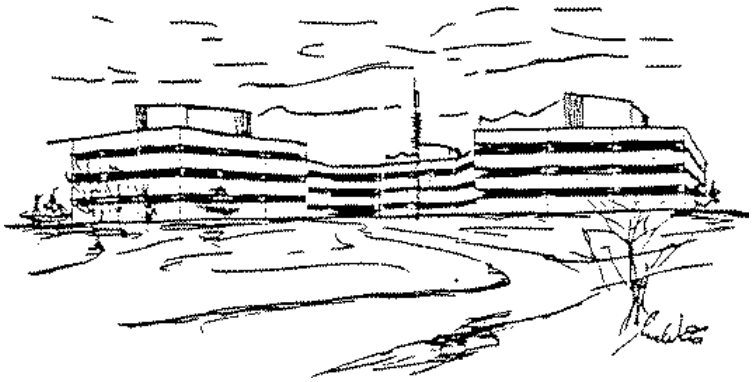
DIPLOMARBEIT

Dimensionierung und Optimierung eines optischen Wellenfilters



Autor: Michael Becker

Betreuer: Dipl.-Ing. Peter Crassen Hruschka

Datum: 27. April 2000

Universität der Bundeswehr München

Fakultät für Elektrotechnik

Institut für Nachrichtentechnik

Prof. Dr.-Ing. Udo Barabas

Inhaltsverzeichnis

Abbildungsverzeichnis

0. Einleitung

Die Anforderungen an die Informationsübertragung sind in den letzten Jahren stark angestiegen. Immer größer werdende Datenmengen müssen an immer mehr Anwender verteilt werden. Die herkömmlichen Übertragungsmedien sind bei solch hohen Datenraten schnell überfordert, so daß optische Medien zum Einsatz kommen.
Der fortschreitende Ausbau der Informationsnetze mit Lichtwellenleitern erfordert Bauelemente, die die anfallende Menge an Signalen verarbeiten können. Zur Zeit werden optische Filter entwickelt, welche u.a. die schmalbandigen Nachrichtenkanäle trennen können. Eine rasche Entwicklung derartiger Bauelemente ist dringend notwendig. Im Zusammenhang damit müssen Werkzeuge geschaffen werden, um die erforderlichen Berechnungsverfahren so schnell wie möglich durchführen zu können und die Herstellung der Filter zu ermöglichen.
Ziel dieser Diplomarbeit ist es, die Berechnung und die Erstellung des Layouts eines optischen Filters, welches in einem Halbleitermaterial realisiert werden soll, auf möglichst schnellem und einfachem Wege zu ermöglichen.

Am Institut für Nachrichtentechnik der Universität der Bundeswehr München wurden solche Berechnungen auf Workstations mit MatLab durchgeführt. Diese Berechnungen wurden immer komplexer, zudem müssen die vorhandenen Workstations auch andere Aufgaben erfüllen. Die Workstations sind derart ausgelastet, daß die Berechnungsdauer auf ein zu hohes Maß angestiegen ist. Da die Anschaffung solch hochwertiger Rechentechnik teuer ist, wurden bestimmte Programmteile auf PCs ausgelagert. Diese Programmteile wurden in C++ programmiert. Die hier nöti-

gen Berechnungen dauern auch in C relativ lange. Mit Hilfe der modernen Programmiersprache Java wurde die Möglichkeit geschaffen, mehrere Berechnungen auf verschiedenen Computern, welche über ein Netzwerk verbunden sind, gleichzeitig durchführen zu lassen.

Zur Herstellung der Filterstruktur in einem Halbleitermaterial sind mehrere Masken erforderlich. Deren Layout soll mit AutoCAD erstellt werden. Dabei gilt es, einige tausend verschiedenartige Polygone auf mehreren Layern zu zeichnen. Bei der Zeichnung dieser Masken sind zudem vielfältige Vorschriften und Restriktionen zu beachten, damit die Layouts von einem CAM-System verktoriell ausgelesen werden können. Das überaus aufwendige und komplizierte Zeichnen von Hand ist sehr schwierig, birgt eine nicht zu unterschätzende Fehlerquelle in sich und würde mehrere Wochen in Anspruch nehmen. Daher stellte sich die Frage nach einer Automatisierung dieser Aufgabe. Im Rahmen der vorliegenden Arbeit soll ein Werkzeug zur Verfügung gestellt werden, welches ein Zeichnen von Hand unnötig macht und in AutoCAD eingebunden werden kann.

In dieser Arbeit werden die bereitgestellten Werkzeuge zur Herstellung eines optischen Wellenfilters chronologisch beschrieben und erläutert — von einer kurzen Theoriebetrachtung in Kapitel 1 über Berechnungen (Kapitel 2 bis 6) sowie Ergebnisse (Kapitel 7) bis hin zur Erstellung der benötigten Zeichnungen in den Kapiteln 8 und 9.

