Fachhochschule
Münster University of
Applied Sciences

Studienarbeit

Modenanpassung von Gaußstrahlen durch einen nichtlinearen Optimierungsalgorithmus

Clemens Schäfermeier 25. Februar 2014

Betreut von Prof. Dr. J. Nellessen

Zusammenfassung

Von den Grundlagen der Laserphysik bis hin zur Lasermaterialbearbeitung findet das Modell des Gaußstrahls vielfältige Anwendung. Dieses beschreibt, als physikalisch sinnvolles und mathematisch simples Modell, die Strahlpropagation eines Lasers. Die Berechnung zur Auslegung von optischen Systemen nutzt ebenfalls den Gaußstrahl als Modell, da dessen Eigenschaften mit Rechnungen aus der geometrischen Optik verknüpft werden können.

In der Praxis ist vor allem der Ort sowie die Größe der Strahltaille des Gaußstrahls von Interesse. Um diese Parameter an einen dafür vorgesehenen Ort zu transferieren, werden zusätzliche optische Elemente eingesetzt. Diese Elemente verändern beide Parameter gleichermaßen.

Nun ist es mit Rechenmethoden möglich, die Wirkung eines jeden optischen Elements auf den Gaußstrahl und damit seine Parameter zu beschreiben. Mit steigender Anzahl an optischen Elementen erhöht sich neben dem Rechenaufwand auch die Schwierigkeit, die Position der resultierenden Strahltaille mit dem vorgesehenen Ort zu verknüpfen. Diese Schwierigkeit einer symbolischen Lösung kann mit Hilfe von numerischen Optimierungsalgorithmen umgangen werden.

Die vorliegende Studienarbeit soll, mit Hilfe eines neuartigen Algorithmus der nichtlinearen Optimierung, die Auslegung von optischen Systemen zur Anpassung der Gaußstrahlpropagation ermöglichen.

In den ersten Abschnitten werden die Grundlagen und deren Verbindungsmöglichkeit beschrieben, im dritten Abschnitt schließlich der Anwendungsfall mit Ergebnissen erläutert.

Inhalt

1 Grundlagen der Gaußstrahlpropagation 3
1.1 Das Gaußstrahl Modell . . . 3
1.2 Der komplexe Strahlparameter . . . 3
1.2.1 Der Modenbegriff im Zusammenhang mit Laserstrahlung . . . 5
1.3 ABCD Matrix Formalismus . . . 5

2 Nichtlineare Optimierung 8
2.1 Prinzip und Anwendung . . . 8
2.2 Das Extremwertproblem . . . 9
2.3 Algorithmen der nichtlinearen Optimierung . . . 10
2.4 Verwendeter Optimierungsalgorithmus . . . 11

3 Praxisnahe Auslegung eines optischen Systems durch nichtlineare Optimierung 12
3.1 Resultate . . . 13
3.1.1 Variation des Zielwertes der Strahltaille . . . 13
3.1.2 Variation der Linsenbrennweiten . . . 13
3.1.3 Auswahl von Linsenbrennweiten unter Variation der Abstände . . . 15
3.1.4 Erhöhung der Linsenanzahl . . . 16
3.2 Fazit . . . 17

A Berechnung der Gaußstrahlpropagation in `C` 18

B Hilfsmittel und Eigenarbeit 26

Literatur 27

Bisherige Ausgaben

15. November 2011

Erste Ausgabe

26. November 2011

Behebung von Fehlern der Rechtschreibung

Korrektur der Transportmatrix M_T in Gl. (12)

Im `C` Programm (Anhang A): Hinzufügung der Funktion `thick_p_ck_lens`

17. Dezember 2011

Einige Veränderungen und Ergänzungen der Textsetzung; u.a. Hinzufügung der Revisionen

Im `C` Programm (Anhang A): Überarbeitung der Funktionen `qtrans` und `refraction`; Korrektur der Funktion `eigenval_res`; Hinzufügung der Funktionen `grin` und `eigenmod_res`

27. Dezember 2011

Korrektur vom rechten Graph sowie der Bildunterschrift aus Abb. 6 auf Seite 8

1 Grundlagen der Gaußstrahlpropagation

1.1 Das Gaußstrahl Modell

Bei der Beschreibung der Ausbreitung von Laserstrahlung stellt der Gaußstrahl ein grundlegendes Modell dar, was sowohl auf dessen mathematischen Ausdruck als auch auf dessen physikalische Implikation zutrifft. Im Gegensatz zur ebenen Welle, die eine transversal konstante Amplitude darstellt, verfügt der Gaußstrahl über eine endliche Energie – er ist somit eine physikalisch sinnvolle(re) Beschreibung. Die mathematische Beschreibung dieser Lichtausbreitung erhält man durch das Lösen der paraxialen Wellengleichung. Diese ist eine Ableitung aus den Maxwell Gleichungen und gilt für eine Betrachtung der Strahlung im ladungsfreien und uniformen Raum. Daraus lässt sich die Wellengleichung $(\nabla^2 + k^2)E(x, y, z)\,\mathrm{e}^{-\imath kz} = 0$ entwickeln. Für die *paraxiale* Wellengleichung gilt

$$\frac{\partial^2 E}{\partial x^2} + \frac{\partial^2 E}{\partial y^2} - 2\imath k\frac{\partial E}{\partial z} = 0, \tag{1}$$

wobei k die Wellenzahl $2\pi/\lambda$ und E das skalare elektrische Feld (fortan als E-Feld bezeichnet) im 3-dimensionalen Raum symbolisiert [22]. Der Zusatz 'paraxial' bezeichnet die Vernachlässigung der longitudinalen Krümmung des E-Felds, ausgedrückt durch

$$\left|2\imath k\frac{\partial E}{\partial z}\right| \gg \left|\frac{\partial^2 E}{\partial z^2}\right|, \quad \left|\frac{\partial^2 E}{\partial x^2}\right| \gg \left|\frac{\partial^2 E}{\partial z^2}\right| \quad \text{und} \quad \left|\frac{\partial^2 E}{\partial y^2}\right| \gg \left|\frac{\partial^2 E}{\partial z^2}\right|. \tag{2}$$

Eine Gleichung, welche die paraxiale Wellengleichung erfüllt, ist eine Gaußfunktion – daher die Bezeichung des Gaußstrahls. Die Gleichung

$$E(x, y, z) = \frac{\sqrt{2}}{w(z)\sqrt{\pi}} \exp\left[-\left(\frac{x^2 + y^2}{w(z)^2}\right) - \imath\left(k\frac{x^2 + y^2}{2R(z)} + kz - \eta(z)\right)\right] \tag{3}$$

beschreibt den räumlichen Verlauf des Strahls. Hierbei dient der Faktor vor der Exponentialfunktion zur Normierung, die Funktion $w(z)$ ist die Entwicklung des Strahlradius, der mit einem Abfall der Amplitude auf $1/\mathrm{e} \approx 37\ \%$ verbunden ist. Die Phase enthält den Krümmungsradius $R(z)$ und die Gouy Phase $\eta(z)$. Letztere sorgt, beim Vergleich von einem Gaußstrahl zu einer Kugelwelle, für eine Verzögerung der Wellenfront um maximal $\pi/2$ [18]. Die Funktionen lauten, beginnend mit der Rayleigh-Länge z_R,

$$z_R = \frac{\pi w_0^2}{\lambda}, \text{ wobei } w_0 \text{ die Strahltaille ist,} \tag{4a}$$

$$w(z) = w_0\sqrt{1 + \left(\frac{z}{z_R}\right)^2} \tag{4b}$$

$$R(z) = z + \frac{z_R^2}{z} \tag{4c}$$

$$\eta(z) = \arctan\frac{z}{z_R}. \tag{4d}$$

Somit genügt die Kenntnis von λ, w_0 und dessen Position auf der optischen Achse zur Beschreibung der Propagation. Eine Skizze zu den Parametern ist in Abb. 1 zu finden.

1.2 Der komplexe Strahlparameter

Die Einführung des komplexen Strahlparameters

$$\tilde{q}(z) = z + \imath z_R \tag{5}$$

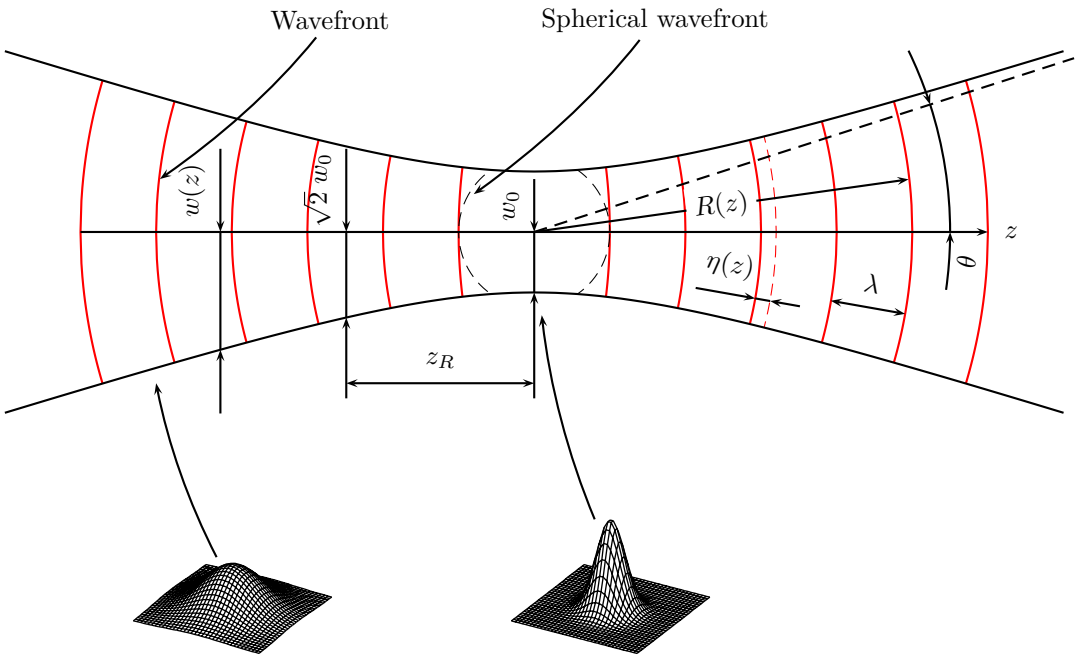

Abb. 1: Veranschaulichung der Strahlpropagtion. Beispielhaft sind 2 normierte E-Felder gezeichnet. Der Verlauf des Strahlradius $w(z)$ ist mit einem Abfall der Amplitude auf $1/\mathrm{e} \approx 37\ \%$ verknüpft. Die eingezeichnete Wellenfront einer Kugelwelle besitzt die Wellenzahl des Gaußstrahls, verfügt allerdings nicht über dessen Gouy Phase η.

