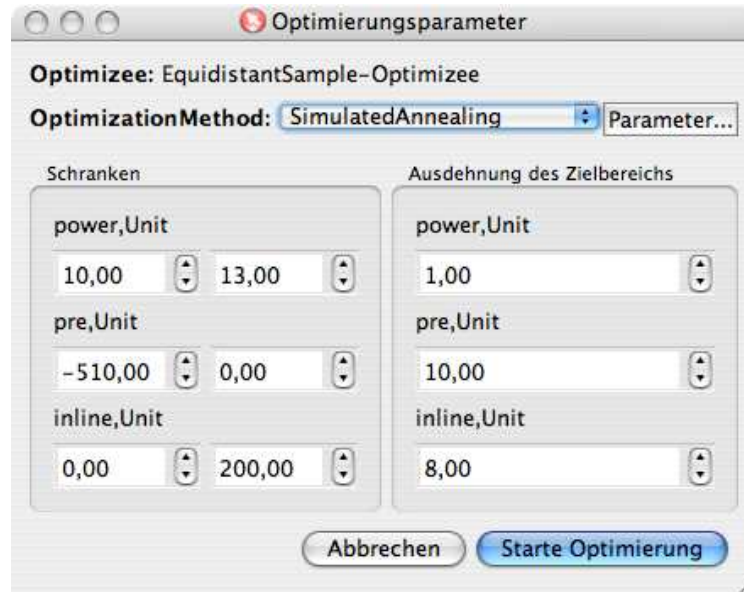


Abbildung 9.12: Dialog zur Eingabe der für die Optimierung nötigen Parameter. Dies ist der Teil des Definitionsbereiches, in dem gesucht werden soll und die Ausdehnung des gesuchten Optimums.

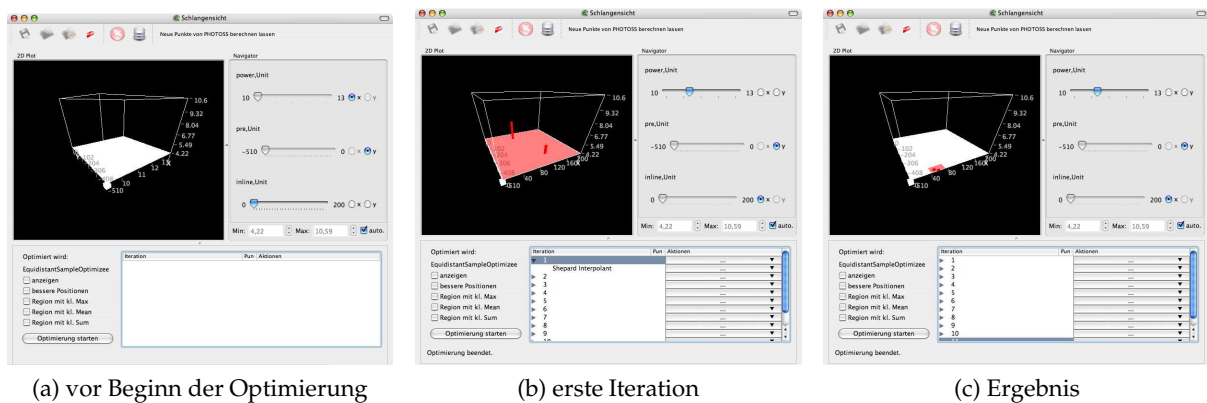


Abbildung 9.13: Das Benutzerinterface des Optimierungsmodus. Der Suchbereich (rote Fläche) erstreckt sich zu Beginn über den gesamten relevanten Bereich, wird aber in jeder Iteration verkleinert und verschoben. Die ausgewerteten Punkte werden als rote Säulen dargestellt.

Das Ergebnis der Optimierung ist der Suchbereich nach der letzten Iteration. Dieser hat die zuvor eingegebene Ausdehnung. Abbildung 9.13(c) zeigt das Ergebnis dieses Laufs. Ohne die optimierte Funktion zu kennen, lässt sich wenig über die Güte dieses Ergebnisses aussagen. Wurde jedoch eine bekannte, äquidistant abgetastete Funktion optimiert, sind die Knöpfe links neben der Objekt-Liste benutzbar. D.h. es ist möglich, die bekannte Funktion sichtbar zu machen, um das Ergebnis oder auch die Funktionsschätzer damit vergleichen zu können. Darüber hinaus ist es bei

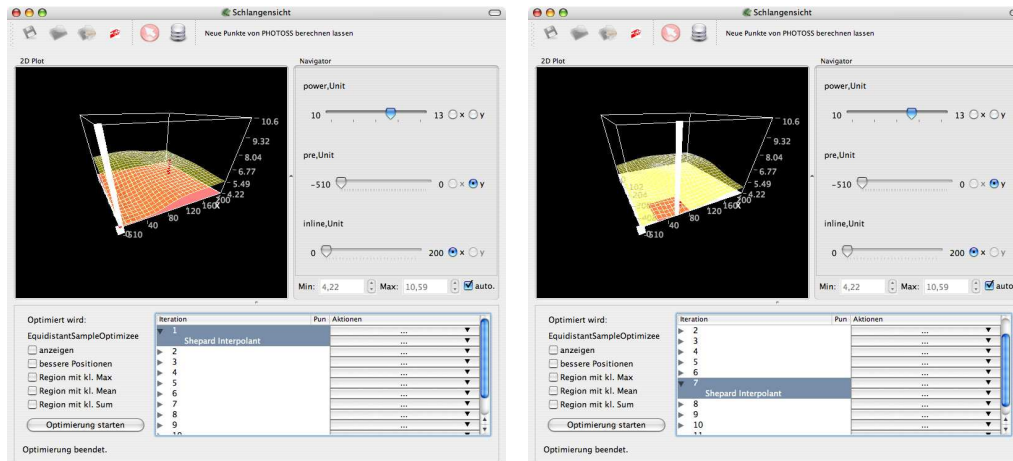


Abbildung 9.14: Ansicht der Funktionsschätzer im Optimierungsmodus. Die weiße Säule markiert das durch Simulated Annealing ermittelte Minimum des Schätzers.

einer äquidistant abgetasteten Funktion möglich, alle Bereiche der gewünschten Ausdehnung zu betrachten und für jeden den maximalen Funktionswert, den Mittelwert der Funktionswerte oder die Summe der Funktionswerte zu berechnen. D.h. es kann der Bereich ausgewählt werden, der der optimale unter Berücksichtigung aller äquidistant verteilten Auswertungen ist. Die Software bietet die Möglichkeit, diesen Bereich anzuzeigen. Abbildung 9.15 zeigt dies.

Dieses Beispiel zeigt, dass der durch das Optimierungsverfahren gefundene Bereich nur leicht von dem optimalen abweicht. Um diesen zu finden wurden jedoch nur $5 \cdot 10 = 50$ Funktionsauswertungen benötigt. Bei der äquidistanten Abtastung der Funktion, die nötig ist, um das Optimum auf dem herkömmlichen Weg zu finden, waren $26 \cdot 52 \cdot 4 = 5408$ Funktionsauswertungen nötig.

10. Implementierung

10.1. Benutzte Werkzeuge

Im Folgenden wird darauf eingegangen, welche bestehende Software und Hardware verwendet wurde, um die vorgestellte neue Software zu entwickeln. Es werden einzeln die Programmiersprache, die verwendeten Bibliotheken und die Entwicklungsumgebung(en) angesprochen.

Programmiersprache

Die Software ist in der Sprache C++[Str00] geschrieben. Es existieren Implementierungen dieser Sprache für alle gängigen und relevanten Plattformen. Der Vorteil dieser Sprache ist eine sehr effiziente Ausführung der darin geschriebenen und kompilierten Programme, in Kombination mit der Möglichkeit, diese gut zu strukturieren und objektorientiert zu entwickeln. Nachteilig, verglichen mit neueren, dynamisch typisierten Sprachen wie z.B. Ruby[Mat], Python[vR] oder ECMAScript[EE], fällt auf, dass der Programmierer durch C++ dazu gezwungen wird, tendenziell viel Code zu schreiben, um eine gewisse Funktionalität zu erreichen. Diese höhere Ausdrucksstärke erkaufen sich die genannten Konkurrenten jedoch durch eine geringere Laufzeiteffizienz.

Die Entscheidung, C++ zu verwenden wurde dadurch bekräftigt, dass zunächst das Ziel war, die Software letztendlich in PHOTOS zu integrieren. Da PHOTOS in C++ geschrieben ist, wäre dies erheblich einfacher möglich, wenn die neue Software ebenfalls in dieser Sprache implementiert ist. Im Laufe der Arbeit hat sich jedoch herausgestellt, dass es durchaus möglich und sogar von Vorteil ist, eine eigenständige Software zu entwickeln, die PHOTOS als Kind-Prozess aufruft und die Ergebnisse aus einer Datei einliest. Es wäre also möglich gewesen, eine andere Sprache zu verwenden, da jedoch zu diesem Zeitpunkt schon Teile in C++ implementiert wurden, wurde diese Sprache beibehalten.

Bibliotheken

Die Standard-Implementierung von C++ beinhaltet keine Möglichkeit, Graphische Benutzerschnittstellen zu erzeugen. Dazu kommt die Klassen-Bibliothek Qt¹ [Tro] von Trolltech zum Einsatz. Der wesentliche Vorteil dieser Bibliothek ist, dass der gleiche Quellcode auf den drei momentan gängigen Plattformen (*Windows*, *Linux/Unix/X11*, *MacOSX*) ohne Änderung kompiliert werden kann, wobei für jedes System ein nativ ausführbares Programm entsteht, dessen GUI optisch dem jeweiligen System angepasst ist. Außerdem erfordert es wenig Aufwand mit Qt Software zu entwickeln, da die bereitgestellten Klassen verständlich und flexibel einsetzbar sind. Verglichen mit den Alternativen, wie z.B. *GTK*[gtk] oder *MFC*[Mica], ist sowohl die Dokumentation als auch das (Klassen-/Software-) Design überlegen. Mit Hilfe des Qt-Designers ist es möglich, GUI-Oberflächen per Drag&Drop aufzubauen. Das mitgelieferte Programm *uic* (User

¹ausgesprochen wie engl. *cute* = hübsch

Interface Compiler) übersetzt so entstandene GUI-Beschreibungen in C++ Code, der in von Hand erzeugten Code eingebunden werden kann. Qt ist sowohl unter einer Open-Source- als auch unter einer kommerziellen Lizenz erhältlich.

Zur Erzeugung von Bildern dreidimensionaler Szenen wird *OpenGL*[Gra] verwendet. Die einzige Alternative, bei der auch aktuelle Grafikkarte zur Berechnung verwendet werden kann, ist *DirectX*[Micb] von Microsoft. Dies würde jedoch (ohne nichtstabile Lösungen wie z.B. *WineX*[Tra] mit einzubeziehen) die Ausführung der Software auf Windows beschränken. OpenGL erfüllt alle Anforderungen, die die Software an eine Bibliothek zur Bilderzeugung stellt. Sie ist frei verfügbar, es existieren Grafikkartentreiber, die die OpenGL-Schnittstelle implementieren und es können alle nötigen grafischen Berechnungen damit durchgeführt werden.

Zur Lösung linearer Gleichungen² und dem *kleinste Quadrate Problem*³ wurde die Bibliothek *Lapack* [ABB⁺99] eingesetzt.

Entwicklungsumgebung

Entwickelt wurde sowohl mit einem *Apple iBook G4* unter MacOSX 10.4 als auch mit einem Intel Core 2 Quad basierten Desktop PC mit dem Betriebssystem Windows XP. Unter Windows kam *Visual C++ 2008 Express* von Microsoft zum Einsatz. Apple liefert *XCode* als Entwicklungsumgebung mit. Der Quellcode wurde von *Subversion*[Col] verwaltet, was es einfach gemacht hat, die beiden verwendeten Systeme zu synchronisieren. Durch die Benutzung von Qt ist es problemlos möglich, den gleichen Code auf beiden Systemen zu kompilieren⁴. Es liefert ein Programm namens *qmake* mit, das aus einer Qt-Projektdatei sowohl *Makefiles*, Visual Studio Projektdateien als auch *XCode* Projektdateien erstellt. D.h. es muss nur eine Projektdatei gepflegt werden, aus der alle (für die verschiedenen Plattformen) nötigen Dateien automatisch generiert werden.

10.2. Softwaredesign

Wie heutzutage üblich, wurde die Software objektorientiert entworfen. Zur Strukturierung wurden also die C++-Sprachmittel zur Erzeugung von Klassen und Objekten benutzt.

Die Architektur entspricht dem *Model View Controller* Muster[Ree79]. Wie jedoch für Desktop-Anwendungen üblich, ist die Trennung zwischen View und Controller nicht sauber, bzw. es ist Controller-Code in View-Klassen zu finden. View-Objekte enthalten teilweise Referenzen auf Model-Objekte und rufen Funktionen dieser Objekte direkt auf. Nichtsdestotrotz existieren drei Controller-Klassen, die als *Einzelstück*⁵[GHJV94] implementiert sind (Siehe Abbildung 10.1). Diese Objekte dienen als *Container* für die Model-Objekte und haben die Zuständigkeit über diese.

MainController hat die Aufgaben, *Data*-Objekte zu verwalten und die graphische Oberfläche zu initialisieren. Änderungen an der Struktur der *Data*-Objekte (wie Laden eines neuen Objektes

²nötig zur Berechnung einer Hardy-Multiquadrik

³nötig zur Berechnung der Ebenenparameter einer Ausgleichsebene

⁴Die einzige Stelle des Quellcodes, bei der Rücksicht auf die Architektur der ausführenden Maschine genommen werden muss, ist das Lesen und Schreiben der Dateien, die eine Kopie des Speichers enthalten. Intel 8086 kompatible Prozessoren haben eine *little-endian* Bytereihenfolge, der PowerPC G4 von IBM, der in dem iBook verbaut ist, hat eine *big-endian* Bytereihenfolge. Da in der Datei immer eine *little-endian* Bytereihenfolge verwendet werden soll, müssen die Bytes vor dem Schreiben und nach dem Lesen von einer *big-endian*-Maschine vertauscht werden.

⁵engl. Singleton



aus einer Datei, Hinzufügen eines neuen Objektes als Kind eines bestehenden und Löschen von Objekten) erfolgen ausschließlich über Funktionen dieses Objekts.

Optimizer hat die Aufgabe, den groben Ablauf der Optimierung zu steuern und die dabei erstellten *Data*-Objekte zu verwalten. Wie bei der Optimierung die Funktion ausgewertet wird, und wie die auszuwertenden Stellen bestimmt werden, ist in *OptimizationMethod* und den Unterklassen von *Optimizee* definiert.

Automator dient der Ausführung von Prozeduren auf dem Domain Model, ohne eine Benutzerinteraktion oder die View-Klassen dazu zu benötigen. Dabei können Prozeduren für eine Reihe von Parametern ausgeführt werden. Die Prozeduren berechnen Ergebnisse, die in CSV-Dateien gespeichert werden. Dies wird für die Evaluation der Verfahren benötigt, da dazu Berechnungen viele Male ausgeführt werden müssen, was von Hand über die graphische Schnittstelle, zu viel Zeit in Anspruch nehmen würde.

Abhängig von den Kommandozeilenparametern, wird in der *main*-Funktion entweder die *init* und *showMainWindow* Methode des *MainControllers*, oder eine Methode von *Automator* aufgerufen.

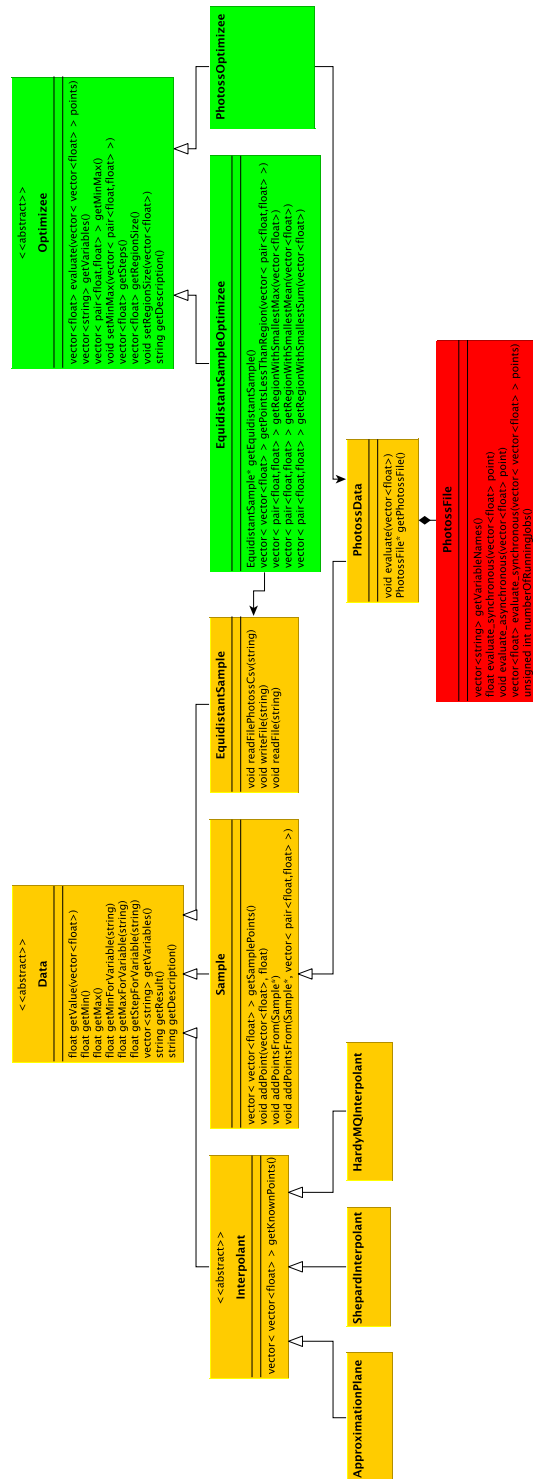
Domain Model

Der wichtigste Teil des Domain Models sind die Klassen, die die mehrdimensionalen Funktionen repräsentieren. Dies sind alle Unterklassen der Klasse *Data* (Abbildung 10.2, die orange gefärbten Klassen). *Data* implementiert nur wenige der deklarierten Funktionen. Die wesentlichste Funktion, die von den Unterklassen zu implementieren ist, ist *float getValue(vector<float>)*, die den Funktionswert für die übergebene Parameterkombination zurückgibt.