1. Theoretische Vorbetrachtung

1.1. Der Filteraufbau

Das Filter wird in einem geeigneten Halbleitermatrial realisiert. Es besteht aus zwei Wellenleitern mit dem Sende-[1] bzw. dem Empfangsrücken[2]. Die Aufgabe des Senderückens besteht grundsätzlich darin, an seinem horizontalen (oberen) Ausgang eine vom Ort unabhängige sowie betrags- und phasenmäßig konstante Feldstärke bereitzustellen. Im Empfangsrücken wird die gewünschte Filterung implementiert.

An der linken Seite des Streurückens wird das Licht eingekoppelt, auf der gleichen Seite des Sammelrückens kann das gefilterte Licht ausgekoppelt werden. Sende- und Empfangsrücken enthalten um 45° bzw. 135° geneigte Gräben, so daß ein „Fischgräten"-Muster entsteht, vgl. auch Abb. 1.1.

1.2. Die Modellstruktur

Entsprechend [1] wird über beide Rücken eine Gitterstruktur gelegt. Diese ist in Abbildung 1.1 dargestellt. Aufgrund der 45°-Neigung der Gräben teilen sich die Zellen in Graben- und Leerzellen auf. Die Länge und Breite einer Gitterzelle entsprechen

[1] auch Streurücken genannt.
[2] im Verlauf der Arbeit auch als Sammelrücken bezeichnet.

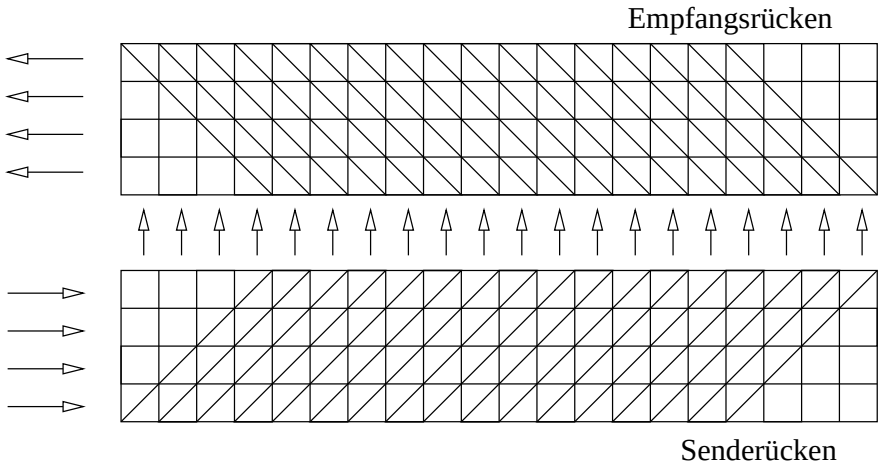


Abbildung 1.1.: Gitterstruktur

der Wellenlänge des Lichts im Halbleitermaterial Λ_m bei der Mittenfrequenz. Leerzellen werden als Grabenzellen mit einem Reflexionsfaktor von $r = 0$ betrachtet.

1.3. Allgemeine Berechnung der Struktur

1.3.1. Der Streurücken

Eine Zelle des Senderückens hat den in Abbildung 1.2 dargestellten Aufbau und die durch Pfeile angezeigten Ein- und Ausgänge. Bei der allgemeinen Betrachtung

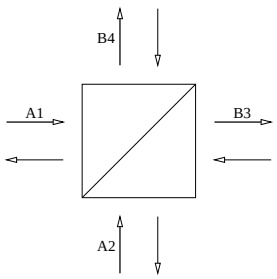


Abbildung 1.2.: Streurückenzelle mit Ein- und Ausgängen

einer solchen Zelle sind vier Ein- und Ausgänge möglich [2]. Aufgrund der hier verwendeten Struktur sind nur die Ein- und Ausgänge zu betrachten, welche in Abb. 1.2 mit fetten Pfeilen eingezeichnet und beschriftet sind. Die restlichen Ein- bzw. Ausgänge (dünn gezeichnete und nicht beschriftete Pfeile) ergeben sich zu Null.