und dessen Reziprok

$$\frac{1}{\tilde{q}(z)} = \frac{1}{z + \imath z_R} = \frac{1}{z + z_R^2/z} - \imath \frac{1}{z^2/z_R + z_R} = \frac{1}{R(z)} - \imath \frac{\lambda}{\pi w(z)^2} \tag{6}$$

gestatten eine verkürzte Schreibweise von Gl. (3)

$$E(x,y,z) = \sqrt{-\frac{2}{\lambda}\Im\left(\frac{1}{\tilde{q}(z)}\right)}\, \exp\left[-\imath\left(k\frac{x^2+y^2}{2\tilde{q}(z)} + kz - \eta(z)\right)\right] \tag{7}$$

und damit die Reduzierung zu *einem* komplexen Strahlparameter, auf dem die Propagation des Gaußstrahl basiert.

Ausgehend von der Kenntnis über einen beliebigen Zahlenwert $\tilde{q} \in \mathbb{C},\ \Im(\tilde{q}) > 0$ sollen die Beziehungen zwischen Gln. (5) und (6) der Übersicht halber noch erwähnt werden:

$$z = \Re(\tilde{q}) \tag{8a}$$

$$z_R = \Im(\tilde{q}) \tag{8b}$$

$$w_0 = \sqrt{\frac{\lambda}{\pi}\Im(\tilde{q})} \tag{8c}$$

$$R(z) = \Re(\tilde{q}) + \frac{\Im(\tilde{q})^2}{\Re(\tilde{q})} \tag{8d}$$

$$w(z) = \sqrt{\frac{\lambda}{\pi}\left(\Im(\tilde{q}) + \frac{\Re(\tilde{q})^2}{\Im(\tilde{q})}\right)} \tag{8e}$$

$$\eta(z) = \arctan \frac{\Re(\tilde{q})}{\Im(\tilde{q})} \tag{8f}$$

$$\theta = \arctan \sqrt{\frac{\lambda}{\pi} \frac{1}{\Im(\tilde{q})}}, \text{ wobei } \theta \text{ als Fernfelddivergenz bezeichnet wird.} \tag{8g}$$

Die Formeln wurden für dieses Projekt als Funktionen in die Programmiersprache `C` übersetzt. Neben weiteren Funktionen zur Bestimmung der Stabilität und Eigenmoden von optischen Resonatoren findet der geneigte Leser ein komplettes Programm im Anhang A vor.

1.2.1 Der Modenbegriff im Zusammenhang mit Laserstrahlung

Zusätzlich dem Gaußstrahl erfüllen weitere Funktionen die paraxiale Wellengleichung. Dazu zählen 2 Mengen von orthogonalen Funktionen; sie werden durch eine Multiplikation von Hermite-, respektive Laguerre-Polynomen an die Gaußfunktion erzeugt. Der Gaußstrahl stellt in beiden Fällen den sogenannten 'Grundmode' dieser transversal elektrischen Moden dar. Physikalisch sind die Moden durch die Resonatorsymmetrie (allgemeiner: durch die Symmetrie des durchlaufenen optischen Systems) zu erklären. Als Superposition der transversal elektrischen Moden (TEM) lassen sich beliebige Laserstrahlen beschreiben[1]. Die bisher erfolgten Formel gelten allerdings nur für einmodige TEM Strahlen; die skalaren E-Feld Funktionen insbesondere nur für den Gaußstrahl.

In dieser Projektarbeit bezieht sich der Begriff der 'Mode' *nur* auf den Gaußstrahl und dessen Strahlparameter $\tilde{q}(z)$. Eine diskrete "Gaußmode" ist daher formal mit einem diskreten $\tilde{q}$ gleichzusetzen. Eine physikalische Vorstellung erhält man am einfachsten im Zusammenhang mit optischen Resonatoren: Stellt man sich einen Resonator mit 2 identischen, konkaven Spiegeln vor, so bestimmt deren Abstand und Krümmung die Gaußmode. Die Krümmung $R(z)$ der Mode passt sich gewissermaßen der Spiegelkrümmung an – es handelt sich somit um eine longitudinale Charakteristik des Gaußstrahls.

1.3 ABCD Matrix Formalismus

Wird ein Laserstrahl als rein geometrischer 'Strahl' betrachtet, genügt das Snelliussche Gesetz

$$n_1 \sin(\alpha_1) = n_2 \sin(\alpha_2), \tag{9}$$

möchte man die Propagation durch eine Reihe von optischen Elementen beschreiben [8]. Allgemeiner ist es in Vektorform darstellbar, so dass nicht nur ebene Strahlen verfolgt werden können:

$$n_1(\vec{o} \times \vec{s_\mathrm{i}}) = n_2(\vec{o} \times \vec{s_\mathrm{t}}) \tag{10}$$

Die Brechungsindizes sind als n gekennzeichnet. In der skalaren Form genügt der Winkel α zur Oberflächennormalen, in Vektorform wird der Normalenvektor $\vec{o}$ sowie der Strahlvektor $\vec{s}$ zur Berechnung dieser Invarianten benötigt. Die Bezeichnungen sind in Abb. 2 zusammengefasst und in Abb. 3 für einige Strahlen numerisch durchgespielt.

Bewegt man sich bei der Strahlverfolgung in einem Winkelbereich von circa $|\alpha| \leq 30°$, so lässt sich, auf Kosten eines maximalen Fehlers von 5 %, $\sin(x)$ mit x approximieren – die bezeichnete paraxiale Näherung. Der Numerik erspart dies einige Operationen, die relative Zeitdifferenz liegt je nach Implementation bei circa 85 %. Desweiteren führt die Näherung auf den ABCD Matrix Formalismus, der die analytische Geometrie als Hilfsmittel zur Strahlpropagation nutzt. Ein geometrischer Strahl, beschrieben durch eine Höhe x und einen Winkel β, wird dabei mittels einer Matrix M transformiert und kann so durch ein optisches Element 'propagiert' werden:

$$\begin{pmatrix} x_1 \\ \beta_0 \end{pmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{pmatrix} x_0 \\ \beta_0 \end{pmatrix} = M \cdot \begin{pmatrix} x_0 \\ \beta_0 \end{pmatrix} \quad \text{mit} \quad \det(M) = 1 \tag{11}$$

[1] Anzumerken sei, dass jede Mode in der Superposition aufgrund der Gouy Phase eine leicht unterschiedliche Eigenfrequenz besitzt.

Abb. 4 soll dies veranschaulichen. Prinzipiell genügt die Matrix zur Translation M_T und die Matrix zur Brechung M_B an einer sphärischen Fläche mit Radius R

$$M_T = \begin{bmatrix} 1 & d \\ 0 & 1 \end{bmatrix}, \quad M_B = \begin{bmatrix} 1 & 0 \\ \frac{n_1 - n_2}{n_2 R} & n_1/n_2 \end{bmatrix}, \tag{12}$$

um die 'Strahlformung' vieler gebräuchlicher Elemente entlang der optischen Achse zu beschreiben. Eleganter sind hingegen angepasste Matrizen, die typische Elemente (dicke, dünne Linse etc.) zusammenfassen. Ausführliche Listen solcher Matrizen sind in einem Großteil der Literatur [8,13,18,22] zur technischen Optik zu finden[2]. Weiterhin besteht die Möglichkeit, nicht-paraxiale Optik sowie Strahlen, die nicht in der meridionalen Ebene liegen, mittels Matrizen zu berechnen. Die Lösungen sind dann mit dem klassischen Ray-Tracing vergleichbar, für die Beschreibung sind nunmehr 3 Raumwinkel zu betrachten und keine Näherungen der Winkelfunktionen gültig [15,16].

Kehrt man wieder zum Gaußstrahl zurück, besteht zunächst keine offensichtliche Verbindung zum ABCD Matrix Formalismus. Diese soll nun anhand eines Beispiels skizziert werden: Trifft ein Gaußstrahl auf eine plan-konvex Linse mit Radius R, Brennweite f und Brechungsindex n (siehe Abb. 5), so induziert die Linse einen Phasenterm ϕ_L von

$$\phi_L(d, n, R) = kd + k(n-1)\left(d - \frac{x^2 + y^2}{2R}\right) \propto \frac{1}{f}. \tag{13}$$

In der Herleitung wurde eine Näherung verwendet, um die Wurzelfunktion als Polynom zweiten Grades darzustellen.

In dieser 'Wellen-Betrachtung' muss die Phase $\imath\phi_L$ zur Phase des einfallenden Strahls addiert – oder durch $\mathrm{e}^{\imath\phi_L}$ an das E-Feld multipliziert – werden, um die Wirkung der Linse einzubringen.