Die drei direkten Unterklassen spiegeln die Aufteilung in Funktionen wider, die an jeder Stelle ausgewertet werden können (*Interpolant*), solche die nur an wenigen Stellen Funktionswerte vorweisen können (*Sample*) und solche die an äquidistant verteilten Stellen Funktionswerte vorweisen können (*EquidistantSample*). Interpolanten haben Stützstellen, die mit der Methode *Interpolant::getKnownPoints()* erfragt werden können. Einem *Sample*-Objekt können neue Punkte hinzugefügt werden (*Sample::addPoint()*). *EquidistantSample*-Objekte können aus Dateien gelesen und in Dateien geschrieben werden (*EquidistantSample::writeFile()/readFile()*).

Die Unterklassen von *Interpolant* repräsentieren die vorgestellten Interpolationsverfahren. *PhotossData*, die Unterklasse von *Sample*, repräsentiert eine Funktion, die durch eine Photoss-Datei gegeben ist. Sie kann zwar durch Ausführung von Photoss an jeder Stelle ausgewertet werden, zu einem Zeitpunkt sind aber immer nur einige wenige Stellen bekannt. Deshalb ist *PhotossData* eine Unterklasse von *Sample*, die jedoch die Methode *void evaluate(vector<float>)* zur Berechnung neuer Punkte mitbringt. Die Klasse *PhotossFile* kapselt alles was dafür nötig ist (das Starten von Photoss über Condor und das nachträgliche Einlesen der Ergebnisse).

Die grünen Klassen in Abbildung 10.2 werden für die Optimierung benötigt. Die Controller-Klasse *Optimizer* benötigt ein *Optimizee*, welches die zu optimierende Funktion repräsentiert. Es gibt die *Optimizee*-Unterklassen *PhotossOptimizee* und *EquidistantSampleOptimizee*, die jeweils eine Referenz auf ein entsprechendes *Data*-Objekt enthalten. *PhotossOptimizee* delegiert *Optimizee::evaluate()* an *PhotossData::evaluate()*. D.h. bei der Optimierung eines *PhotossData* wird bei einer Funktionsauswertung *PhotossData::evaluate()* aufgerufen, und damit Photoss gestartet und die Simulation berechnet. *EquidistantSampleOptimizee* greift bei einer Funktionsauswertung auf das gegebene *EquidistantSample* zurück und liefert den Funktionswert der bekannten Stelle, die der auszuwertenden Stelle am nächsten liegt. Außerdem bietet *EquidistantSampleOptimizee* Funktionen an, mit denen die für dieses *EquidistantSample*-Objekt optimalen Bereiche berechnet werden



Powered by yFiles

Abbildung 10.2: *Data*- und *Optimizee*-Klassen.

können (*getRegionWithSmallestMax()*...).

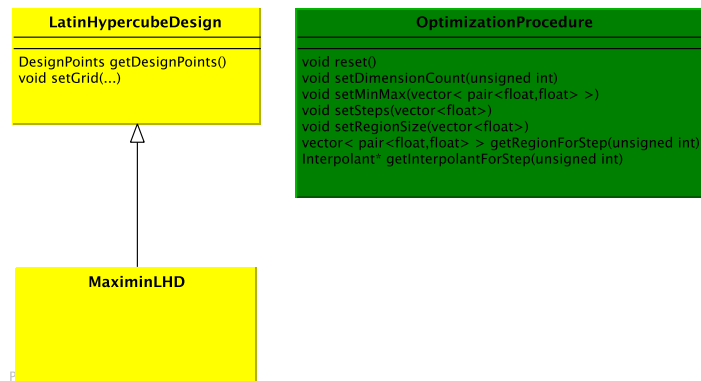


Abbildung 10.3: Klassen zur Erzeugung von Versuchsplänen und *OptimizationProcedure*.

Die in Abbildung 10.3 gelb dargestellten Klassen werden zur Erstellung von Versuchsplänen, also Mengen von auszuwertenden Punkten, benötigt. Sie sind keine Unterklassen von *Data*, da sie keine Funktionswerte enthalten. Sie werden aber verwendet, um *Sample*-Objekte zu erstellen, indem die Funktion an den Design-Punkten ausgewertet wird und das Sample mit diesen Funktionswerten initialisiert wird. Listing 10.1 zeigt Code aus einer View-Klasse, der ein *Sample*-Objekt zu einem zuvor erstellten Latinhypercube-Design erstellt.

Listing 10.1: Code zur Erstellung eines neuen Sample-Objektes anhand eines LHDs. Auszug aus einer View-Klasse.

```

1 LatinHypercubeDesign lhd(m_data, m_designPointsCount);
2 Sample* s = new Sample(
3     m_data->getVariables(),
4     m_data->getMinimaMaximaForAllVariables(),
5     lhd.getDesignPoints(),
6     m_data->getValues(lhd.getDesignPoints()));
7
8 MainController::getInstance().addChildData(m_data, s);
  
```

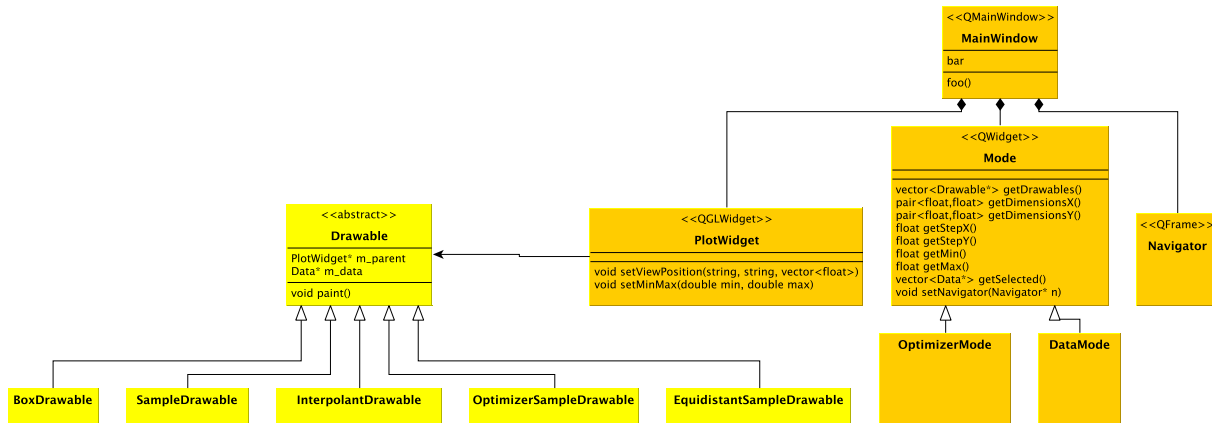
Die Klasse *OptimizationProcedure* repräsentiert das in Kapitel 8 vorgestellte Verfahren. Die Controller-Klasse *Optimizer* benötigt in jeder Iteration der Optimierung eine Menge von auszuwertenden Punkten, die die Methode *OptimizationProcedure::nextPoints(vector<Sample*>)* zurück liefert. Die Software kann leicht um ein weiteres, von dem vorgestellten Verfahren verschiedenes, Verfahren ergänzt werden, indem eine Unterklasse von *OptimizationProcedure* gebildet wird, die diese Methode überschreibt⁶

View

Kernstück der Präsentationsschicht ist die Klasse *MainWindow*. Diese repräsentiert das Hauptfenster mit seinen Menüs und der Werkzeugeiste. Sind *Data*-Objekte vorhanden (nach dem Laden

⁶ bzw. indem ein Interfaces aus *OptimizationProcedure* extrahiert wird und eine weitere Implementierung davon hinzugefügt wird.

einer Funktion z.B.), enthält das *MainWindow*-Objekt jeweils genau ein Objekt der Klassen *PlotWidget*, *Mode* und *Navigator*.



Powered by yFiles

Abbildung 10.4: Präsentationsschicht der Software.

Das *Navigator*-Objekt repräsentiert das Unterfenster, das in dem rechten Drittel des Hauptfensters zu sehen ist und einen Schieberegler für jede Variable der Funktion enthält. Eine benutzerausgelöste Änderung eines Schiebereglers führt dazu, dass das Objekt ein Signal emittiert, das mit dem Slot *setViewPosition()* des *PlotWidget*-Objektes verbunden ist. Dieses erfährt auf diese Weise, dass seine Inhalte aktualisiert werden müssen, da sich die Position der Schnittebene geändert hat.

Das *PlotWidget*-Objekt repräsentiert das Unterfenster auf der linken Seite, das die dreidimensionale Ansicht enthält. *PlotWidget* ist ein *QGLWidget*[Tro]. Dadurch ist es einfach möglich, *OpenGL* zum Zeichnen dieses Fensters zu verwenden. *PlotWidget* legt die *Modelview*- und *Projectionmatrix*[ASW⁺05] anhand der Mausereignisse (Maustastendruck, Mausbewegung) fest, so dass der Benutzer mit der Maus die Perspektive verändern kann. Der (*OpenGL*-) Code zum Zeichnen der eigentlichen Inhalte befindet sich in den Unterklassen von *Drawable*. *PlotWidget* enthält eine Liste von Referenzen auf *Drawable*-Objekte und ruft auf jedem dieser Objekte beim Zeichnen seines Inhaltes die *paint()*-Methode auf. Es existieren verschiedene *Drawable*-Implementierungen, wobei alle außer *BoxDrawable* (die zum Zeichnen des Rahmens und der Achsenbeschriftungen zuständig ist) eine Referenz auf ein Objekt einer *Data*-Unterklasse enthalten, die sie visualisieren.

Erzeugt und verwaltet werden diese *Drawable*-Objekte von dem aktuellen *Mode*-Objekt. Sind Daten geladen, gibt es immer ein solches Objekt, wobei es von dem aktiven Modus abhängt, ob dies ein *OptimizerMode*- oder ein *DataMode*-Objekt ist. *Mode* ist eine Unterklasse von *QWidget*[Tro]. Die *Mode*-Objekte repräsentieren das Unterfenster in der unteren Hälfte des Hauptfensters, das vom Modus abhängig unterschiedliche Inhalte enthält. Über die *vector<Drawable*>getDrawables()* Methode gibt das *Mode*-Objekt an, welche Drawables sichtbar sind.

Teil IV.

Ergebnisse

11. Evaluationsdaten

Bevor in den folgenden Kapiteln die vorgestellten Verfahren evaluiert werden, wird in Abschnitt 11.1 auf die dafür nötigen Datensätze eingegangen. Abschnitt 11.2 gibt die Werte der Parameter der eingesetzten Verfahren an, die bei der Evaluation nicht variiert werden.

11.1. Evaluationsdatensätze

Um berechnen zu können, welchen Fehler ein Funktionsschätzer macht, oder wie nahe das gefundene Optimum an das tatsächliche Optimum herankommt, müsste die zugrunde liegende Funktion bekannt sein. Deshalb wurden zu diesem Zweck zwei typische Simulationen erstellt und für eine große Anzahl äquidistant verteilter Parameterkombinationen berechnet. Dies geschah mit Hilfe von PHOTON und Condor (Kapitel 7.1) und einem Pool von zehn aktuellen, baugleichen Arbeitsplatzrechnern; jeweils bestückt mit einem Intel Core2 Quad Q6600 Prozessor (bei 2,4 GHz) und 2 GiB Arbeitsspeicher. Die Rechner wurden tagsüber teilweise als Arbeitsplatzrechner benutzt, wodurch Condor die Berechnung der Simulation zu dieser Zeit pausiert hat. Die Gesamtlaufzeit einer Simulation betrug etwa ein bis zwei Wochen.

10x10Gbit/s

Es wurde eine Simulation eines WDM-Systems mit zehn Kanälen á 10 Gbit/s Bitrate erstellt. Als Modulationsformat kam NRZ-OOK zum Einsatz. Die variierten Parameter sind zum einen die Leistung des eingespeisten Lichtes bzw. der optischen Verstärker und zum anderen das Dispersionsmanagement der beteiligten Fasern. Abbildung 11.1 zeigt den logischen Aufbau der Simulation.

Die Komponente *Tx* hat zehn Ausgänge, die die zehn Kanäle repräsentieren. Der *Coupler* setzt diese Kanäle so zusammen, dass sie gemeinsam in eine Faser eingespeist werden können. Die eigentliche Übertragungsstrecke wird von der Komponenten *Iterator* repräsentiert. Vor sowie nach der Übertragungsstrecke werden dispersionskompensierende Fasern eingesetzt (*DCF(pre)* und *DCF(post)*). *Iterator* ist eine Komponente, die ein Teilnetz enthält, welches mehrfach hintereinander gelegt simuliert wird. Abbildung 11.2 zeigt den Inhalt dieser Komponente, der fünfmal hintereinander wiederholt wird.

Im Wesentlichen besteht eine Sektion der Übertragungsstrecke aus einem optischen Verstärker und einer Einmodenfaser (*Single Mode Fiber (SMF)*, Komponente *SMF(nonlinear)*) von 100 Kilometern Länge. (Da die Übertragungsstrecke aus fünf Sektionen besteht, beträgt die simulierte Gesamtlänge 500 km.) Die dispersionskompensierende Faser *DCF(linear)* hat eine negative Dispersion, die so gewählt ist, dass sie die akkumulierte Dispersion der vorhergehenden Faser (teilweise) kompensiert. Die akkumulierte Dispersion der Faser *DCF(inline)* ist einer der variierten Parameter. Diese Unterteilung in zwei kompensierende Fasern würde in realen Systemen nicht vorgenommen werden, vereinfacht für die Simulation jedoch die Wahl der Parameter.

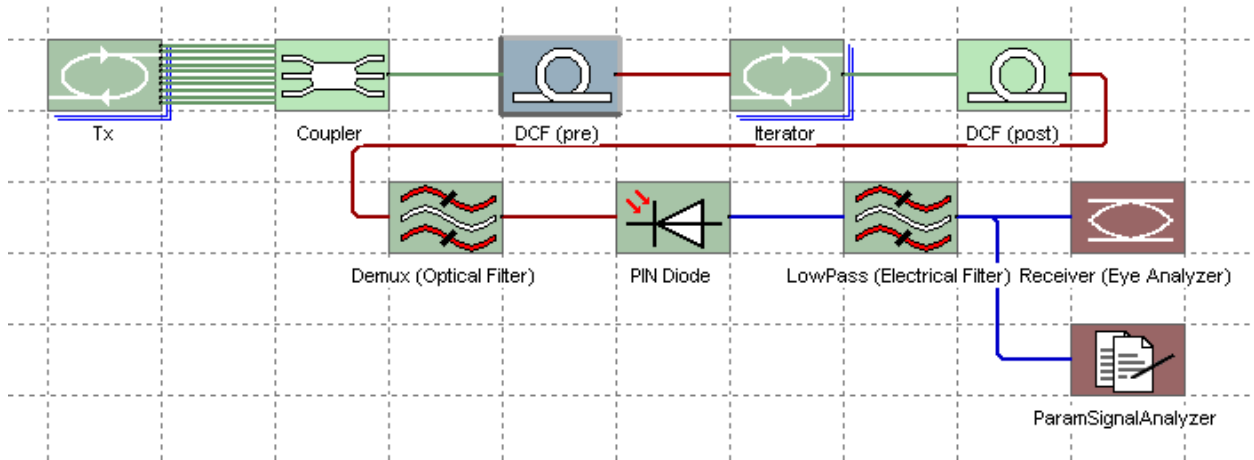
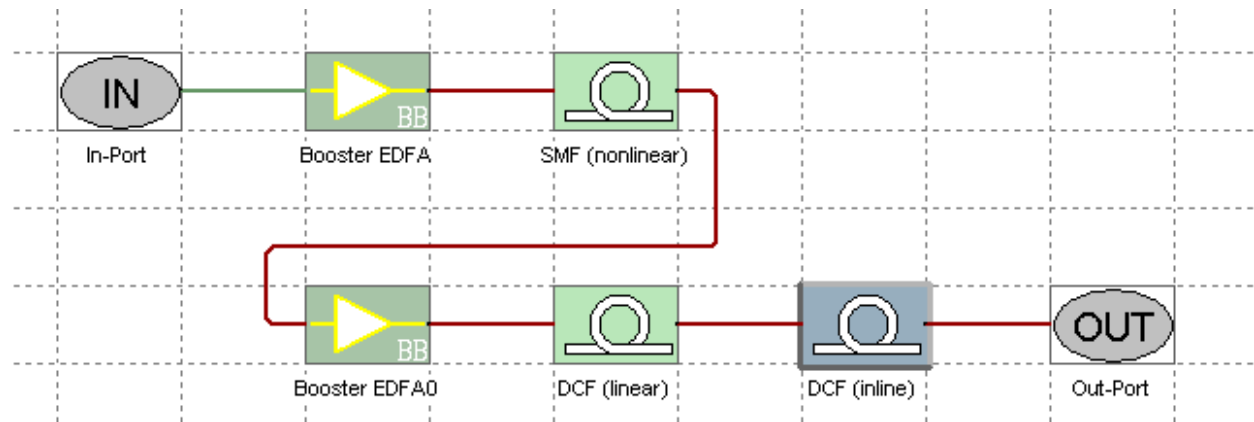


Abbildung 11.1: Versuchsaufbau der 10x10Gbit/s Simulation. PHOTOSS Bildschirmfoto.