Die Beschreibung einer solchen Zelle erfolgt durch ihre Streumatrix $\underline{S}$. Die Eingangsgrößen werden im Vektor A und die Ausgangsgrößen im Vektor B zusammengefaßt. Der allgemeine Zusammenhang dieser Größen läßt sich durch

$$B = \underline{S} \cdot A \tag{1.1}$$

darstellen. Die Berechnung der relevanten Ausgänge einer Streugrabenzelle erfolgt nach der Formel

$$\begin{pmatrix} B_3 \\ B_4 \end{pmatrix} = \begin{pmatrix} t & jr \\ jr & t \end{pmatrix} \cdot \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \tag{1.2}$$

Dabei sind A_1 und A_2 die komplexen Werte der Eingangs- sowie B_3 und B_4 die komplexen Werte der Ausgangswellengrößen einer Zelle.
r ist der Reflexionsfaktor, der dieser Grabenzelle zugeordnet ist. Bei der Reflexion an dem um 45° geneigten Graben erfährt der Wellenanteil eine Phasendrehung von 90° [3]. Das wird durch die Multiplikation des Refelxionsfaktors mit der komplexen Zahl $j = e^{j\frac{\pi}{2}}$ berücksichtigt.
Der Transmissionsfaktor t des Grabens in dieser Zelle ergibt sich aus dem Zusammenhang

$$t = \sqrt{1 - r^2} \tag{1.3}$$

Reflexionsfaktor r und Transmissionsfaktor t sind rein reele Größen mit der Bedingung $0 \leq r \leq 1$ bzw. $0 \leq t \leq 1$.

Auf der Basis dieser Formel kann der gesamte Senderücken von links nach rechts und dabei jeweils von unten nach oben berechnet werden, d.h. von der ersten (links) bis zur letzten Spalte (rechts), wobei eine Spalte jeweils von der untersten bis zur obersten Zeile berechnet wird.

1.3.2. Der Sammelrücken

Das Modell einer Zelle des Empfangsrückens ist in Abbildung 1.3 inklusive der relevanten Ein- und Ausgangswellengrößen dargestellt.

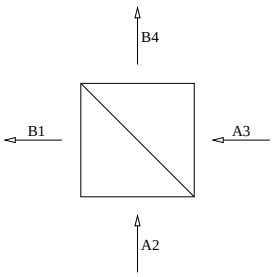


Abbildung 1.3.: Sammelrückenzelle mit Ein- und Ausgängen

Die Berechnung einer Sammelgrabenzelle erfolgt nach der Formel

$$\begin{pmatrix} B_1 \\ B_4 \end{pmatrix} = \begin{pmatrix} jr & t \\ t & jr \end{pmatrix} \cdot \begin{pmatrix} A_2 \\ A_3 \end{pmatrix} \tag{1.4}$$

Dabei sind A_2 und A_3 die komplexen Werte der Eingangs- sowie B_1 und B_4 die komplexen Werte der Ausgangswellengrößen dieser einen Zelle.
r ist der Reflexionsfaktor und t der Transmissionsfaktor des Grabens in dieser Zelle.

Aufgrund des anderen Grabenwinkels muß der Sammelrücken von unten nach oben und dabei jeweils von rechts nach links berechnet werden.

2. Bestimmung der Filterkoeffizienten

2.1. Die Theorie

Die vorliegende Struktur realisiert stets ein Bandpaßfilter. Der Entwurf des Filters erfolgt entsprechend der in [1] beschriebenen Filtertheorie. Hierbei wird ein Transversalfilter, wie aus der Theorie digitaler Filter bekannt, entworfen. Die ermittelten Filterkoeffizienten werden entsprechend der vorliegenden flächigen Struktur in die effektiven Reflexionsfaktoren der Gräben umgerechnet (siehe Kapitel 3).

Ein idealer Bandpaß stellt im Spektralbereich ein Rechteck dar. Im Zeit- bzw. Ortsbereich ergibt sich nach Anwendung der inversen diskreten Fouriertransformation (DFT) [4] eine diskrete Ortsfunktion der Filterkoeffizienten[1]. Der Abstand zweier Koeffizienten im Ortsbereich beträgt Λ_m [1].

2.2. Die si-Funktion und das Kaiserfenster

Die mit Hilfe der inversen DFT erhaltene Ortsfunktion ist eine si-Funktion. Diese ist sowohl nach links als auch nach rechts undendlich ausgedehnt, und wie folgt

[1] z.T. auch nur als Koeffizienten bezeichnet

definiert:

$$si(x) = \begin{cases} \frac{sin\ x}{x} & \text{für } x \neq 0 \\ 1 & \text{für } x = 0 \end{cases} \tag{2.1}$$

Zur Veranschaulichung ist in Abb. 2.1 eine solche *si*-Funktion dargestellt.

Für die Herstellung des optischen Wellenfilters muß diese Funktion an einer bestimmten Stelle abgebrochen werden, da das Filter eine endliche Länge hat. Der Hauptschwerpunkt der Berechnungen wurde auf die nach jeweils 2π abgebrochene *si*-Funktion ($\pm 2\pi = 2 \cdot 2\pi = 4\pi$) für die Filterkoeffizienten gelegt. In Abb. 2.2 ist eine solche Ortsfunktion der Filterkoeffizienten dargestellt.
Natürlich kann die unendlich ausgedehnte *si*-Funktion auch an jeder anderen Stelle abgebrochen werden, z.B. nach je π (siehe Abb. 2.3).