Die Matrix, die in der geometrischen Optik die Transformation durch eine Linse beschreibt, enthält, wie auch Gl. (13), die Parameter d, n und R. Die Pfeilhöhe x und der Winkel β aus der geometrischen Betrachtung finden somit ihr 'Wellen-Equivalent' in der induzierten Phase ϕ_L. Durch einen Vergleich der Phase vor und hinter einer Linse bestimmt sich schließlich der funktionale Zusammenhang $\mathcal{T}$[3] zwischen den Einträgen der ABCD Matrix und dem $\tilde{q}$-Parameter. Die Herleitung, die dem Leser hier motiviert werden möchte, ist von A. Yariv bewiesen [22].

Die Abbildung $\mathcal{T}$ lautet

$$\mathcal{T} : \tilde{q} \mapsto \frac{A\tilde{q} + B}{C\tilde{q} + D}. \tag{14}$$

Eine interessante Verbindung besteht zur Funktionentheorie: Die Abbildung $\mathcal{T}$ ist auch als Möbiustransformation bekannt [4]. Für diese gilt allgemein $A, B, C, D \in \mathbb{C}$ und $AD - CB \neq 0$, als Einträge einer Matrix M also $\det(M) \neq 0$.

Die Verbindung zur Physik wird ersichtlich, bedenkt man die Eigenschaften der Möbiustransformation. Sie vereinigt simplere Transformationen, nämlich die der

- Translation
- Inversion beziehungsweise Spiegelung
- Rotation
- Streckung

[2] Wichtig ist hierbei, die in der Literatur auftretenden Formulierungen des Strahlvektors sowie der elementaren Matrizen M_T und M_B zu beachten. Die Ansätze sind teilweise nicht zueinander kompatibel, so müssen beispielsweise die Matrizen im Buch *Optics* [8] von E. Hecht transponiert werden, wenn man sie mit den Matrizen aus A. Yarivs *Quantum Electronics* [22] kombinieren möchte. In A. Siegmans *Lasers* [18] ist der Brechungsindex in die Transportmatrix eingefügt; letztere Autoren umgehen dies durch eine andere Formulierung der Brechungsmatrix.

[3] Im mathematischen Sinne vielmehr eine Abbildung.

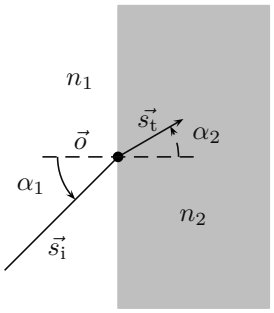

Abb. 2: Das Snelliussche Gesetz anhand einer Grenzfläche mit $n_1 < n_2$ veranschaulicht. Das Gesetz kann über verschiedene Konzepte hergeleitet werden, vom Fermat'schen Prinzip bis zur Quantenelektrodynamik [3].

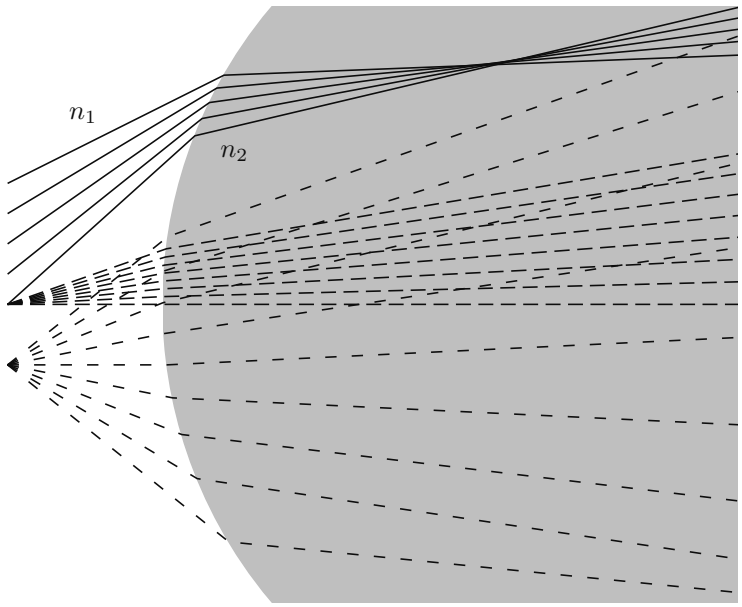

Abb. 3: Anwendung des Snelliusschen Gesetzes auf einer sphärischen Grenzfläche mit $n_1 = 1$ und $n_2 = 1.57$.

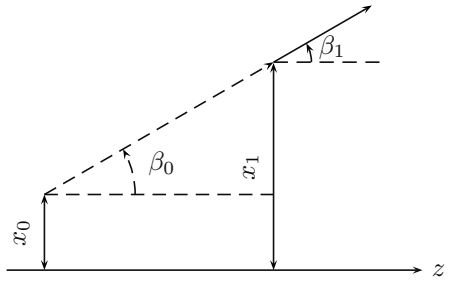

Abb. 4: Die Notation zur Anwendung des ABCD Matrix Formalismus.

und beschreibt damit alle möglichen Einflüsse, die ein optisches Element auf den $\tilde{q}$-Parameter ausüben kann. Da die Matrix Einträge hier allesamt reell sind, ist aus der obigen Aufzählung die Rotation auszunehmen. Bei einer reinen Translation ($C = 0$ bei der M_T Matrix) tritt zudem keine Inversion auf. Die quantitative Wirkung einer dünnen Linse ist in Abb. 6 genauer beschrieben.

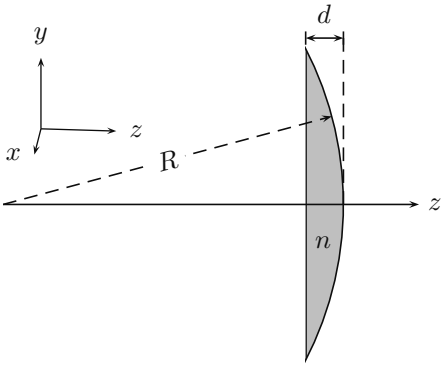

Abb. 5: Die Notation zur Berechnung der Propagation eines Gaußstrahls durch eine plan-konvex Linse.

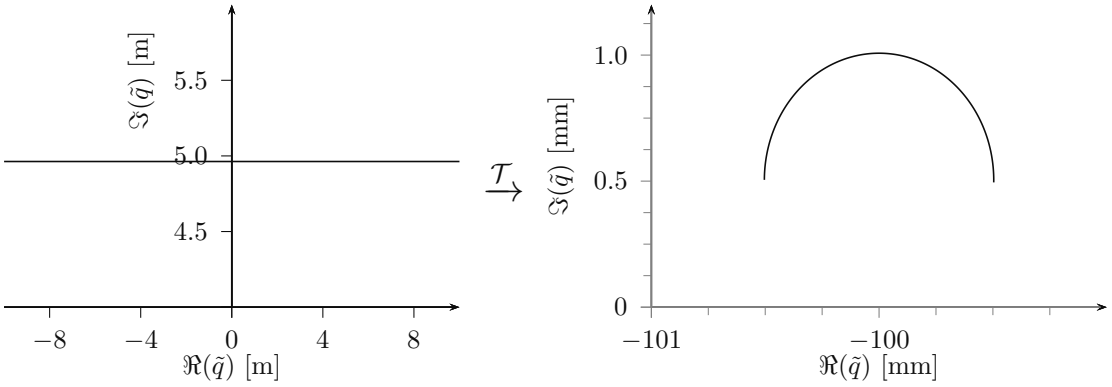

Abb. 6: Die Wirkung einer dünnen Linse (Brennweite $f = 100$ mm) auf einen Gaußstrahl, veranschaulicht anhand dessen $\tilde{q}$-Parameter. Die Matrixeinträge lauten $A = 1, B = 0, C = -1/f$, $D = 1$. Der simulierte Gaußstrahl besitzt eine Taille von 1 mm bei einer Wellenlänge von 632 nm (HeNe-Laser). Im linken Graph ist der 'rohe' $\tilde{q}$-Parameter abgebildet. Die Rayleigh-Länge beträgt konstant 4.9 m, wie es an der imaginären Achse abzulesen ist. Die reale Achse stellt mögliche Lagen der Taille dar: Bei $\Re(\tilde{q}) = 0$ liegt diese genau 'in' der Linse, ist $\Re(\tilde{q}) < 0$, befindet sie sich *rechts* davon. Wenn also $\Re(\tilde{q}) = 100$ mm lautet, ist die Strahltaille im vorderen Fokus (d.h. links der Linse) zu finden. Durch die Wirkung der Linse, beschrieben mit der Abbildung $\mathcal{T}$, wird der $\tilde{q}$-Parameter in den rechten Graph transformiert. Diesem ist zu entnehmen, dass die Strahltaille um höchstens $\pm$ 0.5 mm vom hinteren Fokus abweicht. Die Rayleigh-Länge wird durch die Fokussierung auf einen Wert $\lessapprox 1$ mm reduziert. Die Strahltaille ist in 3 Fällen genau im hinteren Fokus der Linse zu finden: Wenn vor der Linse $\Re(\tilde{q}) \in \{f, \pm\infty\}$ gilt – in jenen Fällen ist die Wellenfront, die sich an der Position des vorderen Fokus befindet, eben. Demzufolge ist dies eine Abbildung, welche die Wellenfront symmetrisch zur Linsenposition transformiert.