Abbildung 11.2: Inhalt der Komponente *Iterator*; die Übertragungsstrecke. PHOTOSS Bildschirmfoto.

Der Empfänger, bestehend aus den restlichen in Abbildung 11.1 gezeigten Komponenten, extrahiert mit einem optischen Filter nur einen (den zentralen) der zehn Kanäle. Mit Hilfe der *PIN Diode* wird das optische Signal in ein elektrisches Signal umgewandelt. Der Augenanalysator (Komponente *Receiver (Eye Analyzer)*) beurteilt die Güte des Signals, durch die Berechnung der *Eye Opening Penalty (EOP)*. Diese ist der zu optimierende Ergebniswert.

Tabelle 11.1 zeigt die Parameter der beteiligten Fasern. Der optische Filter simuliert einen *Gaussianfilter 2. Ordnung* mit einer Bandbreite $\Delta f = 40\text{GHz}$, der elektrische (Tiefpass-) Filter simuliert einen *Besselfilter 10. Ordnung* mit einer Bandbreite von $\Delta f = 14\text{GHz}$.

Variiert werden die Dispersionseigenschaften der Komponenten *DCF(pre)*, *DCF(inline)* und *DCF(post)*, sowie die Leistung, auf die der optische Verstärker innerhalb eines Elementes der Übertragungsstrecke das Signal verstärkt. Die Länge der kompensierenden Fasern wurde auf 1

Tabelle 11.1: Parameter der beteiligten Fasern

Komponentenname	Länge[km]	$\alpha \left[\frac{\text{dB}}{\text{km}} \right]$	$D \left[\frac{\text{ps}}{\text{km} \cdot \text{nm}} \right]$	$S \left[\frac{\text{ps}}{\text{km} \cdot \text{nm}^2} \right]$	$\gamma' \left[\frac{1}{\text{W} \cdot \text{km}} \right]$
SMF (nonlinear)	100	0,2	$\approx 15,62981$	≈ -0.02013	1,54
DCF (linear)	19,96	0,2	$\approx -78,30566$	≈ 0.10067	0
DCF (inline)	1	0,2	variiert	0	0
DCF (pre)	1	0,2	variiert	0	0
DCF (post)	1	0,2	variiert	0	0

km gesetzt. $D \cdot L$ wurde variiert:

- für die Vorkompensation von $-350 \frac{\text{ps}}{\text{nm}}$ bis $0 \frac{\text{ps}}{\text{nm}}$ in Schritten der Weite $25 \frac{\text{ps}}{\text{nm}}$
- für die Unterkompensation der Übertragungsstrecke von $0 \frac{\text{ps}}{\text{nm}}$ bis $165 \frac{\text{ps}}{\text{nm}}$ in Schritten der Weite $15 \frac{\text{ps}}{\text{nm}}$
- für die Nachkompensation von $-420 \frac{\text{ps}}{\text{nm}}$ bis $0 \frac{\text{ps}}{\text{nm}}$ in Schritten der Weite $60 \frac{\text{ps}}{\text{nm}}$.

Die Ausgangsleistung der Verstärker (P_{tot}) wurde auf 9, 10, 11 und 12 dBm gesetzt. Es wurden also 7680 Parameterkombinationen simuliert.

Abbildungen 11.3, 11.4 und 11.5 geben einen Überblick über die Ergebnisse dieser Simulation. Die Bilder wurden mit der im Rahmen dieser Arbeit erstellten Software generiert. Abbildung 11.3 und Abbildung 11.4 wurden mit der gleichen Ausrichtung der Schnittebene (X-Achse: Inline-Kompensation, Y-Achse: Vor-Kompensation), jedoch für unterschiedliche Werte der Verstärkerleistung, erzeugt. Bei Abbildung 11.5 wurde eine andere Ausrichtung der Schnittebene gewählt (X-Achse: Inline-Kompensation, Y-Achse: Nach-Kompensation).

10x107Gbit/s

Als zweite Simulation wurde ein WDM-System mit zehn Kanälen á 107Gbit/s¹ erstellt. Der Aufbau ist analog zu der bereits vorgestellten Simulation, wobei jedoch der sendende Teil anders modelliert wurde. Als Modulationsformat wurde diesmal Duobinär eingesetzt. Da jedoch bei einer höheren Bitrate die Pulse, die die Bits repräsentieren, zeitlich wesentlich dichter beieinander liegen, führt Dispersion schneller zu einem nicht mehr zu erkennenden Signal. Die durch Dispersion induzierte Signaldegradation steigt mit dem Quadrat der Bitrate. Deshalb wird hier die Nachkompensation in Abhängigkeit der beiden anderen Dispersionswerte so berechnet, dass das Produkt $D \cdot L$ für das gesamte System Null wird. Dadurch gibt es einen Freiheitsgrad, also einen zu variierenden Parameter weniger. Dafür können aber die drei übrigen Parameter dichter abgetastet werden.

Die weiteren Faserparameter sind zu denen der 10x10Gbit/s Simulation (Tabelle 11.1) identisch. Die Filtercharakteristiken weichen jedoch ab: der optische Filter simuliert auch hier einen *Gaussfilter 2. Ordnung*, jedoch mit einer Bandbreite von $\Delta f = 100\text{GHz}$, der elektrische (Tiefpass-) Filter ist wieder ein *Besselfilter 10. Ordnung*, hier mit einer Bandbreite von $\Delta f = 14\text{GHz} \cdot 10,7 = 149,8\text{GHz}$

$D \cdot L$ wurde diesmal wie folgt variiert:

- für die Vorkompensation von $-510 \frac{\text{ps}}{\text{nm}}$ bis $0 \frac{\text{ps}}{\text{nm}}$ in Schritten der Weite $10 \frac{\text{ps}}{\text{nm}}$

¹ 7% des Übertragungsvolumens wird für Fehlerkorrektur benötigt.

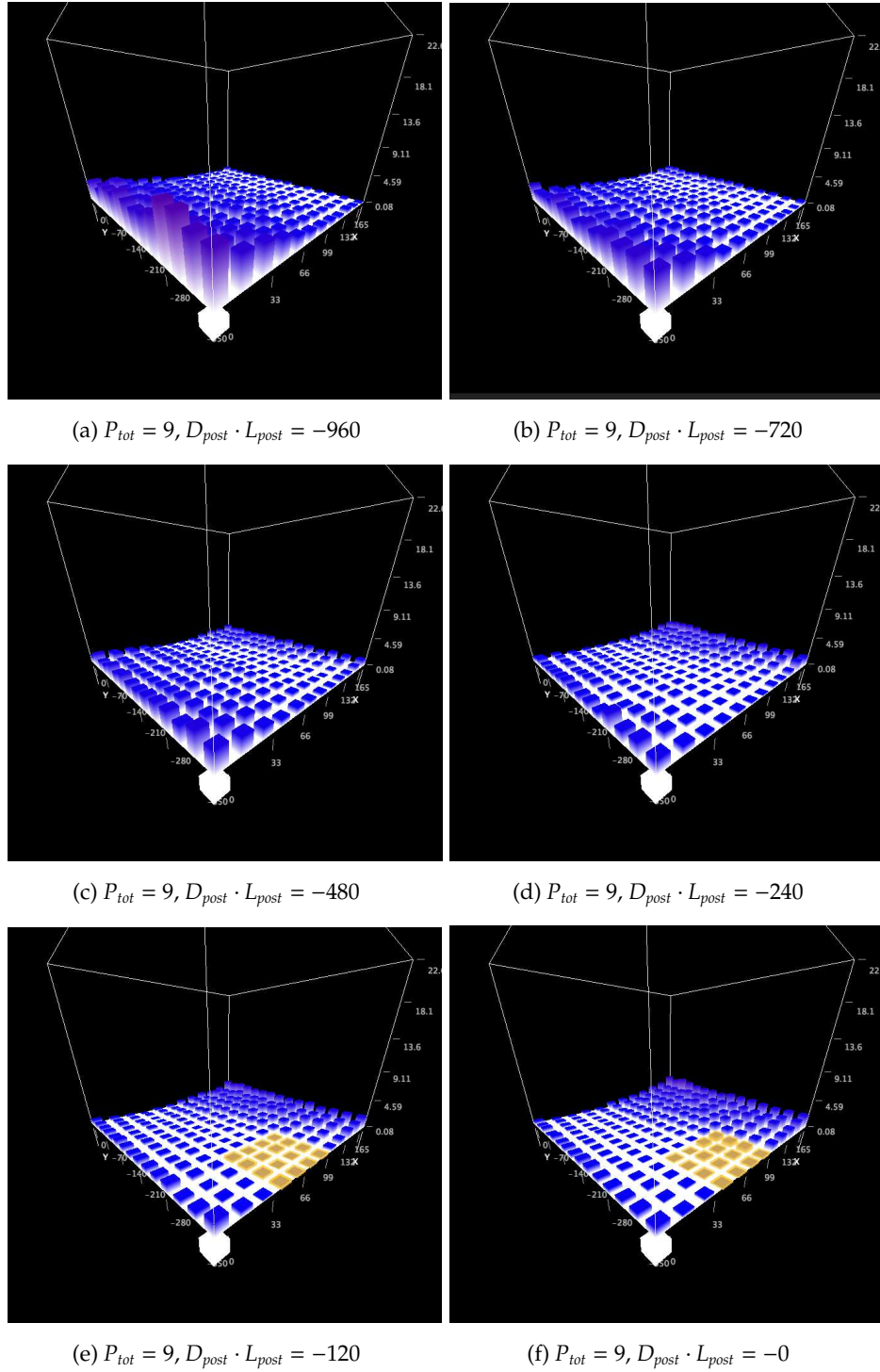


Abbildung 11.3: Visualisierung der 10x10Gb/s Simulation mit Hilfe der erstellten Software. Die X-Achse zeigt $D \cdot L$ der DCF(inline), die Y-Achse zeigt $D \cdot L$ der DCF(pre). Leistung der Verstärker $P_{tot} = 9$ und $D \cdot L$ der DCF(post) variiert zwischen den Bildern. Die Goldmarkierung zeigt den (echt) optimalen Bereich.

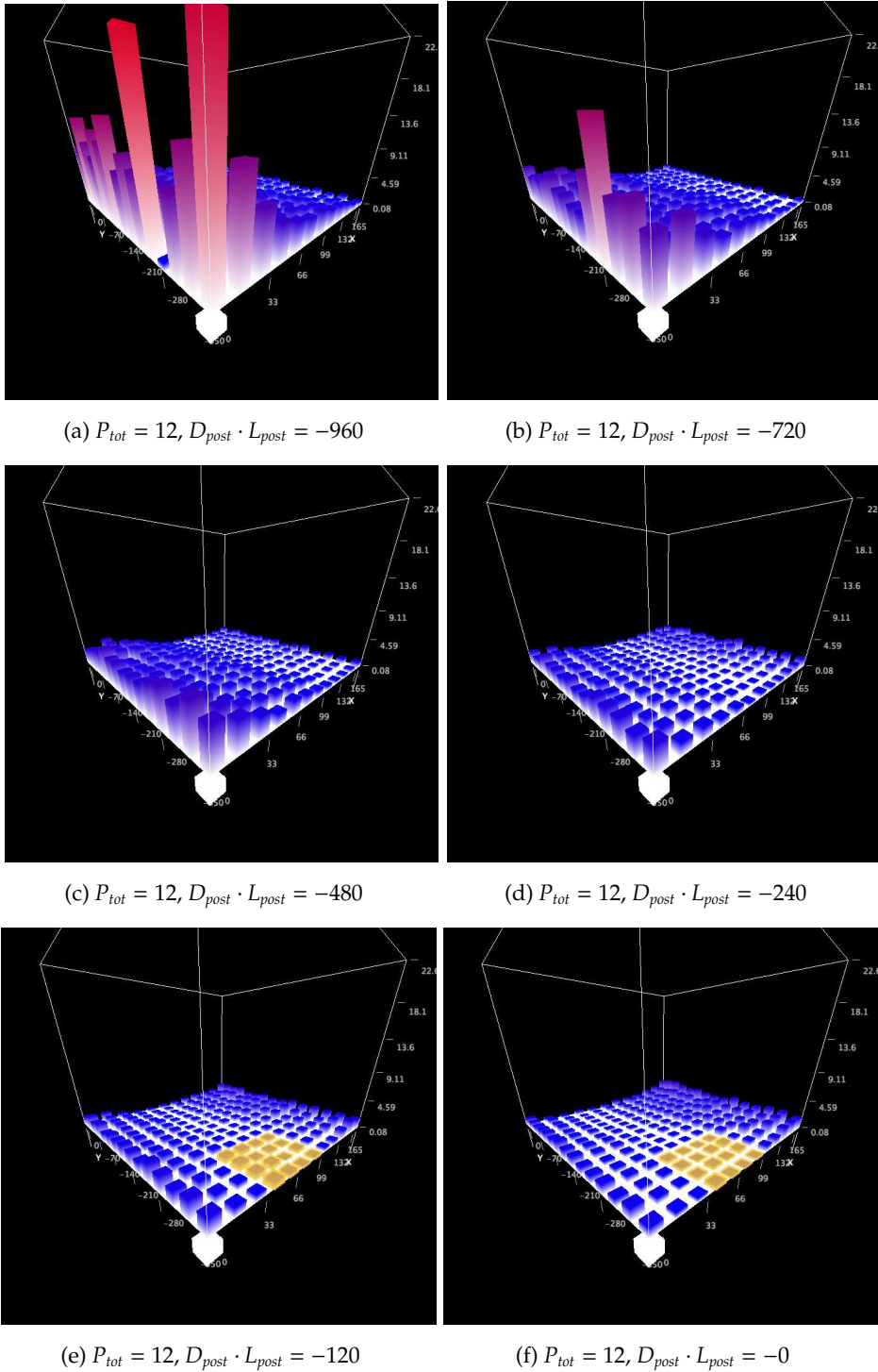


Abbildung 11.4: Visualisierung der 10x10Gb/s Simulation mit Hilfe der erstellten Software. Die X-Achse zeigt $D \cdot L$ der DCF(inline), die Y-Achse zeigt $D \cdot L$ der DCF(pre). Leistung der Verstärker $P_{tot} = 12$ und $D \cdot L$ der DCF(post) variiert zwischen den Bildern. Die Goldmarkierung zeigt den (echt) optimalen Bereich.