2 Nichtlineare Optimierung

2.1 Prinzip und Anwendung

Die Aufgabenstellung der Optimierung ist das Lösen von

$$\min_{\mathbf{v}} f(\mathbf{v}), \tag{15}$$

wobei $\mathbf{v} = (v_1, v_2, \dots v_m) \in V,\ V \subset \mathbb{R}^m$ und $f : V \to \mathbb{R}_0^+$. Weiterhin können Grenzen

$$\mathbf{b}_{\min} \in V,\ \mathbf{b}_{\max} \in V, \quad b_{\max\, i} > b_{\min\, i} \quad \text{mit} \quad i = 1, 2, \dots m \tag{16}$$

für die einzelnen Parameter aus $\mathbf{v}$ definiert sein. So können beispielsweise mechanische Grenzen des speziellen Anwendungsfalles in die Optimierung einbezogen werden. Wird f quadriert,

so ist dieses Verfahren unter dem Synonym “Methode der kleinsten Quadrate” bekannt. Der Funktionswert von f stellt das Gütemaß[4] χ der gefundenen Menge $\mathbf{v}$ dar.

Eine weitere Rolle nimmt die Abbruchbedingung ein. Sie bestimmt, wann die Evaluation von f abbricht. Wird $\chi = 0$ irgendwo durch ein $\mathbf{v}$ erreicht, so ist das Abbrechen bei der kleinsten, von 0 verschiedenen, Zahl akzeptabel. Da das Erreichen von $\chi = 0$ oft nicht absehbar ist, reicht diese Bedingung allein nicht aus – eine Endlosschleife wäre die Folge. Abhilfe bietet der geometrische Abstand zwischen zwei möglichen Minima: Ist dieser kleiner als die Maschinengenauigkeit (engl. “machine epsilon”), ist die Bedingung erfüllt und die Evaluation bricht ab.

Wird das Lösen von Gl. (15) mit der Suche nach einem bestimmten Wert verbunden, so ist es gängig, neben den Grenzen $\mathbf{b}_{\min}$, $\mathbf{b}_{\max}$ noch eine Zielwertmenge $\{t_1, t_2, \dots t_n\}$ einzuführen. Liegen mehrere Zielwerte vor, so können diese mit den Faktoren $g_1, g_2, \dots g_n$ unterschiedlich gewichtet werden. Die Formulierung geschieht dann folgendermaßen:

$$\min_{\mathbf{v}} \; g_1 \left(t_1 - f_{t_1}(\mathbf{v})\right)^2 + g_2 \left(t_2 - f_{t_2}(\mathbf{v})\right)^2 + \cdots \tag{17}$$

Die Indizes von f sollen verdeutlichen, dass hier nicht mehr $f : V \to \mathbb{R}_0^+$ gilt, die Funktion f vielmehr eine n-fache Menge von ‘Rückgabewerten’ besitzt.

Es besteht eine Vielzahl von praktischen Anwendungen, unter anderem die Parameterschätzung zur Anpassung von Funktionen an Messwerte, wie auch das Auffinden von optimalen Prozessparametern in der Industrie. In den Beispielen ist kein geschlossener algebraischer Ausdruck des Problems vorhanden, sodass numerische Verfahren zum Einsatz kommen müssen. Möchte man beispielsweise eine Reihe von Messwerten I, aufgenommen an den diskreten Punkten $x_i, i \in \mathbb{N}^+$, mit einer Geradengleichung beschreiben, so muss

$$\chi^2 = \sum_i \left(I(x_i) - (v_1 x_i + v_2)\right)^2 \tag{18}$$

einen minimalen Wert annehmen. Dies ist die bekannte lineare Regression, die im eigentlichen Sinne kein Optimierungsverfahren darstellt, denn die Parameter v_1 und v_2 lassen sich durch einen geschlossenen Ausdruck berechnen. Auch bei scheinbar nichtlinearen Problemen kann auf die lineare Regression zurückgegriffen werden: Hat man den Strahlradius w an mehreren Stellen z_i aufgenommen, so kann nach der Quadratur

$$w(z_i)^2 = \underbrace{w_0^2}_{v_1} + (z_i - z_0)^2 \underbrace{\left(\frac{\lambda}{\pi w_0}\right)^2}_{v_2} \tag{19}$$

die Strahltaille leicht berechnet werden. Für die Berechnung muss allerdings der Abstand z_0, gemessen zwischen der Position von w_0 und der ersten Position z_1, bekannt sein[5].

Mit der (numerischen) nichtlinearen Optimierung[6] können ebenso symbolische algebraische und transzendente Gleichungen gelöst werden.

2.2 Das Extremwertproblem

Wichtig bei der Bezeichnung des Minimums ist das Attribut lokal, respektive global. Zielgabe sollte es sein, dass $\mathbf{v}_0$ zu finden, für welches $\forall\ \mathbf{v} \in V : f(\mathbf{v}_0) \leq f(\mathbf{v})$ erfüllt ist. Probleme, die einen *Maximal*wert für χ liefern sollen, können allgemein in Minimierungsprobleme umformuliert werden.

[4] Manchmal wird das berechnete Gütemaß auch als Residuum bezeichnet.

[5] Wenn z_0 bekannt ist, könnte natürlich die Messung direkt an dieser Position erfolgen, vorausgesetzt diese ist mechanisch zugänglich. Ist z_0 hingegen unbekannt, müssen w_0 und z_0 mit einem Optimierungsverfahren bestimmt werden. Effizient ist das Vorgehen, z_0 mittels nichtlinearer Optimierung und anschließend w_0 durch lineare Regression zu berechnen.

[6] Im Folgenden wird der Zusatz ‘nichtlinear’ nur genannt, falls die Möglichkeit der Verwechslung zur linearen Optimierung besteht.

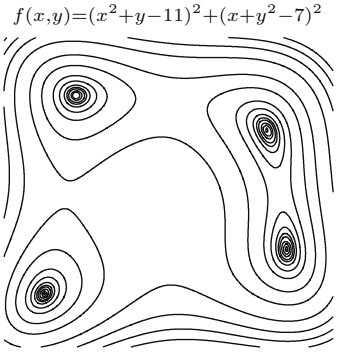

Abb. 7: Die Kontur der Himmelblau Funktion zum Testen von Optimierungsalgorithmen. Sie weist 3 lokale und 1 globales Minimum auf und stellt damit einen konkaven Funktionstyp [20] dar.

Problematisch ist die Klassifizierung des Minimums deshalb, weil bei einem Gebiet, das nur punktweise bekannt ist, kein Entscheidungskriterium für dessen 'Globalität' existiert. Die Berechnung der Hesse-Matrix kann zumindest entscheiden, ob es sich bei einem bestimmten Funktionswert um ein lokales Minimum handelt [2]. Testfunktionen können Optimierungsalgorithmen auf solche Topographien führen, die mehrere lokale Minima besitzen. Ein Beispiel zeigt, rein qualitativ, Abb. 7.

Anhand der beschriebenen Problematik kann eine Kategorisierung von Algorithmen der nichtlinearen Optimierung vorgenommen werden: Die Algorithmen der

- globalen, also möglicherweise unbeschränkten,
- sowie der lokalen

Optimierung.

2.3 Algorithmen der nichtlinearen Optimierung

Als Übersicht soll Tabelle 1 dienen. Die zuletzt genannte Kategorisierung wurde dabei übernommen. Einige der Algorithmen sind, mitsamt Quellcodes in der Programmiersprache `C`, im Buch *Numerical Recipes in C* [17] ausführlich beschrieben.

Weitere Kriterien zur Auswahl einer der Algorithmen sind

- die Robustheit des Algorithmus, also die Eigenschaft, auf Singularitäten, schlecht gewählte Anfangsbedingungen oder Rauschen (entstanden z.B. durch die Aufnahme der Messdaten) ohne Programmabbruch zu reagieren. Beispielsweise können Algorithmen, die die Berechnung des Gradienten durchführen, durch Singularitäten oder Sprungstellen 'fehlgeleitet' werden. Die CMA Evolution Strategy berechnet hingegen keinen Gradienten und schließt so diesen anfälligen Punkt aus.
- der Zeitverbrauch des Algorithmus, insbesondere bei hochdimensionalen Problemen. Diese "Skalierbarkeit" kann bei ungünstig gewählten, dass heißt zu 'strengen', Abbruchbedingungen dazu führen, dass der Algorithmus aufgrund Zufallspermutationen viele Schleifen durchführt, die den Algorithmus bis zur Unpraktikabilität verlangsamen. Algorithmen wie das Simulated Annealing können durch ihre Zufallsprozesse betroffen sein.
- die Anzahl von durchgeführten Evaluationen von $f(\mathbf{v})$ bis zum Erreichen der Abbruchbedingung. In der Regel ist die Rechenzeit für den Erhalt von χ deutlich höher als jene des eigentlichen Algorithmus. So kann, bei einer Parameterschätzung zur Anpassung von Funktionen an Messwerte, die Evaluation von $f(\mathbf{v})$ mehr als 90 % der Gesamtrechenzeit benötigen.