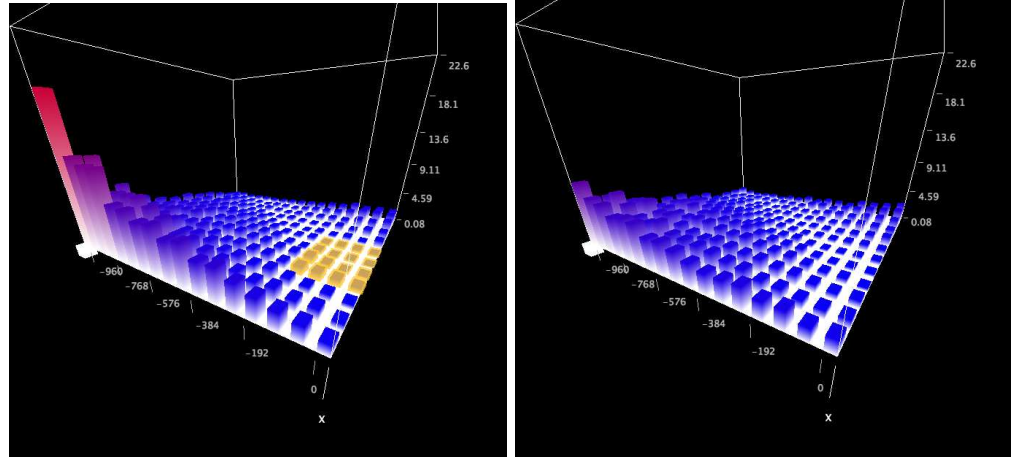
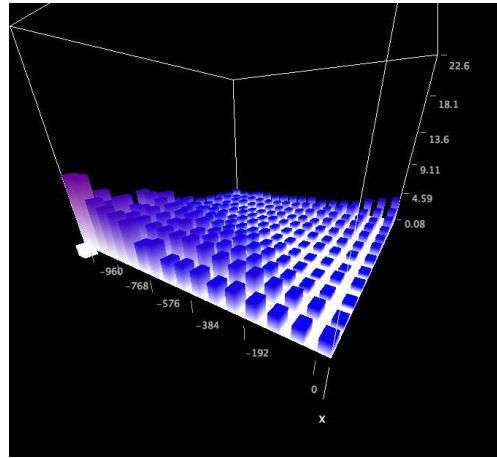
(a) $P_{tot} = 12, D_{pre} \cdot L_{pre} = -350$ (b) $P_{tot} = 12, D_{pre} \cdot L_{pre} = -175$ (c) $P_{tot} = 12, D_{pre} \cdot L_{pre} = 0$

Abbildung 11.5: Visualisierung der 10x10Gb/s Simulation mit Hilfe der erstellten Software. Andere Ausrichtung der Schnitteben als bisher: Die X-Achse zeigt $D \cdot L$ der DCF(post), die Y-Achse zeigt $D \cdot L$ der DCF(inline). Leistung der Verstärker $P_{tot} = 12$ und $D \cdot L$ der DCF(pre) variiert zwischen den Bildern. Die Goldmarkierung zeigt den (echt) optimalen Bereich.

- für die Unterkompensation der Übertragungsstrecke von $0 \frac{ps}{nm}$ bis $200 \frac{ps}{nm}$ in Schritten der Weite $8 \frac{ps}{nm}$

Die Ausgangsleistung der Verstärker wurde wieder auf 9, 10, 11 und 12 dBm gesetzt. Es wurden diesmal also 5408 Parameterkombinationen simuliert.

Einen mit der erstellten Software ermöglichten Überblick über die Simulationsergebnisse zeigt Abbildung 11.6

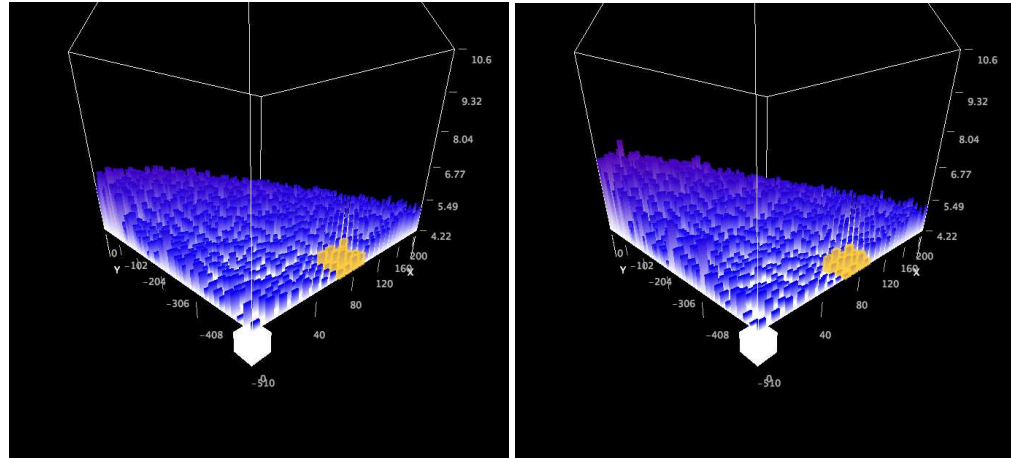
11.2. Nichtangegebene Parameter

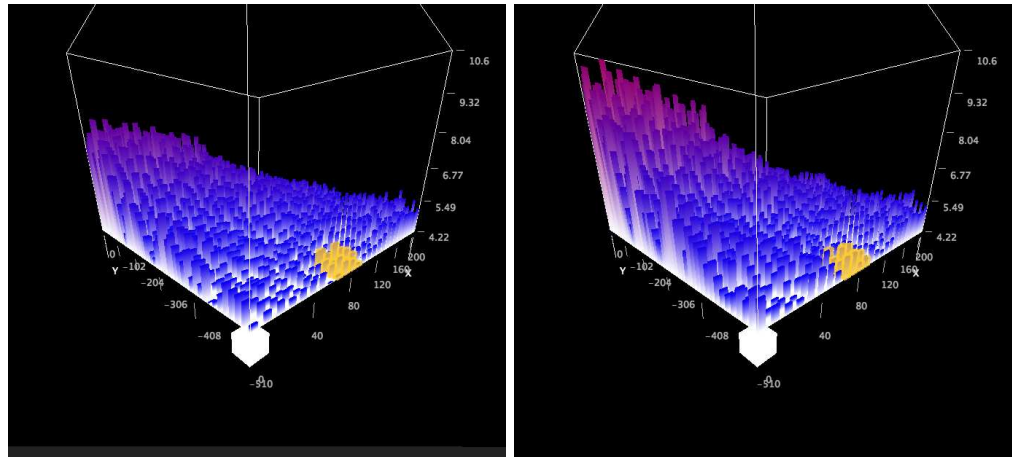
In den folgenden Kapiteln werden nur die relevanten und variierten Parameter der vorgestellten Verfahren genannt. Es wird nicht eingegangen auf die Parameter des Verfahrens zur Erstellung von Maximin-Latinhypercube-Designs und den Parametern der Funktionsschätzer, sowie die Parameter des Suchverfahrens Simulated Annealing. Diese Parameter wurden auf die in Tabelle 11.2 genannten Werte festgelegt.

Zu der Wahl des Parameters p der Maximin-Bewertungsfunktion siehe [MM92]. Die Parameter der Funktionsschätzer wurden so gewählt, dass diese möglichst glatt erscheinen. Simulated Annealing wird mit diesen Parametern so eingestellt, dass es in angemessener Zeit terminiert.

Tabelle 11.2: Festgelegte Parameter

Verfahren	Parameterausprägung
Maximin-Bewertungsfunktion: $\Phi_p(\mathcal{D})$	$p = 5$
Shepard-Interpolant	$\mu = 4$
Hardy Multiquadrik	$\mu = \frac{1}{2}$
Hardy Multiquadrik	$R = 0$
Simulated Annealing: Schrittzahl	$i_{max} = 50$
Simulated Annealing: Initialtemperatur	$t_0 = 5$
Simulated Annealing: Temperatur-Faktor	$temperatureFactor(t) = 0,7$


(a) $P_{tot} = 9$

(b) $P_{tot} = 10$

(c) $P_{tot} = 11$

(d) $P_{tot} = 12$

Abbildung 11.6: Visualisierung der 10x107Gb/s Simulation mit Hilfe der erstellten Software. Die X-Achse zeigt $D \cdot L$ der DCF(inline), die Y-Achse zeigt $D \cdot L$ der DCF(pre). Die Goldmarkierung zeigt den (echt) optimalen Bereich.

12. Evaluation der Funktionsschätzer

In diesem Kapitel werden die in Kapitel 5 vorgestellten Verfahren evaluiert. Der Fokus liegt dabei auf deren Leistung in dem für diese Arbeit relevanten Bereich, also der Schätzung einer Funktion, die auf der Simulation eines optischen Nachrichtennetzes beruht. Dazu wurden die zwei im Voraus berechneten Simulationen als Datenbasis verwendet. Von den (vielen) bekannten Parameterkombinationen dieser Simulationen können wenige ausgewählt werden, die als Stützstellen für einen Funktionsschätzer dienen. Trotzdem sind die restlichen Werte der Funktion bekannt und können zur Bewertung des Schätzers herangezogen werden.

Zur Verdeutlichung zeigt Abbildung 12.1 die Ergebnisse der 10x107Gb/s Simulation mit einem Hardy-Multiquadrik Interpolanten. Abbildung 12.2 zeigt die Ergebnisse der 10x10Gb/s Simulation ebenfalls mit einem Hardy-Multiquadrik Interpolanten.

12.1. Schätzfehler

Bei einem Funktionsschätzer ist zunächst von Interesse, wie groß der Fehler im Mittel ist, den dieser bei der Schätzung macht. Da die geschätzten Werte (und damit die Fehler dieser) von der Anzahl und der Wahl der Stützstellen abhängt, wurde jedes evaluierte Schätzverfahren für viele verschiedene Designs mit unterschiedlichen Anzahlen von Designpunkten erstellt und die Abweichung zu der im Voraus berechneten Funktion gemessen. Für eine Instanz eines Funktionsschätzers wurde der gemittelte relative Fehler und der root mean square (RMS) Fehler berechnet. Der RMS-Fehler wurde normiert, indem er durch die Differenz zwischen Maximum und Minimum der zugrunde liegenden Funktion geteilt wurde. Für eine Anzahl an Abtastpunkten n wurden jeweils 100 Latinhypercube-Designs und 100 Maximin-Latinhypercube-Designs erzeugt und diese als Stützstellen für einen Funktionsschätzer verwendet. Über den gemittelten relativen bzw. den RMS-Fehler dieser jeweils 100 Schätzer wurde ebenfalls gemittelt, so dass für jedes Schätzverfahren, jedes n und die beiden Design-Klassen LHD und MaximinLHD ein mittlerer relativer Fehler und ein mittlerer, normierter RMS-Fehler vorliegen. n wurde von 5 bis 195 mit Schrittweite 10 variiert.

Abbildung 12.3 zeigt die Ergebnisse für die in Kapitel 5.1 vorgestellte Ausgleichsebene, Abbildung 12.4 die, für die in Kapitel 5.2 vorgestellte Shepard Methode und Abbildung 12.5 die, für die in Kapitel 5.3 vorgestellte Hardy Multiquadrik. Alle zeigen den Schätzfehler bei der Schätzung der Funktionswerte der 10x107Gb/s Simulation. Die Schätzfehler bei der Schätzung der Funktionswerte der 10x10Gb/s Simulation verhalten sich ähnlich, sind aber nicht abgebildet. Zunächst ist zu erkennen, dass bei einer größeren Anzahl Abtastungen der Fehler im Mittel kleiner ist. Außerdem zeigen die Ergebnisse, dass die Maximin-Latinhypercube-Designs im Mittel zu einer genaueren Schätzung führen, als die reinen Latinhypercube-Designs. Abbildung 12.6 zeigt einen Vergleich der drei Schätzverfahren nur auf Maximin-Latinhypercube-Designs bezüglich beider Evaluationsdatensätze. Es ist deutlich zu erkennen, dass bei beiden Datensätzen die Hardy Multiquadrik ab 25 Abtastpunkten im Mittel den kleinsten relativen Fehler aufweist.

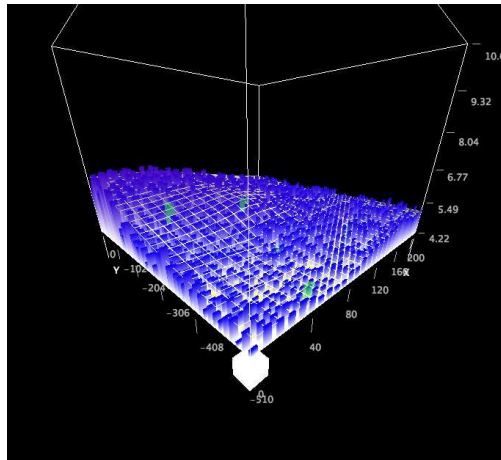
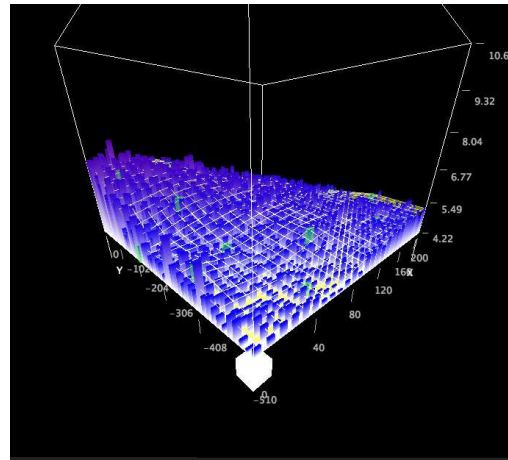
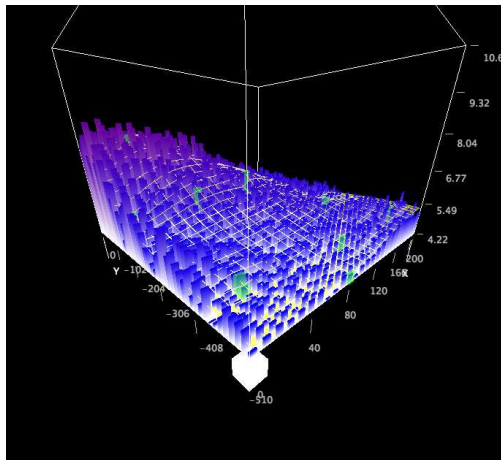
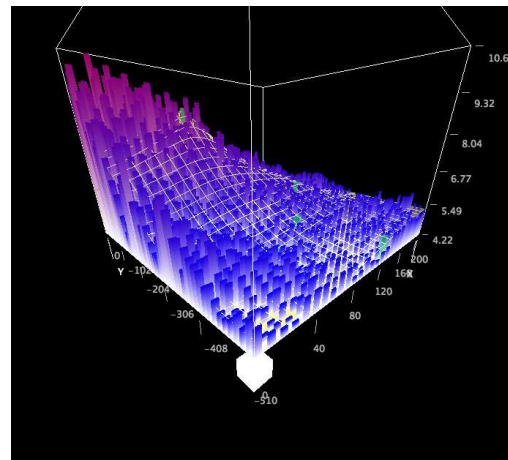
(a) $P_{tot} = 9$ (b) $P_{tot} = 10$ (c) $P_{tot} = 11$ (d) $P_{tot} = 12$

Abbildung 12.1: Ergebnisse der 10x107Gb/s Simulation mit einem Hardy Multiquadrik Interpolanten basierend auf einem Maximin-Latinhypercube-Design mit 50 Punkten. X-Achse: $D \cdot L$ der DCF(inline), Y-Achse: $D \cdot L$ der DCF(pre).

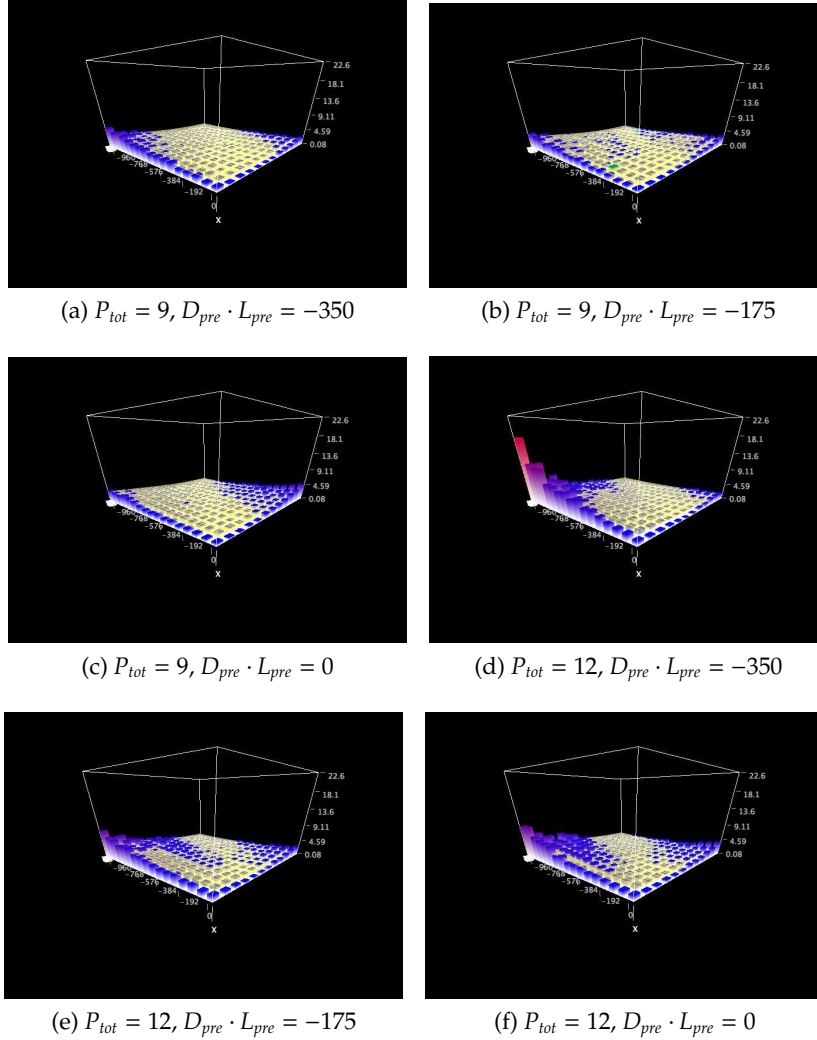
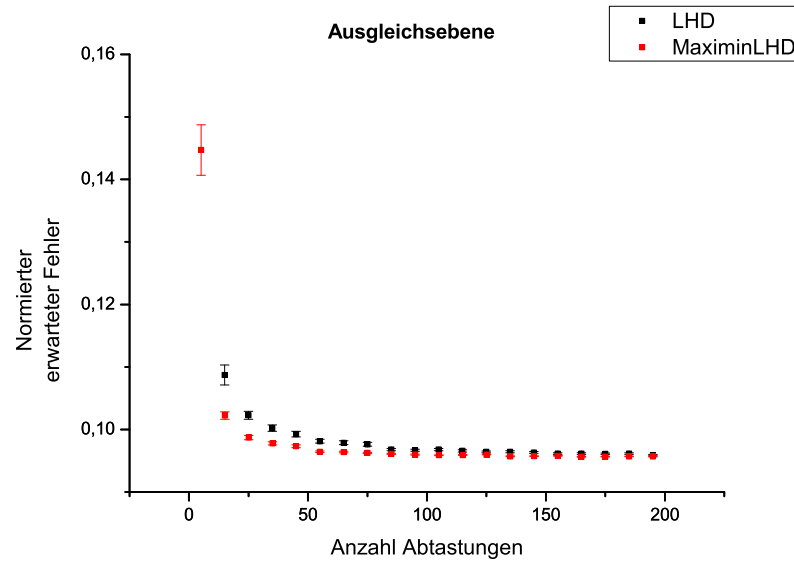
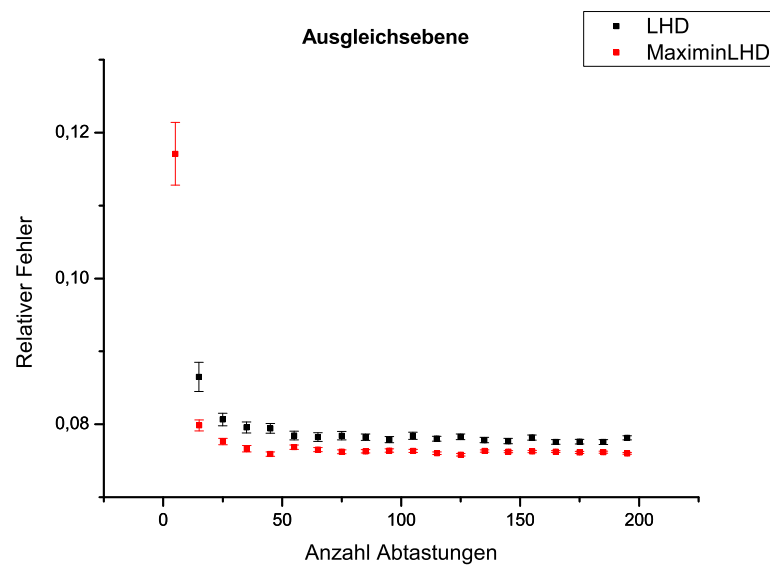


Abbildung 12.2: Ergebnisse der 10x10Gb/s Simulation mit einem Hardy Multiquadrik Interpolanten basierend auf einem Maximin-Latinhypercube-Design mit 50 Punkten. X-Achse: $D \cdot L$ der DCF(inline), Y-Achse: $D \cdot L$ der DCF(pre).

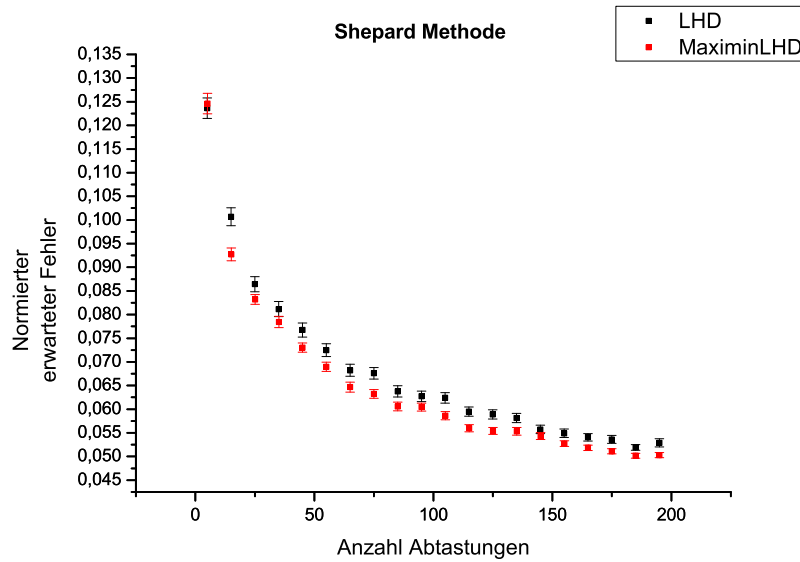


(a) Normierter RMS-Fehler

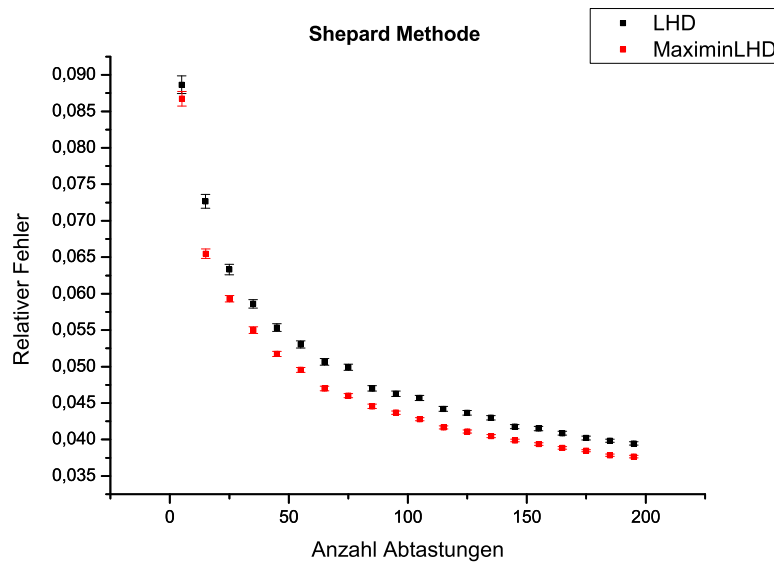


(b) Relativer Fehler

Abbildung 12.3: Fehler der Ausgleichsebene auf den Daten der 10x107Gb/s Simulation für Latinhypercube- und Maximin-Latinhypercube-Designs von 5 bis 195 Abtastungen. Mit 90% Konfidenzintervall. 100 Läufe pro errechnetem Mittelwert.

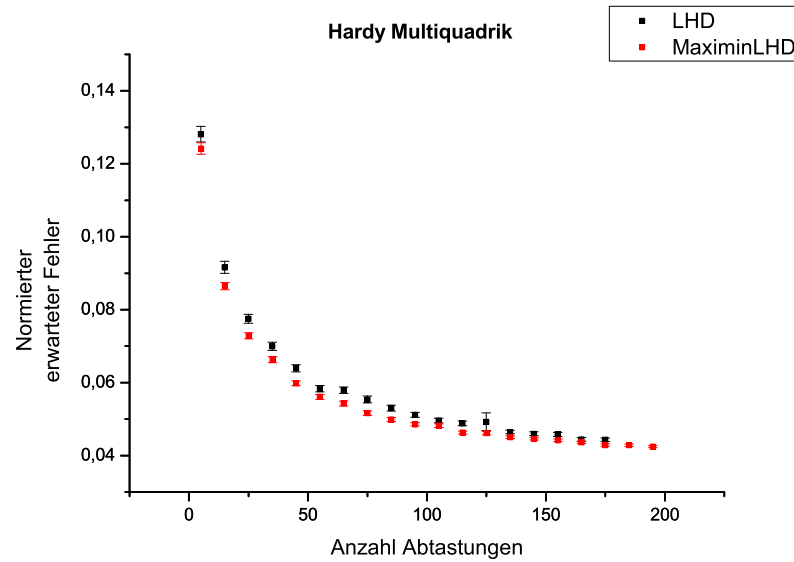


(a) Normierter RMS-Fehler

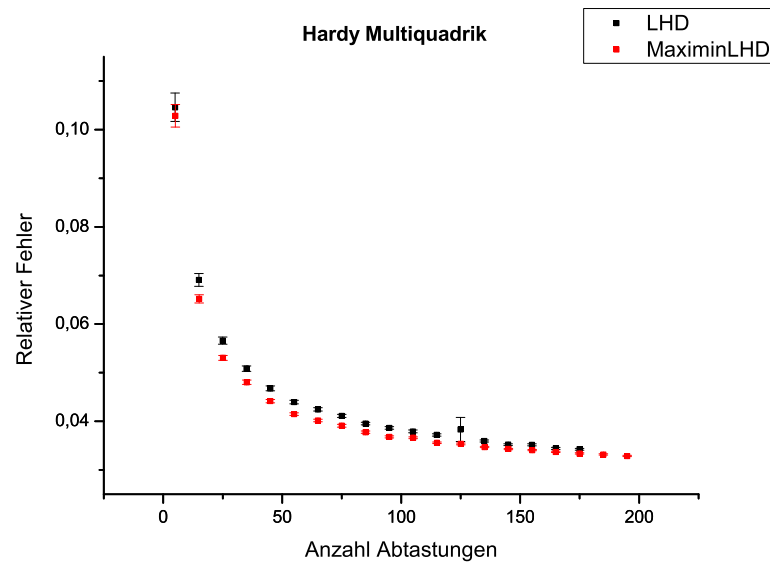


(b) Relativer Fehler

Abbildung 12.4: Fehler der Shepard Methode auf den Daten der 10x107Gb/s Simulation für Latinhypercube- und Maximin-Latinhypercube-Designs von 5 bis 195 Abtastungen. Mit 90% Konfidenzintervall. 100 Läufe pro errechnetem Mittelwert.

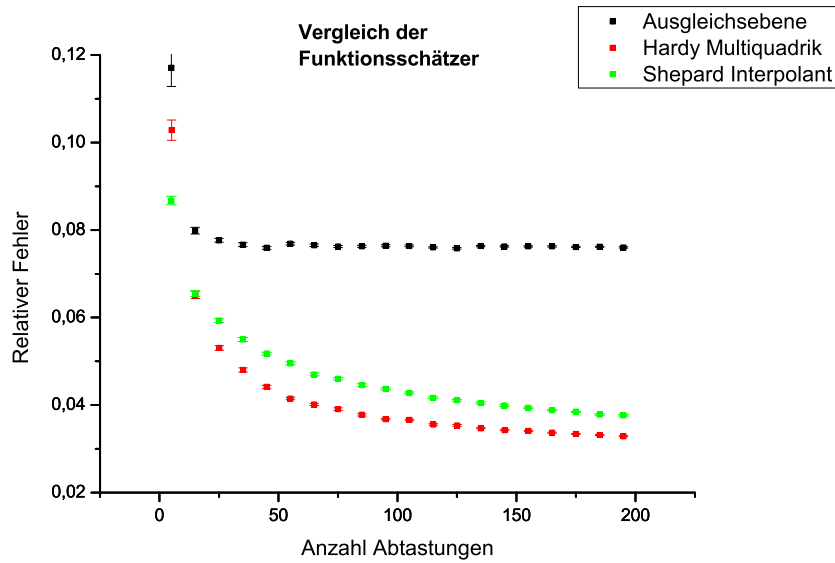


(a) Normierter RMS-Fehler

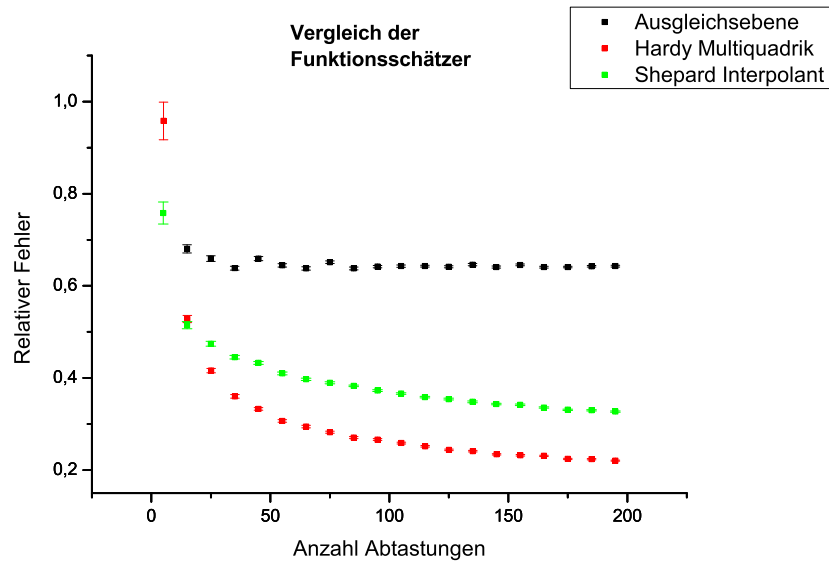


(b) Relativer Fehler

Abbildung 12.5: Fehler der Hardy Multiquadrik auf den Daten der 10x107Gb/s Simulation für Latinhypercube- und Maximin-Latinhypercube-Designs von 5 bis 195 Abtastungen. Mit 90% Konfidenzintervall. 100 Läufe pro errechnetem Mittelwert.



(a) 10x107Gb/s Simulation



(b) 10x10Gb/s Simulation

Abbildung 12.6: Vergleich der gemittelten relativen Fehler der Schätzverfahren. Für beide Evaluationsdatensätze, für Maximin-Latinhypercube-Designs von 5 bis 195 Abtastungen. Mit 90% Konfidenzintervall. 100 Läufe pro errechnetem Mittelwert. Die Hardy-Multiquadrik weist den kleinsten Fehler auf.

12.2. Optimierung auf Schätzer

Als nächstes wurde untersucht, inwiefern mit Hilfe eines Funktionsschätzers basierend auf wenigen Stützstellen das Optimum der Funktion gefunden werden kann. Dazu wurden wieder zunächst Latinhypercube- und Maximin-Latinhypercube-Designs mit verschiedenen Anzahlen an Design-Punkten generiert und mit Hilfe der Funktionswerte an diesen Punkten die Funktionsschätzer erzeugt. Anstatt den Fehler des Schätzers zu berechnen, wurde nun mit Simulated Annealing das Optimum des Funktionsschätzers gesucht. Der Funktionswert der zugrunde liegenden Funktion an der Stelle des Optimums des Schätzers wurde verglichen mit dem Funktionswert des Optimums der echten Funktion. Sei f die echte Funktion, F der Funktionsschätzer, o_{est} das optimale Argument des Schätzers F und o_{real} das optimale Argument der Funktion f . Es gilt (für den Fall der Minimierung) $f(o_{est}) - f(o_{real}) \geq 0$. Es sei weiterhin f_{min} der minimale und f_{max} der maximale Funktionswert von f . Als relative Abweichung des gefundenen Optimums wurde

$$A = \frac{f(o_{est}) - f(o_{real})}{f_{max} - f_{min}} \quad (12.1)$$

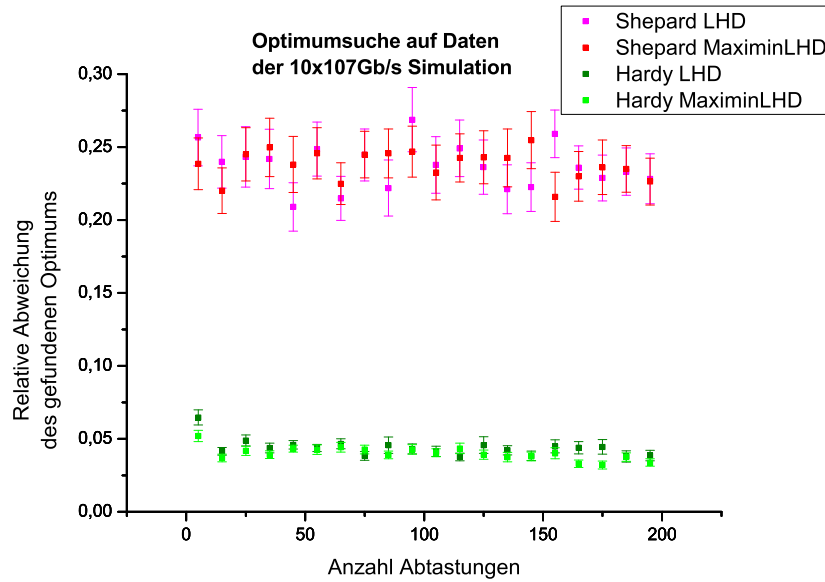
berechnet.