In der Praxis hat es sich durchgesetzt, mehrere Algorithmen zu kombinieren. So kann der Nelder-Mead Simplex mit der Idee des Simulated Annealing verknüpft werden. Allgemein können lokale Verfahren, nach einer geeigneten Schrittweite, ein globales Verfahren ablösen.

lokale Optimierung	globale Optimierung
Nelder-Mead Simplex [14]	CMA Evolution Strategy [7]
Levenberg-Marquardt Algorithmus [11]	Direct Search Simulated Annealing [9]
Gauß-Newton Algorithmus	Particle Swarm Optimizers [10]
	Controlled Random Search
	genetische Algorithmen

Tabelle 1: Übersicht einiger bekannter Optimierungsalgorithmen. Soweit offene Quellen gefunden, sind entsprechende Literatureinträge verzeichnet.

```
long_p *fit=alloc_long_p(2); //Allokiert Speicher

strncpy(fit[0].name,"Linse␣1",7);    // Name des Parameters
strncpy(fit[0].unit,"[mm]",4);       // Einheit des Parameters
fit[0].init=80;        // Initialwert
fit[0].min=-100.;      // Untere Grenze
fit[0].max=120.;       // Obere Grenze
fit[0].log=0;          // Logarithmischer Modus
fit[0].fit=1;          // Entscheidet, ob der Parameter konstant oder variabel ist

strncpy(fit[1].name,"Linse␣2",7);
strncpy(fit[1].unit,"[mm]",4);
fit[1].init=100.;
fit[1].min=0.;
fit[1].max=1000.;
fit[1].log=0;
fit[1].fit=1;
```

Quelltext 1: Beispiel einer Nutzereingabe zur Bedienung des Optimierungsalgorithmus. Der erste Schritt ist die Eingabe der (variablen) Parameter.

2.4 Verwendeter Optimierungsalgorithmus

Der in diesem Projekt verwendete Algorithmus ist bis zum heutigen Datum nicht veröffentlicht. Er basiert auf der Berechnung von Momenten, um Vorteile globaler und lokaler Optimierung zu vereinen. Um die Topographie von $f(\mathbf{v})$ möglichst gleichverteilt zu untersuchen, wurde ein effektiver Pseudo-Zufallszahlen Generator [12] implementiert. Weitere Permutationen produziert ein Algorithmus, der multidimensionale Rotationsmatrizen erzeugt. Dieser wurde von mir anhand einer Veröffentlichung [1] in `C` übersetzt. Bei 2-dimensionalen Funktionen kann der Anwender direkt eine graphische Ausgabe durch `gnuplot` [6] erzeugen, was in erster Linie dem Überprüfen der Lösung dient oder zum Testen verwendet werden kann.

Um die bisherige Erläuterungen zu veranschaulichen, wird die Anwendung des Algorithmus nun ‘programmatisch’ beschrieben: Beispielsweise möchte der potentielle Anwender die Strahltaille w_0 und deren Position z (siehe Abb. 1) durch Wahl der Brennweiten von 2 Linsen an gegebene Zielwerte anpassen.

Zunächst ist dafür die Definition von Grenzen, einem Initialwert und optionalen Einstellungen nötig. Im Quelltext 1 ist dies beschrieben.

Danach sind noch die mit g_1, g_2 gewichteten Zielwerte t_1, t_2 einzugeben. Diese werden der Funktion, die χ als Rückgabewert hat, übergeben. Falls keine Zielwerte erwünscht sind, muss mindestens ein Zielwert vorgegeben werden. Der Wert muss dann nicht definiert werden, da dies beim Allokieren geschieht. In diesem Beispiel soll die Strahltaille w_0 und deren Position z entlang

```
target *tt=alloc_target(2); //Allokiert Speicher

tt[0].val=10.; // Zielwert t_1 für w_0
tt[0].wgt=.8;  // Wichtung g_1 für w_0

tt[1].val=0.5; // Zielwert t_2 für z
tt[1].wgt=.2;  // Wichtung g_2 für z
```

Quelltext 2: Die Definition von gewichtete Zielwerten, die an den Optimierungsalgorithmus weitergegeben werden.

```
double minime(void (*f) (double *, double *, const target *), long_p *ips, int
    nparm, int ntry, unsigned char chatty, const target *ctv)

minime(ray_prop,fit,2,11,1,tt);

/*
        *f: Zeiger auf die Funktion f(v)
        *ips: Zeiger auf die (variablen) Parameter aus Quelltext 1
        nparm: Anzahl der Parameter
        ntry: Anzahl von Versuchen pro Parameter
        chatty: In Binärziffer kodiertes Ausgabeformat
        *ctv: Zeiger auf die gewichteten Zielwerte
*/
```

Quelltext 3: Aufruf des Optimierungsalgorithmus. Als Kommentare sind die Übergabeparameter erläutert. Die Deklaration ist zuerst aufgeführt.

der optischen Achse auf die Werte $w_0 = 10$ µm, $z = 0.5$ mm angepasst werden. Die Größe der Strahltaille ist dem Anwender hier wichtiger, daher ist $g_1 > g_2$. Die Definition dieser gewichteten Zielwerte geschieht wie in Quelltext 2.

Zum Schluss ist der Optimierungsalgorithmus aufzurufen. Das Vorgehen, samt dem Funktionsrumpf, ist in Quelltext 3 beschrieben.

3 Praxisnahe Auslegung eines optischen Systems durch nichtlineare Optimierung

Der im Abschnitt 1 beschriebene Zusammenhang zwischen Gaussstrahlen und dem ABCD Matrix Formalismus wurde für dieses Projekt in `C` derart programmiert (siehe Anhang A), dass praktische Aufgabenstellungen bei optischen Experimenten durch einen neuartigen Optimierungsalgorithmus gelöst werden können.

Die Aufgabe ist es, einen Gaußstrahl, mit vorgegebener Größe und Position seiner Strahltaille, durch Linsen an eine entfernte Position – ebenfalls mit vorgegebener Strahltaille – anzupassen. Dies entspricht zahlreichen praxisnahen Anwendungen, die beispielsweise in der Lasermaterialbearbeitung aufkommen: Beim Schweißen zweier Materialien hängt die Qualität der Naht stark von der Lage der Strahltaille relativ zur Position der Materialoberfläche ab. Die Strahllage wird durch Linsen und / oder deren Abstände bestimmt, der Gaußmode des Laserresonators bleibt hingegen konstant.

Im speziellen Fall soll der Gaußstrahl durch den Aufbau aus Abb. 8 transportiert werden. Eine Auflistung der verwendeten Parameter ist in Tabelle 2 zu finden.

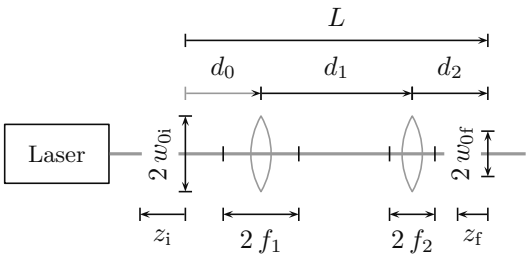

Abb. 8: Eine schematische Darstellung des Aufbaus und die Bezeichnung der Parameter. Zielwert ist die Strahltaille $w_{0\mathrm{f}}$ und deren Position. Da hier das Modell der dünnen Linse Verwendung findet, ist die Brennweite unabhängig von der Betrachtungsrichtung – die Brennweite vor der Linse gleicht jener hinter ihr. Die Pfeilrichtung symbolisiert eine *positive* Länge. Die Längen d_i können, entgegen z_{i} und z_{f}, keine negativen Werte annehmen.

Parameter	Wert	Bemerkung
λ	632.8 nm	konstant
$w_{0\mathrm{i}}$	100 µm	konstant
z_{i}	0 mm	konstant
$w_{0\mathrm{f}}$	40 µm	Zielwert
z_{f}	0 mm	Zielwert
d_0	400 mm	variabel
d_1	1400 mm	variabel
d_2	$\approx$ 172.1 mm	variabel
f_1	258 mm	variabel / konstant
f_2	140.4 mm	variabel / konstant
$d_0 + d_1 + d_2$		konstant

Tabelle 2: Übersicht der Parameter, die der Aufgabenstellung entstammen. Wird die Lage der Strahltaille $w_{0\mathrm{f}}$ vom Scheitel der letzten Linse aus gemessen, spart man eine Transportmatrix M_T. In den folgenden Unterabschnitten wird eine solche Transportmatrix eingebaut, z_{f} hat dann, wie gelistet, 0 mm als Zielwert. Je nach Aufgabenstellung sind die Brennweiten konstant oder variabel.

3.1 Resultate

Zunächst wurden die vorgegebenen Parameter aus Tabelle 2 verifiziert. Die Position d_2 ist um

$$z_{\mathrm{f}} = -0.072 \text{ mm} \tag{20}$$

zu korrigieren, um diese genau mit der Lokalisation der Strahltaille zu überdecken. Dies korrigiert die Gesamtlänge $L = \sum_i d_i$ auf

$$L = d_1 + d_1 + d_2 - z_{\mathrm{f}} = 1972.172 \text{ mm}, \tag{21}$$

wenn ausgehend von der Taillenposition z_{i} gemessen wird.

3.1.1 Variation des Zielwertes der Strahltaille

Um den Optimierungsalgorithmus anfangs mit einer leichten Abweichung zu testen, soll der Aufbau aus Abb. 8 auf eine Strahltaille von $w_{0\mathrm{f}} = 30$ µm verändert werden. Variabel sind die Längen d_0 und d_1. Die Abb. 9 soll die Topologie verdeutlichen, in der sich der Algorithmus ‘bewegt’.

Der Algorithmus lieferte ein Gütemaß von $6.7 \cdot 10^{-28}$, womit eine erfolgreiche Minimierung erzielt wurde. Die Daten fassen Tabellen 3 und 4 zusammen.

3.1.2 Variation der Linsenbrennweiten

Im nächsten Schritt werden die Abstände zwischen den Linsen konstant gehalten, die Brennweiten der Linsen müssen somit variiert werden, um die Zielwerte zu erreichen. Tabelle 5 führt die Werte auf.

Parameter	Wert	Bemerkung
w_{0f}	30 µm	Zielwert
z_f	0 mm	Zielwert
d_2	$L - d_0 - d_1$	Randbedingung
f_1	258 mm	konstant
f_2	140.4 mm	konstant

Tabelle 3: Übersicht der Parameter, die der Algorithmus nicht variierte.