Die Anzahl an Abtastpunkten wurde wieder von 5 bis 195 in Schritten der Weite 10 variiert. Für jede Anzahl an Abtastpunkten wurden 10.000 Designs, Schätzer und somit A -Werte erzeugt. Über alle 10.000 A -Werte wurde gemittelt und das 90% Konfidenzintervall berechnet. Abbildung 12.7 zeigt diese Ergebnisse für alle vier Kombinationsmöglichkeiten aus Shepard-Methode, Hardy-Multiquadrik, Latinhypercube-Design und Maximin-Latinhypercube-Design; jeweils für beide Evaluationsdatensätze.

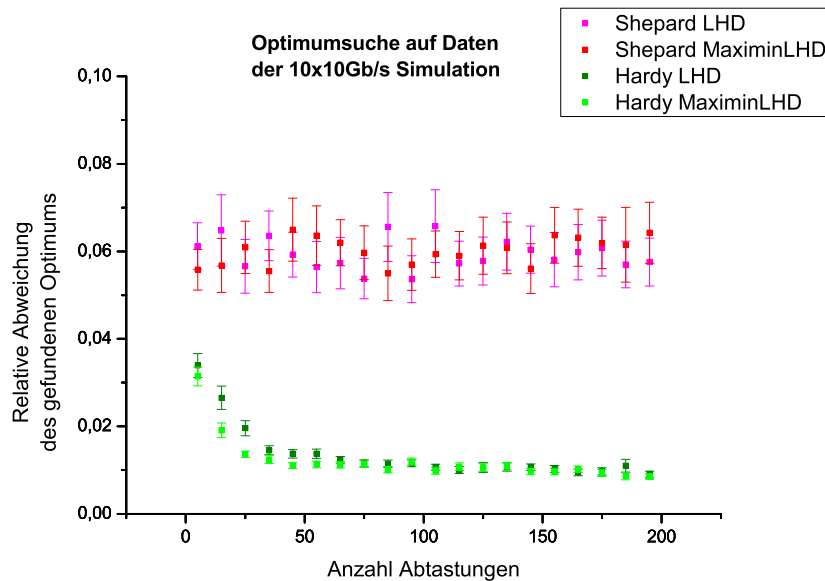
Es ist deutlich zu sehen, dass auch hier das Hardy-Multiquadrik Verfahren die Aufgabe besser bewältigt, als die Shepard-Methode. Erstaunlich ist es, dass die Anzahl der Abtastpunkte nur geringen Einfluss auf das Ergebnis nimmt. Ab einer Zahl von 35 Abtastpunkten bringt eine Erhöhung kaum noch Vorteile.

Eine mögliche Erklärung für die unterschiedliche Leistung der Shepard Interpolation und der Hardy Multiquadrik könnte darin liegen, dass die Extremwerte des Shepard Interpolanten immer auf seinen Stützstellen liegen (siehe Abschnitt 5.2). Wie gut das Optimum des Shepard Interpolanten das echte Optimum beschreibt, hängt also nur von dem zugrunde liegenden Design ab. Dies erklärt auch das, im Vergleich zu den Ergebnissen der Hardy-Multiquadrik, größere Konfidenzintervall (also die geringere Konfidenz des Mittelwertes, die eine größere Varianz anzeigt).

Die Ergebnisse zeigen, dass beide Verfahren bei der 10x10Gb/s Simulation besser abschneiden, als bei der 10x107Gb/s Simulation. Bei letzterer scheint es (obwohl sie einen Parameter weniger variiert) schwieriger zu sein, das Optimum zu finden, was evtl. in einer höheren Varianz begründet sein könnte. Vergleiche dazu die Abbildungen 12.1 und 12.2.



(a) 10x107Gb/s Simulation



(b) 10x10Gb/s Simulation

Abbildung 12.7: Vergleich des Funktionswertes des per Simulated Annealing auf dem Funktionsschätzer gefundenen Optimums mit dem globalen Optimum. Für beide Evaluationsdatensätze, für Maximin-Latinhypercube-Desings von 5 bis 195 Abtastungen. Mit 90% Konfidenzintervall. 10.000 Läufe pro errechnetem Mittelwert.

13. Evaluation des Optimierungsverfahrens

Schließlich wurde das in dieser Arbeit eingeführte Meta-Modell basierte Optimierungsverfahren evaluiert. Dazu wurden die in Tabelle 8.1 zusammengefassten Parameter des Verfahrens untersucht. Diese sind die Anzahl an Abtastungen pro Schritt, die Faktoren fac_i und das verwendete Schätzverfahren. Die Anzahl an Abtastungen pro Schritt soll möglichst gering sein. Es wurde für die Evaluation mit den Werten 2, 5, 10 und 20 gearbeitet. Die fac_i wurden zusammengefasst zu einem Parameter fac , d.h. die Änderung der Größe des Suchfensters ist für alle Dimensionen gleich. Als Funktionsschätzer kamen die vorgestellten Verfahren Shepard-Methode und Hardy-Multiquadrik zum Einsatz.

Das Optimierungsverfahren wurde auf den Evaluationsdatensätzen ausgeführt, so dass es möglich war, die optimale Lösung (nach Definition 8.1.3) durch Ausprobieren aller Möglichkeiten zu berechnen und die von dem Verfahren gefundene Lösung damit zu vergleichen. Dazu wurde der maximale Funktionswert max_{opt} der optimalen hyperquaderförmigen Teilmenge (Definition 8.1.1) mit dem maximalen Funktionswert max_{found} der von dem Verfahren gefundenen hyperquaderförmigen Teilmenge verglichen. Der Quotient

$$Q = \frac{max_{found}}{max_{opt}} \quad (13.1)$$

ist der in den folgenden Abbildungen aufgetragene Wert.

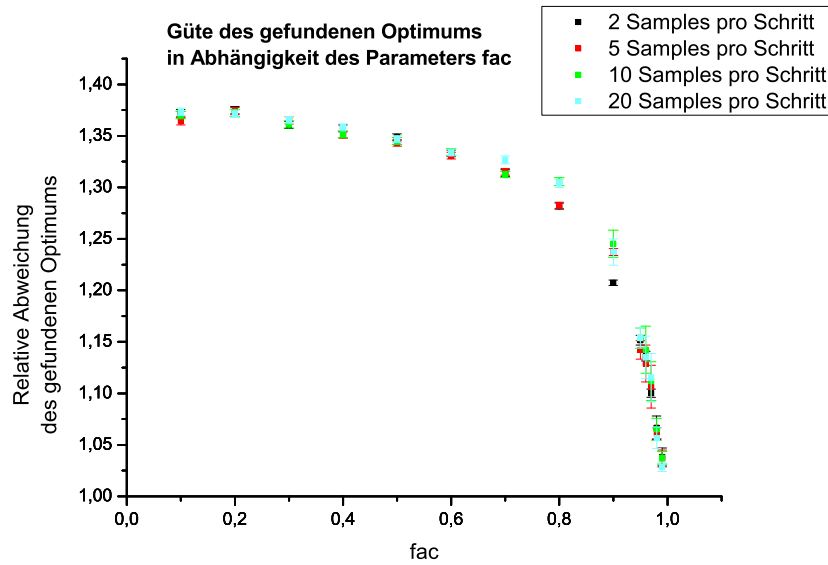
13.1. Analyse der Parameter

Auf der 10x107Gb/s Simulation wurde das Optimierungsverfahren jeweils 2000 mal für die fac -Werte 0.1 bis 0.9 (mit Schrittweite 0.1) ausgeführt und jeweils 100 mal für die fac -Werte 0.95, 0.96, 0.97, 0.98, 0.99. Dies geschah vier mal mit unterschiedlichen Werten für die Anzahl an Abtastungen pro Schritt des Suchverfahrens (2, 5, 10, 20). Um die Rechenzeit gering zu halten, wurden nur Latinhypercube-Designs (und keine Maximin-Latinhypercube-Designs) erzeugt. Der gesuchte Bereich hat die Ausdehnungen

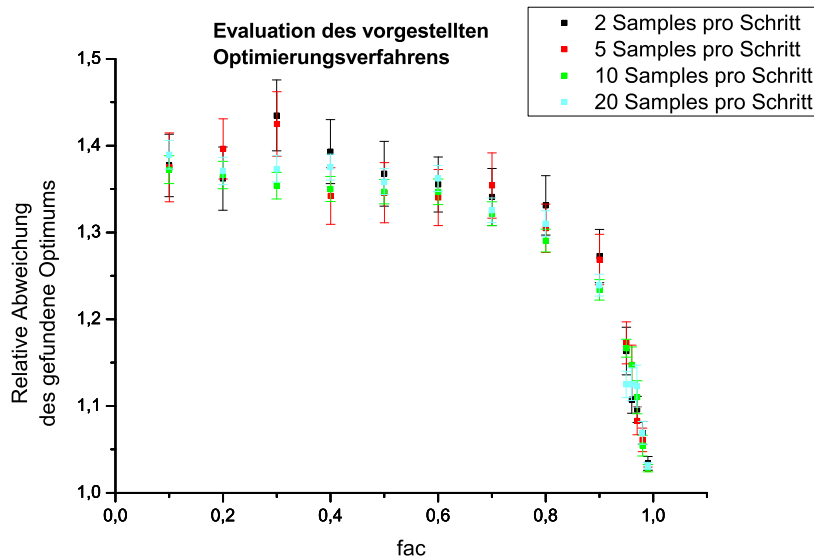
- 3 dBm für die Verstärkerleistung P_{tot} ,
- $30 \frac{ps}{km \cdot nm}$ für die Dispersionseigenschaften der Vor-Kompensation $D_{pre} \cdot L_{pre}$ und
- $24 \frac{ps}{km \cdot nm}$ für die Dispersionseigenschaften der Inline-Kompensation $D_{inline} \cdot L_{inline}$.

Abbildung 13.1 zeigt die Ergebnisse.

Erstaunlich ist, dass die Anzahl an Abtastungen pro Schritt kaum einen Einfluss zeigt. Bei 20 Abtastungen ist das Verfahren nicht messbar besser als bei nur 2 Abtastungen. Die Wahl des Parameters fac ist jedoch von Bedeutung. In dem Bereich zwischen 0.9 und 0.99 führt eine Erhöhung dieses Parameters zu einer deutlichen Reduzierung der Abweichung des gefundenen Bereichs zu dem tatsächlichen Optimum.



(a) Hardy Multiquadrik als Funktionsschätzer



(b) Shepard Interpolant als Funktionsschätzer

Abbildung 13.1: Evaluierung der Parameter des Suchverfahrens. Läufe auf der 10x107Gb/s Simulation mit unterschiedlichen fac -Werten und unterschiedlichen Anzahlen an Abtastungen pro Schritt des Suchverfahrens. Für fac -Werte ≤ 0.9 jeweils 2000 Läufe, sonst 100 Läufe. Die Ordinate zeigt den Mittelwert des berechneten Q -Wertes (Gleichung 13.1), mit 90% Konfidenzintervall.

Die gleiche Untersuchung wurde mit der Shepard Methode anstelle der Hardy Multiquadrik durchgeführt. Aus Gründen der Rechenzeit wurden hier für alle Parameterkombinationen jeweils nur 100 Durchläufe gerechnet, was die größeren Konfidenzintervalle erklärt. Aber auch hier ist zu sehen, dass die Anzahl der Abtastungen pro Schritt kaum eine Auswirkung hat. Analog zu den Ergebnissen der Hardy Multiquadrik hat der Parameter fac in dem Bereich ≥ 0.9 eine deutliche Auswirkung auf die Güte des gefundenen Optimums.

Um die Auswirkung der Wahl des Funktionsschätzers auf die Leistung des Verfahrens betrachten zu können, wurden die Q -Werte für beide Schätzverfahren in Abbildung 13.2 zusammen in ein Diagramm gezeichnet. Die Anzahl an Abtastungen pro Schritt wurde auf 2 gesetzt. Es wurden 2000 Läufe pro Parameterkombination berechnet. Die Hardy-Multiquadrik scheint auch hier zu besseren Ergebnissen zu führen.

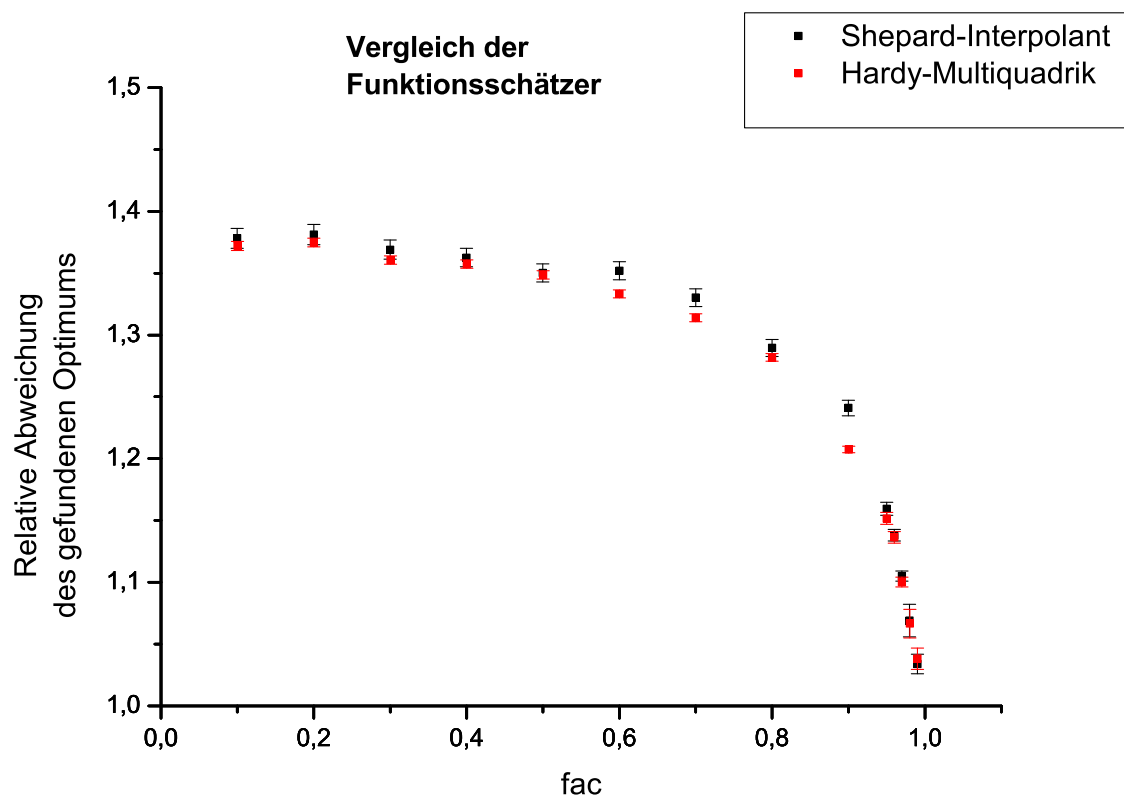


Abbildung 13.2: Vergleich der Schätzverfahren als Teilverfahren der Optimierung.

13.2. Betrachtung des Parameters fac

Eine Verbesserung der Güte des gefundenen Optimums für fac -Werte nahe an 1 ist nicht verwunderlich, wenn man Abbildung 13.3 betrachtet. Die Anzahl der erfolgten Schritte des Verfahrens

steigt für solche fac -Werte stark an. Der Grund dafür liegt darin, dass das Verfahren abbricht, wenn das Suchfenster auf die gewünschte Größe verkleinert wurde. fac gibt gerade an, wie stark das Fenster in einem Schritt verkleinert wird. Für einen Wert von 1 bliebe es gleich groß, für 0,5 würde die Ausdehnung in jeder Dimension halbiert. Da in jedem Schritt eine konstante Anzahl an Funktionsauswertungen vorgenommen wird, ist die Anzahl an Schritten proportional zu der Anzahl an Funktionsauswertungen insgesamt.

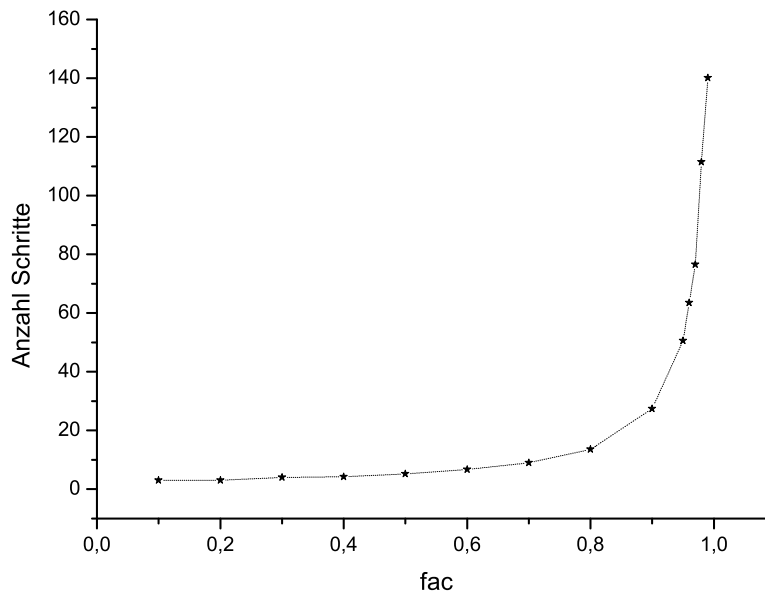


Abbildung 13.3: Anzahl Schritte in Abhängigkeit von fac

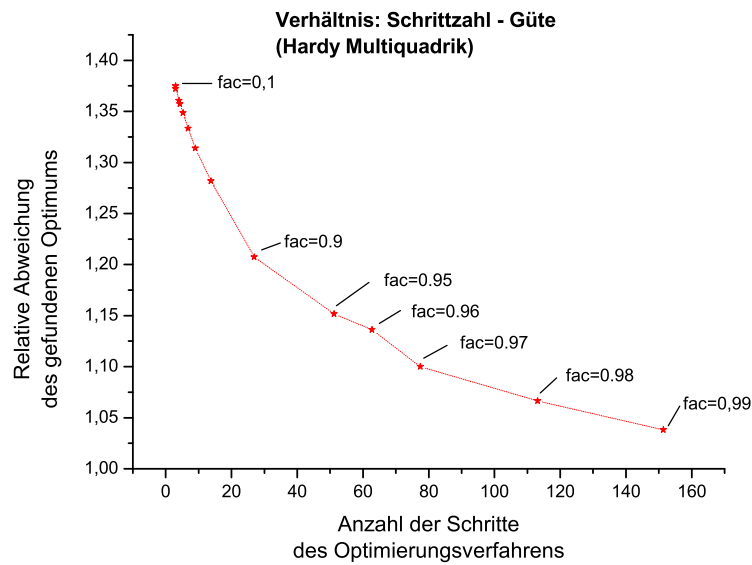
Das bedeutet also, dass über den Parameter fac sowohl die potentielle Güte des gefundenen Optimums als auch die dafür nötigen Kosten festgelegt werden. Abbildung 13.4 zeigt den Zusammenhang zwischen Güte und Kosten für die im letzten Abschnitt vorgestellten Ergebnisse.

Als anschauliches Beispiel wurde ein Lauf des Verfahrens mit einem fac -Wert von 0.9 und nur 2 Abtastungen pro Schritt auf der 10x107Gb/s Simulation mit den o.g. Ausdehnungen des gesuchten Bereiches durchgeführt. Als Model wurde die Hardy-Multiquadrik gewählt. Aus Abbildung 13.4 ist ersichtlich, dass mit etwa 25 Schritten zu rechnen ist. Der echt optimale und der von dem Verfahren gefundene Bereich ist in Abbildung 13.5 zu sehen.

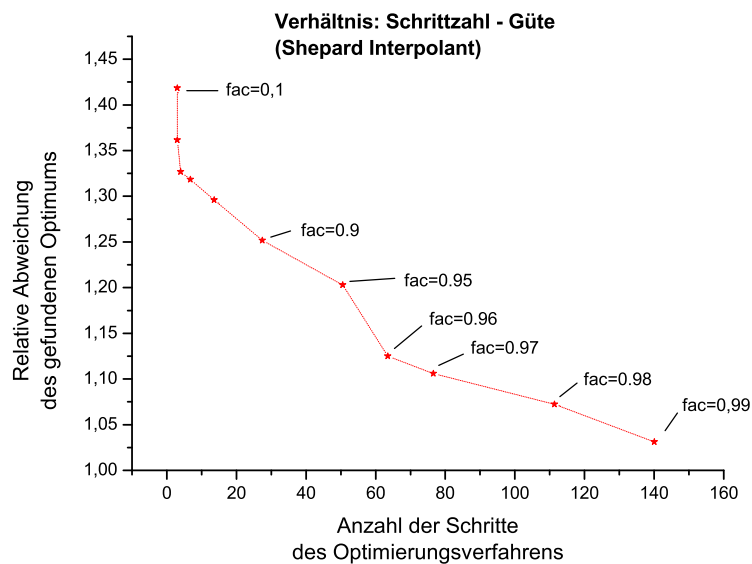
Dies ist eine Lösung, die nur wenig schlechter als die optimale ist. Jedoch wurden nur $2 \cdot 25 = 50$ Funktionsauswertungen benötigt, um diese zu finden. Verglichen mit den 5408 Auswertungen, die insgesamt gemacht wurden, ist dies eine Kosten- und damit Zeitersparnis um den Faktor 100.

Es wurden zwei weitere Beispiel-Läufe berechnet. Wieder mit der 10x107Gb/s Simulation und einem fac -Wert von 0.9. Diesmal jedoch mit einem größeren gesuchten Bereich:

- 3 dBm für die Verstärkerleistung P_{tot} ,
- $80 \frac{ps}{km \cdot nm}$ für die Dispersionseigenschaften der Vor-Kompensation $D_{pre} \cdot L_{pre}$ und



(a) Hardy Multiquadrik als Funktionsschätzer



(b) Shepard Interpolant als Funktionsschätzer

Abbildung 13.4: Verhältnis zwischen der Anzahl der Schritte des Suchverfahrens und der Güte der gefundenen Lösung.

- $80 \frac{\text{ps}}{\text{km} \cdot \text{nm}}$ für die Dispersionseigenschaften der Inline-Kompensation $D_{\text{inline}} \cdot L_{\text{inline}}$.

Bei dem ersten Lauf wurden wieder 2 Abtastungen pro Schritt vorgenommen, bei dem zweiten 5. Die Ergebnisse sind in den Abbildungen 13.6 und 13.7 zu sehen. Bei dem zweiten Lauf mit 5 Abtastungen pro Schritt ist der gefundene Bereich nahezu kongruent zu dem optimalen. Da der gesuchte Bereich diesmal größer ist, terminierte die Suche bereits nach 15 Schritten. D.h. es wurden diesmal nur 30 bzw. 75 Auswertungen vorgenommen.

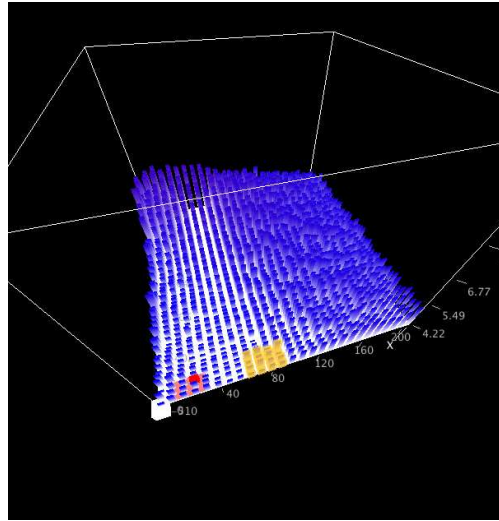
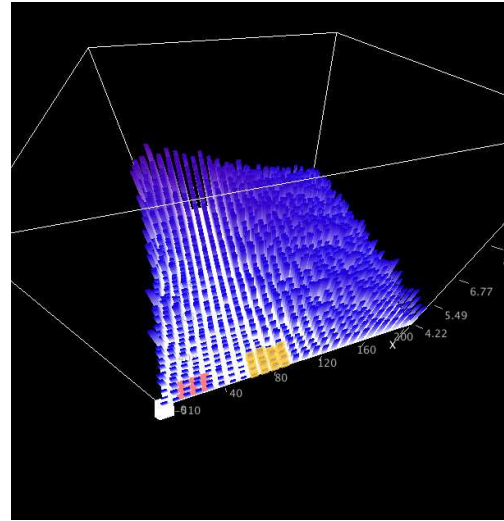
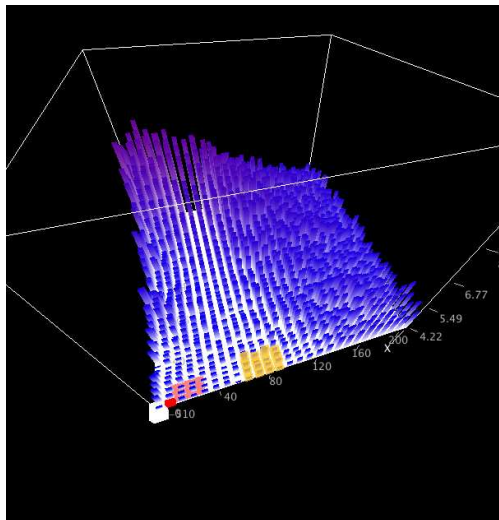
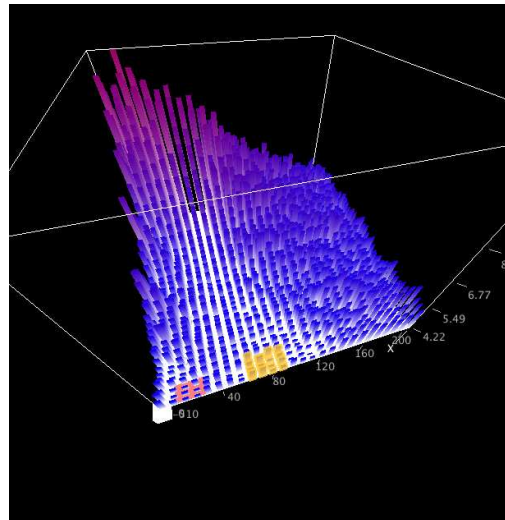
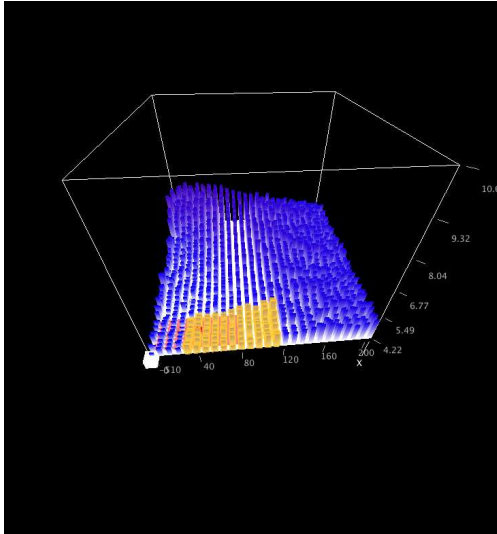
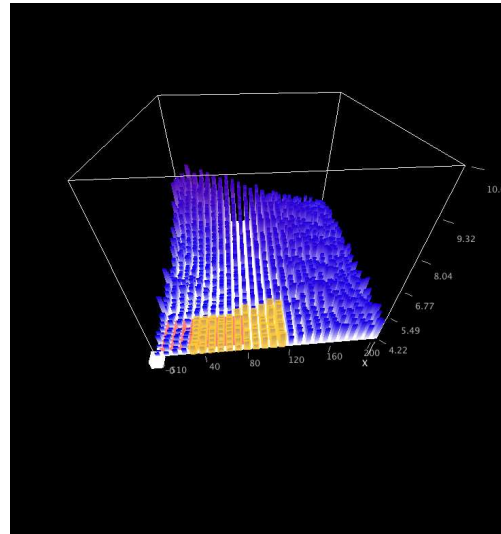
(a) $P_{tot} = 9$ (b) $P_{tot} = 10$ (c) $P_{tot} = 11$ (d) $P_{tot} = 12$

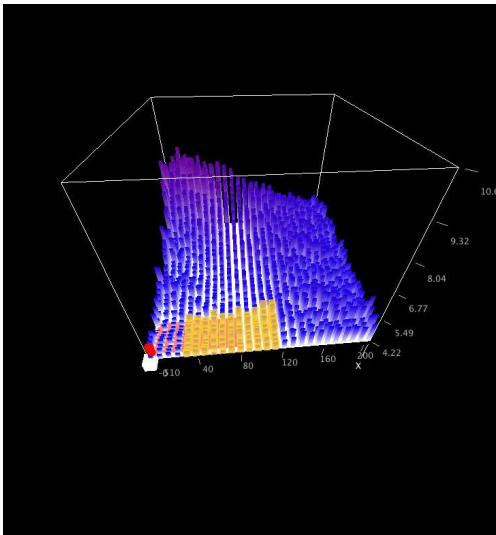
Abbildung 13.5: Ergebnisse der 10x107Gb/s Simulation. Die X-Achse repräsentiert $D \cdot L$ der Inline-Kompensation, Die Y-Achse repräsentiert $D \cdot L$ der Vor-Kompensation. Abbildung zeigt die Gefundene Lösung des Optimierungsverfahrens. Der echt optimale Bereich ist durch die Goldfärbung gekennzeichnet. Der von dem Verfahren gefundene Bereich ist durch eine rote Fläche unterhalb der Säulen zu erkennen.



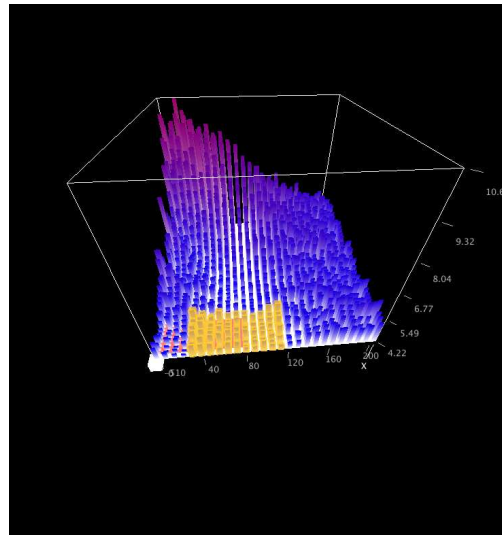
(a) $P_{tot} = 9$



(b) $P_{tot} = 10$



(c) $P_{tot} = 11$



(d) $P_{tot} = 12$

Abbildung 13.6: Ergebnisse der 10x107Gb/s Simulation. Die X-Achse repräsentiert $D \cdot L$ der Inline-Kompensation, Die Y-Achse repräsentiert $D \cdot L$ der Vor-Kompensation. Abbildung zeigt die Gefundene Lösung des Optimierungsverfahrens für den ersten Lauf mit einem größeren gesuchten Bereich. Es wurden wieder 2 Abtastungen pro Schritt vorgenommen. Der echt optimale Bereich ist durch die Goldfärbung gekennzeichnet. Der von dem Verfahren gefundene Bereich ist durch eine rote Fläche unterhalb der Säulen zu erkennen.