Parameter		Initialwert	Grenzen		Resultat
d_0	[mm]	400	300	500	443.073
d_1	[mm]	1400	1000	1500	1357.590
d_2	[mm]				172.028
Zielwerte					
w_{0f}	[µm]				30.0
z_f	[mm]				$-7.2 \cdot 10^{-11}$
χ	[/]				$6.7 \cdot 10^{-28}$

Tabelle 4: Übersicht der Zielwerte und variierten Parameter, die am Ende des Programms ausgegeben wurden. Die Länge d_2 wird, zur leichteren Kontrolle, immer mitgeschrieben. Sie ergibt sich aus der Konstanz der Gesamtlänge L.

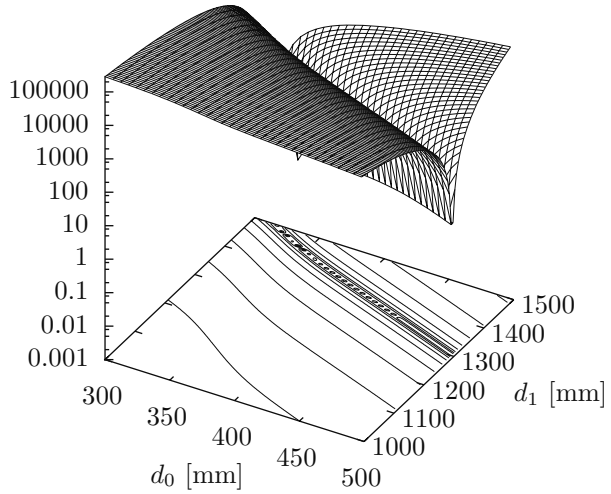

Abb. 9: Die Topographie der zu minimierenden Funktion $f(\mathbf{v})$ in logarithmischer Darstellung. Leicht zu erkennen ist der 'Graben', in dem das Minimum liegt. Es besteht eine große Ähnlichkeit zur Rosenbrock Funktion [21], die zum Testen von Optimierungsalgorithmen genutzt wird. Aufgrund der Länge und 'Flachheit' des Grabens kann die Aufgabe nicht als trivial bezeichnet werden, es liegt hier insbesondere eine konkave Funktion vor.

Parameter	Wert	Bemerkung
w_{0f}	30 µm	Zielwert
z_f	0 mm	Zielwert
d_0	50 mm	konstant
d_1	1600 mm	konstant
d_2	322.028 mm	konstant

Tabelle 5: Übersicht der Konstanten und der Zielwerte.

Auch in diesem Fall passte der Algorithmus die Parameter auf die Zielwerte erfolgreich an, wie ein Gütemaß von $1.1 \cdot 10^{-25}$ bestätigt. In Tabelle 6 sind die Ergebnisse aufgelistet.

Parameter		Initialwert	Grenzen		Resultat
f_1	[mm]	258	50	600	309.510
f_2	[mm]	140.4	50	600	336.076
Zielwerte					
$w_{0\mathrm{f}}$	[µm]				30.0
z_{f}	[mm]				$-1.9 \cdot 10^{-6}$
χ	[/]				$1.1 \cdot 10^{-25}$

Tabelle 6: Übersicht der Programmausgabe mit den variierten Parametern.

```
L ← {400, 200, 150, 100}
n ← #L
l ← 2

for i from 1 to n^l by 1 do
 for j from 1 to l by 1 do

    t ← ⌊i/n^(l−j)⌋ mod n
    f_j ← L_t

 end
end
```

Quelltext 4: Syntax zur Permutation mit Beachtung der Reihenfolge. Hier ist n die Anzahl der Elemente aus L und l die Zahl von verwendeten Linsen (Paaren). Die nach unten geschlossenen Klammern (Gaußklammern) symbolisieren die Abrundung.

3.1.3 Auswahl von Linsenbrennweiten unter Variation der Abstände

In einem praktischen Fall könnte die Gesamtlänge L und die Gaußmode vorgegeben sein, offen sind allerdings die zu wählenden Linsen sowie deren Position. In diesem Schritt wird ein solcher Fall nachgespielt: Es sollen 2 Linsen verwendet werden, wobei die Brennweiten 400, 200, 150 und 100 mm verfügbar sind.

Unter der Randbedingung, dass L konstant und eine bestimmte Strahltaille gefordert ist, können bereits ohne Optimierungsalgorithmus Linsenkombinationen ausgeschlossen werden. Bei kurzer Gesamtlänge werden 2 Linsen mit langer Brennweite die geforderte Strahltaille nicht erzielen. Da dieses Ausschlussverfahren bereits mit Matrixmultiplikationen verbunden ist und zudem die Rechenzeit zur Optimierung *einer* Kombination unter 1/2 Sekunden liegt, werden in meinem Programm alle Kombinationen durch Permutation erstellt und zur Optimierung freigegeben. Die Permutation findet der Leser in Quelltext 4. Zwecks Lesbarkeit ist der Code in einer “Pseudosprache” verfasst. Zum Schluss wird das minimale Gütemaß aus den Kombinationen gesucht und dieses, samt Linsen, ausgegeben. Die gesamte Prozedur dauert, je nach Rechenleistung, circa 2 Sekunden.

Gewissermaßen problematisch bei dem Verfahren ist, dass mehrere Kombinationen die Anforderungen aus Tabelle 7 erfüllen. Da der Algorithmus teilweise mit Zufallsprozessen arbeitet, liefert die Optimierung nach mehrmaligen Durchläufen unterschiedliche Kombinationen. Eine bessere Entscheidungsfähigkeit würde weitere Zielwerte, formuliert beispielsweise durch die Minimierung von Aberrationen, benötigen.

Schließlich wählte ich die Kombination $f_1 = 400$ mm, $f_2 = 200$ mm aus. Zur Auswahl suchte ich nach der Kombination mit den größten Brennweiten, um die genannten Aberrationen (sphärische Linsenfehler) gering zu halten. Die Ergebnisse sind in Tabelle 8 aufgelistet.

Parameter	Wert	Bemerkung
w_{0f}	30 µm	Zielwert
z_f	0 mm	Zielwert
f_1	Liste	konstant
f_2	Liste	konstant
d_2	$L - d_0 - d_1$	konstant

Tabelle 7: Zusammenfassung der Vorgaben zur automatischen Linsenwahl.

Parameter		Initialwert	Grenzen		Resultat
d_0	[mm]	100	40	500	281.679
d_1	[mm]	1000	800	1500	1471.628
d_2	[mm]				218.721
Zielwerte					
w_{0f}	[µm]				30.0
z_f	[mm]				$-3.2 \cdot 10^{-13}$
χ	[/]				$2.0 \cdot 10^{-15}$

Tabelle 8: Übersicht der Programmausgabe mit den Brennweiten $f_1 = 400$ mm, $f_2 = 200$ mm. Die Länge d_2 ergibt sich mit der Konstanz von L, der Gesamtlänge des Systems.

3.1.4 Erhöhung der Linsenanzahl

Nachdem die 2-dimensionalen Optimierungsversuche erfolgreich verliefen, wird nun eine weitere Dimension durch eine dritte Linse hinzugefügt. Den Aufbau skizziert Abb. 10.

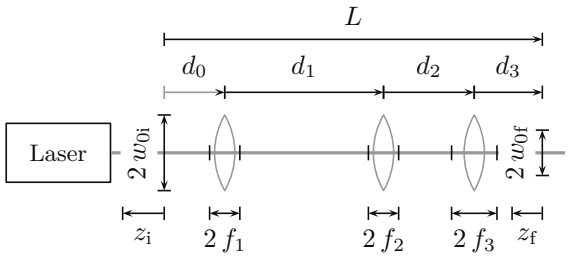

Abb. 10: Eine schematische Darstellung des erweiterten Aufbaus aus Abb. 8.

Wie im vorigen Unterabschnitt 3.1.3 implementierte ich für diese Aufgabe eine Permutation. Somit werden $4^3 = 64$ Linsenkombinationen erstellt und auf die Zielwerte aus Tabelle 9 optimiert. Die Ausführung stellte heraus, das 30 von den 64 Kombinationen die Zielwerte nicht erreichen.

Parameter	Wert	Bemerkung
w_{0f}	30 µm	Zielwert
z_f	0 mm	Zielwert
f_1	Liste	konstant
f_2	Liste	konstant
f_3	Liste	konstant
d_3	$L - \sum_{i=0}^{2} d_i$	konstant

Tabelle 9: Zusammenfassung der Vorgaben zur automatischen Linsenwahl.

Aus den ‘erfolgreichen’ Kombinationen wählte ich die in Tabelle 10 gelisteten Werte, da hierbei die größten Brennweiten zum Einsatz kommen. Wie bereits angeführt, helfen große Brennweiten, dass heißt geringe Brechkräfte, Aberrationen durch Linsenfehler zu minimieren.

Parameter		Initialwert	Grenzen		Resultat
d_0	[mm]	100	8	1000	546.038
d_1	[mm]	100	8	1000	201.918
d_2	[mm]	100	8	1000	930.760
d_3	[mm]				293.312
Zielwerte					
$w_{0\mathrm{f}}$	[µm]				30.0
z_{f}	[mm]				$4.6 \cdot 10^{-12}$
χ	[/]				$2.7 \cdot 10^{-16}$

Tabelle 10: Übersicht der Programmausgabe mit den Brennweiten $f_1 = f_2 = 400$ mm, $f_3 = 200$ mm. Die Länge d_3 ergibt sich aus der Konstanz von L.

Die Ergebnisse nahm ich nun zum Anlass, meinen Optimierungsalgorithmus gegen einen Neld-Mead Simplex[14] zu testen. Den Algorithmus entnahm ich dem Buch *Numerical Recipes in C*[17] und passte diesen auf meine Programmierweise an – die Autoren nutzen eine eher orthodoxe Verwendung von Feldern, um, beispielsweise in Schleifen, den Feldinhalt von 1 an zu adressieren.

Da diese 'Urversion' des Simplex keine Randbedingungen akzeptiert, implementierte ich eine Bestrafungsfunktion direkt in die zu minimierende Funktion $f(\mathbf{v})$. Diese erhöht das Gütemaß χ, sobald die Abstände d (siehe Abb. 10) negative Werte annehmen.

Als Initialwert für den Simplex sind die Werte aus Tabelle 10 übernommen worden. Der Nutzer muss bei der verwendeten Implementation nicht nur einen, sondern $n+1$, $n \in \mathbb{N}^+$ Initialwerte vorgeben, wenn n die Anzahl von variablen Parametern (die Dimensionalität des Problems) angibt. Damit spannt der Benutzer im euklidischen Raum einen Vertex auf, der dann durch den Algorithmus gesteuert wird. Beginnend mit dem Punkt (100,100,100) übergab ich, durch die Wahl von (101,100,100), (100,101,100) und (100,100,101), ein Tetraeder. Die Brennweiten der Linsen wurden nicht permutiert, sie entsprechen den Werten aus Tabelle 10: $f_1 = f_2 = 400$ mm, $f_3 = 200$ mm. Dies ermöglicht eine leichte Vergleichbarkeit der Ansätze.

Der Algorithmus lieferte ein Gütemaß von $\chi = 1.2 \cdot 10^6$ und scheiterte damit an diesem Problem. Dies deutet darauf hin, dass die zu minimierende Funktion $f(\mathbf{v})$ konkav ist, also mindestens ein lokales Minimum aufweist, dass nicht mit dem globalen Minimum koinzidiert.

Probehalber setzte ich neue Initialwerte ein, welche diesmal von den *Resultaten* aus Tabelle 10 um -10 mm abwichen. Unter diesen Bedingungen konvergierte der Simplex Algorithmus zu den bekannten Abständen.

3.2 Fazit

Diese Projektarbeit hat bewiesen, dass der verwendete Optimierungsalgorithmus zur Anpassung einer Gaußmode auf gewünschte Zielwerte erfolgreich verwendet werden kann.

Bei Problemen, in denen der Initialwert des Algorithmus nahe des globalen Minimums gewählt wird, sind lokale Verfahren ebenfalls als nutzbar zu bezeichnen. Mit steigender Dimensionalität der Aufgabenstellung oder uneindeutiger (genauer: konkaver) Topographie der zu minimierenden Funktion sollten hingegen globale Verfahren genutzt werden.

A Berechnung der Gaußstrahlpropagation in C

Die Gaußstrahlpropagation durch optische Elemente wurde von mir für dieses Projekt in C programmiert. Grundlage dafür bildet der paraxiale ABCD Matrix Formalismus und dessen Zusammenhang mit dem $\tilde{q}$-Parameter. Der folgende Quelltext enthält eine Sammlung von ABCD Matrizen, einen Stabilitätstest für optische Resonatoren, eine Berechnung von Eigenmoden ebendieser, alle notwendigen Strukturen samt komplexer Algebra und schließlich ein kurzes Testprogramm in der Hauptfunktion. Zum bequemen Kompilieren findet der Leser auf Seite 25 ein Makefile vor.

Der Quelltext wurde auf Windows und Linux Rechnern mit gcc [5] kompiliert. Weiterhin wurde dieser mit Valgrind [19] auf zahlreiche Speicherfehler geprüft. Falls trotz allem ein Fehler auftritt, bin ich für jede Rückmeldung dankbar. Bei nichtkommerzieller Nutzung stimme ich der Weitergabe ausdrücklich zu.

Falls Ihre Software zur pdf-Wiedergabe es unterstützt, können Sie den Anlagen die beiden Quellen 5 und 6 direkt entnehmen.

```
/*
       file: optic_c.c
       author: clemens schaefermeier
       mail: clemens@fh-muenster.de
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <float.h>
#ifndef M_PI
#define M_PI 3.1415926535897932
#endif

typedef struct
{
double R,I,
      lam;
}qbeam;

typedef struct {double R,I;} cplx;

/* header */
cplx *alloc_cplx(int count);

void thin_lens(double *M, const double f);
void thick_lens(double *M, const double r1, const double r2, const double d,
    const double n);
void thick_p_ck_lens(double *M, const double r2, const double d, const double n
    );
void propagation(double *M, const double d);
void refraction(double *M, const double n1, const double n2, const double r);
void reflection(double *M, const double r);
void grin(double *M, const double d, const double n0, const double nmin);

void M_mult_ip(double *Mr, const double *M);
qbeam qtrans(const double *M, const qbeam *q);
```

```
double wz(const double w0, const double z, const double zr);
double rayleighr(const double w0, const double lam);

double stab_parm(const double *M);
cplx *eigenval_res(const double *M);
cplx eigenmod_res(const double *M);

void setq(qbeam *q, const double z, const double zr, const double lam);
double get_w0(const qbeam *q);
double get_z(const qbeam *q);
double get_zr(const qbeam *q);
double get_beamrad(const qbeam *q);
double get_wz(const qbeam *q);
double get_div(const qbeam *q);

void q_out(const qbeam *q, const char *s);
void q_fout(const qbeam *q, const char *s);

cplx *alloc_cplx(int count)
{
int i;
cplx *m;
m=(cplx *)malloc((count)*sizeof(cplx));
if(NULL==m) {fprintf(stderr,"\nfile:line␣%s:%i\nmemory␣allocation␣failed␣in␣
    alloc_cplx.␣good␣bye.",__FILE__,__LINE__);free(m);exit(EXIT_FAILURE);}
for(i=0;i<count;i++) m[i].R=m[i].I=0.;
return m;
}

void thin_lens(double *M, const double f)
{
M[0]=1.;
M[1]=0.;
M[2]=-1./f;
M[3]=1.;
}

void thick_lens(double *M, const double r1, const double r2, const double d,
    const double n)
{
double ni=1./n,dr1,r1i,r2i;
if(r1!=0.) r1i=1./r1;
else r1i=0.;
dr1=d*r1i;
if(r2!=0.) r2i=1./r2;
else r2i=0.;
M[0]=1.+dr1*(ni-1.);
M[1]=d*ni;
M[2]=(1.-n)*(r1i-r2i)+dr1*r2i*(2.-ni-n);
M[3]=1.-d*r2i*(ni-1.);
}
```

```
void thick_p_ck_lens(double *M, const double r2, const double d, const double n
    )
{
double ni=1./n;
M[0]=1.;
M[1]=d*ni;
M[2]=-(1.-n)/r2;
M[3]=1.-d/r2*(ni-1.);
}

void propagation(double *M, const double d)
{
M[0]=1.;
M[1]=d;
M[2]=0.;
M[3]=1.;
}

void refraction(double *M, const double n1, const double n2, const double r)
{
M[0]=1.;
M[1]=0.;
if(r!=0.) M[2]=(n1-n2)/(n2*r);
else M[2]=0.;
M[3]=n1/n2;
}

void reflection(double *M, const double r)
{
M[0]=1.;
M[1]=0.;
M[2]=-2./r;
M[3]=1.;
}

void grin(double *M, const double d, const double n0, const double nmin)
{
double t1,t2,t3;
t1=sqrt(nmin/n0);
t2=t1*d;
t3=sin(t2);
M[0]=cos(t2);
M[1]=t3/t1;
M[2]=-t1*t3;
M[3]=M[0];
}

void M_mult_ip(double *Mr, const double *M)
{
double t1,t2;
t1=Mr[0]*M[0]+Mr[1]*M[2];
t2=Mr[0]*M[1]+Mr[1]*M[3];
Mr[0]=t1;
```

```
Mr[1]=t2;
t1=Mr[2]*M[0]+Mr[3]*M[2];
t2=Mr[2]*M[1]+Mr[3]*M[3];
Mr[2]=t1;
Mr[3]=t2;
}

qbeam qtrans(const double *M, const qbeam *q)
{
qbeam qp;
qp.lam=(*q).lam;
double t1,t2,t3,t4,t5,denom;

t1=M[2]*(*q).R+M[3];
t2=t1*t1;
t3=M[0]*(*q).R+M[1];
t4=M[2]*M[2];
t5=(*q).I*(*q).I;

if((denom=t2+t4*t5)==0.) denom=1e-15;

qp.R=t1*t3+t5*M[0]*M[2];
qp.R/=denom;
qp.I=(*q).I*(M[0]*t1-M[2]*t3);
qp.I/=denom;
return qp;
}

double wz(const double w0, const double z, const double zr)
{
double t=z/zr;
t*=t;
return w0*sqrt(1.+t);
}

double rayleighr(const double w0, const double lam)
{
return M_PI*w0*w0/lam;
}

double stab_parm(const double *M)
{
return ((M[0]+M[3])/2.);
}

cplx *eigenval_res(const double *M)
{
cplx *c=alloc_cplx(2);
double g=stab_parm(M),t;
t=g*g-1.;
if(t<0.)
{
       t=sqrt(-t);
```

```
        c[0].R=c[1].R=g;
        c[0].I=t;
        c[1].I=-t;
}
else if(t>0.)
{
        t=sqrt(t);
        c[0].I=c[1].I=0.;
        c[0].R=g+t;
        c[1].R=g-t;
}
else
{
        c[0].I=c[1].I=0.;
        c[0].R=c[1].R=g;
}
return c;
}

cplx eigenmod_res(const double *M)
{
cplx q;
double t1,t2,t3;
t1=M[3]-M[0];
if(t1!=0.) t3=t1*t1;
else
{
        q.R=0.;
        if((t3=M[1]/M[2])>=0.)
        {
                fprintf(stdout,"\nfound␣no␣eigenmode␣solution");
                q.I=0.;
        }
        else q.I=sqrt(-t3);
        return q;
}
t2=2.*M[2];
q.R=t1/t2;
t2*=t2;
if((t3=(t3+4.*M[1]*M[2])/t2)>=0.)
{
        fprintf(stdout,"\nfound␣no␣eigenmode␣solution");
        q.R=q.I=0.;
}
else q.I=sqrt(-t3);
return q;
}

void setq(qbeam *q, const double z, const double zr, const double lam)
{
(*q).R=z;
(*q).I=zr;
(*q).lam=lam;
```

```
}

double get_w0(const qbeam *q)
{
return sqrt((*q).I*(*q).lam/M_PI);
}

double get_z(const qbeam *q)
{
return (*q).R;
}

double get_zr(const qbeam *q)
{
return (*q).I;
}

double get_beamrad(const qbeam *q)
{
double r=(*q).R;
if(r!=0.) r+=((*q).I*(*q).I/r);
else return DBL_MAX;
return r;
}

double get_wz(const qbeam *q)
{
double wz=(*q).I;
wz+=((*q).R*(*q).R/(*q).I);
wz*=((*q).lam/M_PI);
wz=sqrt(wz);
return wz;
}

double get_div(const qbeam *q)
{
return atan( sqrt((*q).lam/((*q).I*M_PI)) );
}

void q_out(const qbeam *q, const char *s)
{
double r=get_beamrad(q);
(r==DBL_MAX)?
fprintf(stdout,"q.%-6.6s:␣%12s␣%12s\n%9s␣%12g␣%12g\n"
              "%9s␣%12s␣%12s\n%9s␣%12s␣%12g\n"
             ,s,"z[e-3m]","w0[e-6m]","",get_z(q),get_w0(q)*1e3
             ,"","R[e-3m]","wz[e-6m]","","inf",get_wz(q)*1e3):
fprintf(stdout,"q.%-6.6s:␣%12s␣%12s\n%9s␣%12g␣%12g\n"
              "%9s␣%12s␣%12s\n%9s␣%12g␣%12g\n"
             ,s,"z[e-3m]","w0[e-6m]","",get_z(q),get_w0(q)*1e3
             ,"","R[e-3m]","wz[e-6m]","",r,get_wz(q)*1e3);
}
```

```
void q_fout(const qbeam *q, const char *s)
{
time_t tt;
time(&tt);
FILE *fout;
fout=fopen("q_fout.txt","aw");
if(NULL==fout) {fprintf(stderr,"\nfile:line␣%s:%i\ncan␣not␣open␣'q_fout.txt'",
    __FILE__,__LINE__);fclose(fout);}
double r=get_beamrad(q);
(r==DBL_MAX)?
fprintf(fout,"%s\nq.%-6.6s:␣%12s␣%12s\n%9s␣%12g␣%12g\n"
            "%9s␣%12s␣%12s\n%9s␣%12s␣%12g\n",ctime(&tt)
             ,s,"z[e-3m]","w0[e-6m]","",get_z(q),get_w0(q)*1e3
             ,"","R[e-3m]","wz[e-6m]","","inf",get_wz(q)*1e3):
fprintf(fout,"%s\nq.%-6.6s:␣%12s␣%12s\n%9s␣%12g␣%12g\n"
            "%9s␣%12s␣%12s\n%9s␣%12g␣%12g\n",ctime(&tt)
             ,s,"z[e-3m]","w0[e-6m]","",get_z(q),get_w0(q)*1e3
             ,"","R[e-3m]","wz[e-6m]","",r,get_wz(q)*1e3);
fclose(fout);
}

int main()
{
double lambda=632.8e-6,w0=1e-1,
       M1[4],M2[4];
qbeam q,qp;
setq(&q,0.,rayleighr(w0,lambda),lambda);

q_out(&q,"in");

propagation(M1,400.);
thin_lens(M2,258.);
M_mult_ip(M2,M1);

propagation(M1,1400.);
M_mult_ip(M1,M2);

thin_lens(M2,140.4);
M_mult_ip(M2,M1);

propagation(M1,172.1);
M_mult_ip(M1,M2);

qp=qtrans(M1,&q);

q_out(&qp,"out");

return EXIT_SUCCESS;

/*
the output should be read as follows

q.in   :     z[e-3m]    w0[e-6m]
```

```
                    0         100
               R[e-3m]    wz[e-6m]
                  inf         100
q.out   :      z[e-3m]    w0[e-6m]
           -0.0721956     40.0079
               R[e-3m]    wz[e-6m]
              -874.728     40.0095
*/
}
```