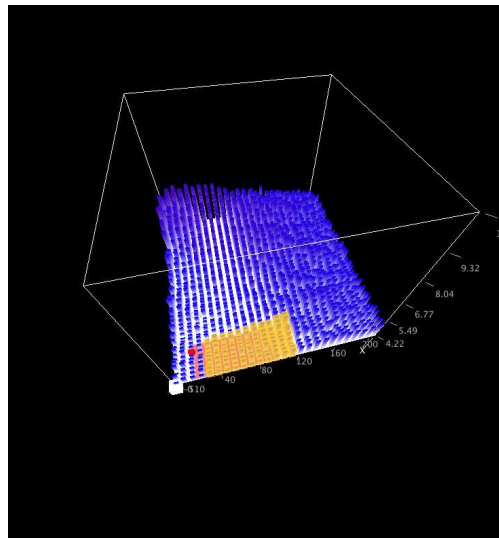
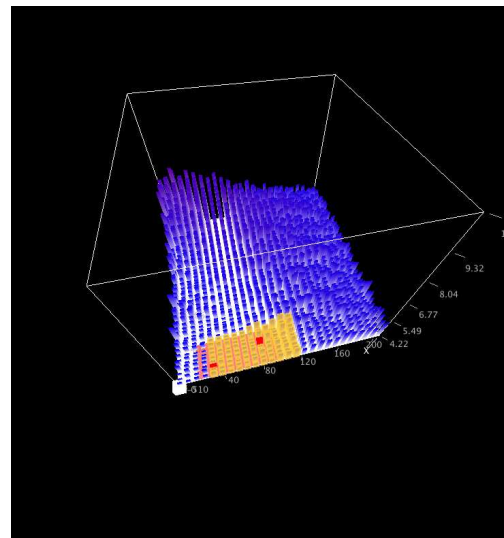
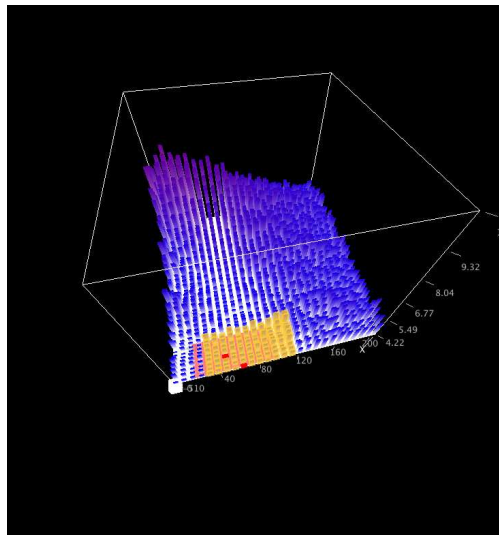
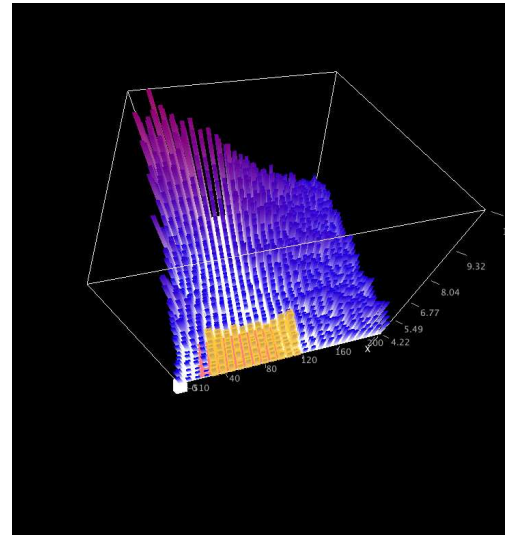
(a) $P_{tot} = 9$ (b) $P_{tot} = 10$ (c) $P_{tot} = 11$ (d) $P_{tot} = 12$

Abbildung 13.7: Ergebnisse der 10x107Gb/s Simulation. Die X-Achse repräsentiert $D \cdot L$ der Inline-Kompensation, Die Y-Achse repräsentiert $D \cdot L$ der Vor-Kompensation. Abbildung zeigt die Gefundene Lösung des Optimierungsverfahrens für den zweiten Lauf mit einem größeren gesuchten Bereich. Diesmal wurden 5 Abtastungen pro Schritt vorgenommen. Der echt optimale Bereich ist durch die Goldfärbung gekennzeichnet. Der von dem Verfahren gefundene Bereich ist durch eine rote Fläche unterhalb der Säulen zu erkennen. Dieser ist nahezu kongruent zu dem optimalen Bereich.

14. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Verfahren vorgestellt, das es erlaubt, mehrdimensionale Funktionen hinsichtlich eines ausgedehnten Bereiches zu optimieren. Für jede Dimension ist eine Ausdehnung gegeben, so dass ein entsprechend großer Bereich gesucht wird, dessen Maximum der Funktionswerte möglichst minimal ist. Motiviert ist dieses Vorgehen durch die Anforderung aus der optischen Datenübertragung, Parameter zu betrachten, die nicht beliebig genau gewählt werden können, sondern eine fertigungsbedingte Schwankung aufweisen. Eine gewählte Stelle des Parameterraumes sollte robust gegenüber solchen Schwankungen sein.

Das vorgestellte Verfahren verwendet ein Suchfenster, das in jedem Schritt verkleinert und verschoben wird. Innerhalb des Suchfensters werden in jedem Schritt Punkte ausgewählt, die ausgewertet werden, um ein Modell der Funktion zu berechnen und zu verfeinern. Auf dem Modell wird in jedem Schritt ein interessanter Bereich gesucht, um diesen herum das Suchfenster platziert wird. Hat das Suchfenster die gewünschte Größe (entsprechend der angegebenen Ausdehnung) erreicht, terminiert das Verfahren und gibt die Koordinaten des Suchfensters als Lösung aus. Auf welche Weise die auszuwertenden Punkte ausgewählt werden, wie das Modell der Funktion repräsentiert wird und wie ein interessanter Bereich eines Modells ermittelt wird, ist von dem Verfahren nicht vorgegeben. Im Zuge dieser Arbeit wurde dieses Verfahren in Form einer eigenständigen Software implementiert. Zur Auswahl der auszuwertenden Punkte wurden die Verfahren Latinhypercube-Design und Maximin-Latinhypercube-Design implementiert. Als Funktionsmodelle kamen der Shepard-Interpolant und die Hardy-Multiquadrik zum Einsatz. Mit Simulated Annealing wurde auf diesen Interpolanten das Minimum, (als interessanter Bereich) gesucht.

Um als Benutzer die zu optimierenden mehrdimensionalen Funktionen explorieren zu können, bietet die erstellte Software die Möglichkeit, Teile dieser Funktionen zu visualisieren. Es wird eine der achsenparallelen Ebenen des Parameterraumes ausgewählt und verschoben. Der Schnitt dieser Ebenen mit dem Parameterraum ist ein zweidimensionaler Unterraum, der als bivariate Funktion gezeichnet werden kann. Dadurch ist es möglich, das von dem Verfahren gefundene Optimum, das Verhalten der Funktion in der Umgebung des Optimums sowie die ausgewählten Stützstellen und das zur Optimierung verwendete Modell zu betrachten.

Der Zweck der Software ist die Anwendung des vorgestellten Verfahrens auf die Optimierung optischer Übertragungssysteme. Zur Simulation dieser Systeme wird die Software PHOTOSS verwendet. Zur Auswertung unbekannter Parameterkombinationen wird von der Software ein PHOTOSS-Prozess gestartet, der die nötigen Berechnungen durchführt. Da diese erheblich viel Zeit und Rechenkapazität in Anspruch nehmen können, werden die PHOTOSS-Prozesse nicht direkt und nicht auf der selben Maschine gestartet. Stattdessen werden mit Hilfe der Condor Gridcomputing-Software die Berechnungen auf beliebig viele Rechner verteilt. Das Erzeugen von Condor Job-Dateien, das Überwachen von gestarteten Jobs sowie das Einlesen der von PHOTOSS erzeugten Ergebnisse wird von der erstellten Software bewerkstelligt.

Es wurde gezeigt, dass mit diesem Verfahren und mit dieser Software sehr gute Lösungen in einem Bruchteil der durch vollständiges Abtasten benötigten Zeit gefunden werden können. An-

statt wie bisher üblich den Parameterraum äquidistant abzutasten und dabei vier- bis fünfstellige Anzahlen an Parameterkombinationen simulieren zu müssen, werden bei den Evaluationsdatensätzen mit unter hundert Auswertungen Lösungen gefunden, die nur wenig schlechter sind, als das Optimum. An dem Beispiel der vorgestellten 10x107Gb/s Simulation wurde gezeigt, dass eine brauchbare Lösung mit nur 50 Funktionsauswertungen gefunden werden kann. Dies bedeutet eine Kostenreduzierung um den Faktor 100. Durch das vorgestellte Verfahren und seine Implementierung wird es also möglich, Optimierungsläufe durchzuführen, die bisher aufgrund der hohen Kosten nicht durchführbar waren. Durch die Wahl eines geeigneten Optimierungsverfahrens sind also erhebliche Kosteneinsparungen beim Design optischer Übertragungssysteme möglich.

Das Verfahren kann leicht erweitert werden, indem die Teilverfahren (Latinhypercube-Design, Hardy-Multiquadrik, Simulated Annealing) durch Alternativen ersetzt werden. Verwandte Arbeiten setzen häufig Kriging oder neuronale Netze als Modell ein. Ausserdem wäre es wünschenswert, das Verfahren an mehreren unterschiedlichen Funktionen zu evaluieren. Interessant wären insbesondere hochdimensionale Funktionen, bei denen durch die Anzahl an Parametern wesentlich mehr Parameterkombinationen möglich sind und deshalb durch das vorgestellte Verfahren mehr Potential zur Einsparung von Rechenzeit besteht. Allerdings erfordert eine entsprechende Evaluation ebenfalls erheblich mehr Rechenzeit, weshalb dies im Rahmen dieser Arbeit nicht möglich war.

Das Verfahren sowie die Software ist nicht auf die Optimierung optischer Übertragungssysteme beschränkt. Mit wenig Aufwand wäre es möglich, anstelle von PHOTOSS ein anderes Programm zur Berechnung der zu optimierenden Funktion zu verwenden.

Teil V.

Anhang

Abbildungsverzeichnis

2.1. RZ-NRZ Vergleich	12
2.2. Duobinär-Verfahren	13
2.3. Augendiagramm	13
2.4. EOP-Box	14
2.5. Dispersion und Dämpfung	15
2.6. Augendiagramme Dispersion	16
2.7. Auswirkung der Dispersion	18
2.8. SPM Chirp	19
2.9. Auswirkungen der FWM	20
2.10. PHOTOS-GUI	22
4.1. Visuelle Darstellung des LHD(5,2) Beispieldesigns.	29
5.1. Ausgleichsebene	33
5.2. Shepard-Interpolant	35
5.3. Hardy-Multiquadrik	36
6.1. Zweidimensionale Funktion	39
7.1. Bildschirmfoto von <i>condor_status</i> und <i>condor_q</i>	42
9.1. Bildschirmfoto der erstellten Software.	49
9.2. Plot-Fenster, verschiedene Blickwinkel	55
9.3. Navigator-Fenster	56
9.4. Mehrere Schnitte einer dreidimensionalen Funktion	57
9.5. Visualisierung einer äquidistant abgetasteten Funktion.	58
9.6. Objekt-Liste	58
9.7. Sichtbarkeit einzelner Objekte	59
9.8. Visualisierung der Funktionsschätzer	59
9.9. Vergleich: Funktionsschätzer - abgetastete Funktion	60
9.10. Dialog: PHO-Datei	61
9.11. Leeres Plot-Fenster bei unbekannter Funktion.	62
9.12. Dialog: Optimierungsparameter	65
9.13. Benutzerinterface des Optimierungsmodus	65
9.14. Ansicht der Funktionsschätzer im Optimierungsmodus	66
9.15. Vergleich: gefundene - optimale Lösung	67
10.1. UML-Klassendiagramm: Architektur-Übersicht.	70
10.2. <i>Data</i> - und <i>Optimizee</i> -Klassen.	72
10.3. Klassen zur Erzeugung von Versuchsplänen und <i>OptimizationProcedure</i>	73

10.4. Präsentationsschicht der Software.	74
11.1. Versuchsaufbau der 10x10Gbit/s Simulation	77
11.2. Inhalt der Komponente <i>Iterator</i>	77
11.3. Visualisierung der 10x10Gb/s Simulation (1)	79
11.4. Visualisierung der 10x10Gb/s Simulation (2)	80
11.5. Visualisierung der 10x10Gb/s Simulation (3)	81
11.6. Visualisierung der 10x107Gb/s Simulation	83
12.1. Ergebnisse der 10x107Gb/s Simulation mit Hardy-Multiquadrik	85
12.2. Ergebnisse der 10x10Gb/s Simulation mit Hardy-Multiquadrik	86
12.3. Fehler der Ausgleichsebene	87
12.4. Fehler der Shepard-Methode	88
12.5. Fehler der Hardy-Multiquadrik	89
12.6. Vergleich der Schätzverfahren	90
12.7. Optimumsuche auf Funktionsschätzer	92
13.1. Evaluierung des Optimierungsverfahrens	94
13.2. Vergleich der Schätzverfahren als Teilverfahren der Optimierung	95
13.3. Anzahl Schritte in Abhängigkeit von fac	96
13.4. Verhältnis: Anzahl Schritte - Güte	97
13.5. Gefundene Lösung des Optimierungsverfahrens, erster Beispiel-Lauf	99
13.6. Gefundene Lösung des Optimierungsverfahrens, zweiter Beispiel-Lauf	100
13.7. Gefundene Lösung des Optimierungsverfahrens, dritter Beispiel-Lauf	101

Literaturverzeichnis

- [ABB⁺99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
- [Agr01] Govind P. Agrawal. *Nonlinear Fiber Optics*. Academic Press, 2001.
- [Agr02] Govind P. Agrawal. *Fiber-Optic Communication Systems*. Wiley Interscience, 2002.
- [AL97] Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.
- [Asi85] D. Asimov. The grand tour: A tool for viewing multidimensional data. *SIAM Journal of Science & Stat. Comp.*, 6:128–143, 1985.
- [ASW⁺05] ARB, Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R)*. Addison-Wesley Professional, 2005.
- [Bla75] R.W. Blanning. The construction and implementation of metamodels. *Simulation*, pages 177–184, 1975.
- [Che73] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal Amer. Statistical Association*, 68:361–368, 1973.
- [Col] CollabNet. Subversion. <http://subversion.tigris.org/>.
- [Con] High Throughput Computing Condor. <http://www.cs.wisc.edu/condor/>.
- [EE] Brendan Eich and Ecma. EcmaScript. <http://www.ecmascript.org/>.
- [FLS06] Kai-Tai Fang, Runze Li, and Agus Sudjianto. *Design and Modeling for Computer Experiments*. Computer Science and Data Analysis Series, 2006.
- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- [Gra] Silicon Graphics. OpenGL <http://www.opengl.org>.
- [gtk] Gtk+. <http://www.gtk.org/>.
- [Hen07] Nadine Henkenjohann. *Eine adaptive sequentielle Prozedur zur effizienten Optimierung des CNC-gesteuerten Druckprozesses*. PhD thesis, Universität Dortmund, Fachbereich Statistik, 2007.

- [HL87] Hoschek and Lasser. *Grundlagen der geometrischen Datenverarbeitung*. B.G. Teubner Stuttgart, 1987.
- [JE04] Jürgen Jakumeit and Michael Emmrich. Optimization of gas turbine blade casting using evolution strategies and kriging. *Proceedings of the International Conference of Bioinspired Optimization Methods and Their Applications*, '04:95–104, 2004.
- [JMY90] M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Inference and Planning*, 26:131–148, 1990.
- [KG05] Marios K. Karakasis and Kyriakos C. Giannakoglou. Metamodel-assisted multi-objective evolutionary optimization. *Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems*, 2005.
- [KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, Number 4598, 13 May 1983, 220, 4598:671–680, 1983.
- [KL02] Ivan Kaminow and Tingye Li, editors. *Optical Fibre Telecommunications - Systems and Impairments*. Academic Press, 2002.
- [Kri51] D.G. Krige. A statistical approach to some mine valuations and allied problems at the witwatersrand. Master's thesis, University of Witwatersrand, 1951.
- [LWW90] J. LeBlanc, M. O. Ward, and N. Wittels. Exploring n-dimensional databases. *Visualization '90*, pages 230–239, 1990.
- [Lük75] H. D. Lüke. *Signalübertragung*. Springer-Verlag, 1975.
- [Mat] Yukihiro Matsumoto. Ruby. <http://www.ruby-lang.org>.
- [MCB79] M.D. McKay, W.J. Conover, and R.J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.
- [Mica] Microsoft. MFC - Microsoft Foundation Classes
[http://msdn.microsoft.com/en-us/library/d06h2x6e\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/d06h2x6e(VS.80).aspx).
- [Micb] Microsoft. DirectX (Direct3D) Programmierschnittstelle
[http://msdn.microsoft.com/de-de/directx/default\(en-us\).aspx](http://msdn.microsoft.com/de-de/directx/default(en-us).aspx).
- [Mic86] Charles A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.
- [MM92] Max D Morris and Toby J. Mitchell. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, 43:381–402, 1992.
- [MTS91] T. Mihalisin, J. Timlin, and J. Schwegler. Visualizing multivariate functions, data, and distributions. *IEEE Computer Graphics and Applications*, 1991.
- [Pac] Stephan Pachnicke. Vorlesung: Faseroptische Nachrichtennetze, TU Dortmund, Lehrstuhl für Hochfrequenztechnik.

- [Pac05] Stephan Pachnicke. *Fast Analytical Assessment of the Signal Quality in Transparent Optical Networks*. PhD thesis, Universität Dortmund, 2005.
- [PAGN07] Anna Persson, Marcus Andersson, Henrik Grimm, and Amos Ng. Metamodel-assisted simulation-based optimization of a real-world manufacturing problem. *Proceedings of the 17th International Conference on Flexible Automation and Intelligent Manufacturing*, June 18-20:950–956, 2007.
- [Pic70] R. M. Pickett. Visual analyses of texture in the detection and recognition of objects. *Picture Processing and Psycho-Pictorics*, Lipkin B. S., Rosenfeld A. (eds.), Academic Press, 1970.
- [Ree79] Trygve Reenskaug. MVC - Model View Controller <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>, 1978-79.
- [Str00] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, 2000.
- [Tra] TransGaming. Cedega. <http://www.transgaming.com/products/cedega/6.0/>.
- [Tro] Trolltech. Qt Cross-Platform Application Framework <http://trolltech.com/products/qt/>.
- [TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
- [VP02] Edgar Voges and Klaus Petermann. *Optische Kommunikationstechnik*. Springer-Verlag, 2002.
- [vR] Guido van Rossum. Python. <http://www.python.org/>.
- [WE06] P.J. Winzer and R.-J. Essiambre. Advanced optical modulation formats. *Proceedings of the IEEE*, 94:952–985, 2006.