Quelltext 5: Programm zur Berechnung der Gaußstrahlpropagation. Um die Formatierung beim Kopieren zu erhalten, sind Leerzeichen in Zeichenketten durch einen Platzhalter gekennzeichnet.

```
CC = gcc

ifndef DEBUG
DEBUG = 1
endif

ifeq ($(DEBUG),1)
CFLAGS = -O0 -g -Wall -Wextra -pedantic -lm -std=c99
else ifeq ($(DEBUG),0)
CFLAGS = -O3 -lm -std=c99
endif

test: optic_c.c
	$(CC) $(CFLAGS) -o $@ $^
```

Quelltext 6: Makefile zum Kompilieren des obigen Quelltextes 5. Um das Ziel zu erstellen, sollte der Tabulator in der letzten Zeile nach dem Kopieren unbedingt auf Vorhandensein geprüft werden. Soll das Programm nur ausgeführt werden, ist der Befehl `make DEBUG=0` dem Kommandozeileninterpreter zu übergeben.

B Hilfsmittel und Eigenarbeit

Verwendete Hilfsmittel

Die Umsetzung und ständige Überprüfung des Projekts mitsamt aller Quelltexte, Grafiken und schließlich dieses Schriftstücks verdanke ich alleinig dem Vorhandensein von quelloffener Software, die durch öffentliche Lizenzen jedem privaten Nutzer zur Verfügung stehen. In besonderem Maße trug die GNU Compiler Collection [5] zur Entstehung bei.

Klausel zur Eigenarbeit

Hiermit garantiere ich, Clemens Schäfermeier, dass die vorliegende Arbeit in seiner Ganzheit alleinig meiner Feder entsprungen ist. Werke anderer Autoren wurden nach bestem Wissen und Gewissen durch Typographie und / oder wörtlichen Ausdruck jedem Leser kenntlich gemacht.

Literatur

[1] Aguilera, A. and R. Pérez-Aguila: *General n-dimensional rotations.* WSCG SHORT Communication papers proceedings, 2004. `http://wscg.zcu.cz/wscg2004/Papers_2004_Short/N29.pdf` – Stand November 2011.

[2] Bronstein, Semendjajew, Musiol und Mühlig: *Taschenbuch der Mathematik.* Harri Deutsch Verlag, 2005.

[3] Feynman, R. P.: *QED – Die seltsame Theorie des Lichts und der Materie.* Piper Verlag GmbH, 1988.

[4] Freitag, E. und R. Busam: *Funktionentheorie 1.* Springer Verlag, 4te Auflage, 2006.

[5] GNU Compiler Collection, Version 4.6.2. `http://gcc.gnu.org/` – Stand November 2011, veröffentlicht Oktober 2011.

[6] gnuplot, Version 4.4.3. `http://www.gnuplot.info/` – Stand November 2011, veröffentlicht März 2011.

[7] Hansen, N.: *The cma evolution strategy: A tutorial.* `http://www.lri.fr/~hansen/cmatutorial.pdf` – Stand November 2011, veröffentlicht Juni 2011.

[8] Hecht, E.: *Optics.* Addison-Wesley Longman, 4th edition, 2003.

[9] Hedar, A. R. and M. Fukushima: *Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization.* `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/go_files/DSSA.pdf` – Stand November 2011, veröffentlicht August 2002.

[10] Hu, X., R. Eberhart, and Y. Shi: *Recent advances in particle swarm.* `http://www.swarmintelligence.org/papers/CEC2004Recent.pdf` – Stand November 2011, veröffentlicht 2004.

[11] Marquardt, D. W.: *An algorithm for the least-squares estimation of nonlinear parameters.* SIAM Journal of Applied Mathematics, 11:431–441, 1963.

[12] Matsumoto, M. and T. Nishimura: *Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.* ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation, 1998.

[13] Mouroulis, P. and J. Macdonald: *Geometrical Optics and Optical Design.* Oxford University Press, 1996.

[14] Nelder, J. A. and R. A. Mead: *A simplex method for function minimization.* Computer Journal, 7:308–313, 1965.

[15] Nussbaum, A.: *Modernizing the teaching of advanced geometric optics.* Education in Optics, 1603:389–400, 1991.

[16] Nussbaum, A.: *Modernizing the teaching of advanced geometric optics.* SPIE, 3190:34–44, 1991.

[17] Press, Teukolsky, Vetterling, and Flannery: *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 2nd edition, 1992.

[18] Siegman, A. E.: *Lasers.* University Science Books, 1986.

[19] Valgrind, Version 3.6.0. `http://valgrind.org/` – Stand November 2011, veröffentlicht Oktober 2010.

[20] Wikipedia: Text of Creative Commons Attribution: *Konvexe und konkave Funktionen*. `http://de.wikipedia.org/wiki/Konvexe_und_konkave_Funktionen`, Stand November 2011.

[21] Wikipedia: Text of Creative Commons Attribution: *Rosenbrock function*. `http://en.wikipedia.org/wiki/Rosenbrock_function`, Stand November 2011.

[22] Yariv, A.: *Quantum Electronics*. John Wiley & Sons, 3rd edition, 1987.