Durch das Abbrechen der *si*-Funktion wird ein Fehler gemacht. Bei der Transformation einer abgeschnittenen *si*-Funktion in den Spektralbereich entsteht kein idealer Bandpaß mehr. Um diesen Fehler so gut wie möglich zu beheben, wird die abgebrochene *si*-Funktion mit einer Fensterfunktion multipliziert.
Hierzu soll ein Kaiserfenster verwendet werden. Die Kaiserfunktion ist in [5] ausführlich beschrieben, sie lautet:

$$w[n] = \begin{cases} \frac{I_0\left(\beta \cdot \sqrt{1-(\frac{n-\alpha}{\alpha})^2}\right)}{I_0(\beta)} & \text{für } 0 \leq n \leq N \\ 0 & \text{sonst} \end{cases} \tag{2.2}$$

Dabei steht $I_0(\dots)$ für die modifizierte Bessel-Funktion erster Art nullter Ordnung. $(N+1)$ ist die Länge des Fensters, $\alpha = \frac{N}{2}$ und β ist der Formparameter.

Mit dem Kaiserkoeffizienten (Formparameter) β kann die Gestalt des Fensters verändert werden, um damit zwischen der Amplitude der Nebenmaxima und der Breite des Hauptmaximums einen Kompromiß zu finden. In Abb. 2.4 sind die Fensterfunktionen für die hier verwendeten Kaiserkoeffizienten β dargestellt. Um diese und auch die in Kapitel 7 dargestellten Grafiken besser deuten zu können, wurden die verwendeten Koeffizienten β_{Kaiser} für die Kaiserfunktion fest einer bestimmten Linienfarbe zugeordnet.

Kaiserkoeffizient β	Linienfarbe
2,0	blau
3,4	rot
5,0	schwarz
ungewichtet	grün

In Abb. 2.5 sind die ungewichtete *si*-Funktion und die durch Kaiserfenster mit verschiedenen Koeffizienten ($\beta_{Kaiser} = 2,0; 3,4; 5,0$) gewichteten Ortsfunktionen der Filterkoeffizienten dargestellt.

2.3. Das Programm zur Ermittlung der Koeffizienten

Mit dem MatLab-Programm **fxsfind.m** wird die Ortsfunktion der Filterkoeffizienten für eine bestimmte Filterbandbreite bestimmt (siehe 2.1). Der Quellcode ist unter C.9 zu finden.
Zur problemlosen Ausführung von **fxsfind** sind folgende MatLab-Skripten erforderlich.

- fxs.m (Erstellung des Spektrums, siehe C.8)
- rect.m (Erstellung eines Bandpasses, siehe C.18)
- invfour.m (Inverse Fouriertransformation, siehe C.14)

2.3.1. Aufruf und Eingaben

Die Verzeichnisse, in welchen sich die oben genannten Skripten befinden, müssen im MatLab-Pfad eingetragen sein, damit MatLab diese Dateien während der Ausführung findet. Um das Programm zu starten gibt man in der MatLab-Kommandozeile

`fxsfind`

ein. Es erscheint die Überschrift des Berechnungsprogrammes. Der Benutzer wird am Anfang zu einigen Eingaben aufgefordert, welche im folgenden erläutert werden.

- *Wellenlänge des Lasers*

  Dies ist gleichzeitig die Mittenwellenlänge des zu bestimmenden Filters. Der Standardwert ist in eckigen Klammern angegeben, dieser wird durch Drücken der Eingabetaste ohne Eingeben eines Wertes übernommen. Die Einheit ist Nanometer.

- *Wellenlängenabstand*

  Dieser bezeichnet den Abstand zum benachbarten Laser mit der nächsten auftretenden Frequenz und somit auch die Bandbreite des Filters. Die Einheit ist ebenfalls Nanometer.

- *Anzahl der genutzen Perioden*

  Hier wird bestimmt, wieviele Perioden der si-Funktion genutzt werden sollen. Die Angabe erfolgt in π-Schritten. `1` bedeutet, daß sowohl im positiven als auch im negativen Bereich die Ortsfunktion nach der ersten Nullstelle abgeschnitten wird.

- *Zwischenergebnisse anzeigen*

  Wird hier `1` eingegeben (also *ja* gewählt), so werden die Daten jedes Berechnungsschrittes ausgegeben, andernfalls nur das Endergebnis.

2.3.2. Speicherung der Ergebnisse

Die Ergebnisse werden relativ zum Heimatverzeichnis des Benutzers gespeichert. Im Normalfall geschieht dies im Verzeichnis **m/daten/fxsfind** unter einem eindeutigen Dateinamen. Das Stammverzeichnis **/m** kann jederzeit im Quellcode geändert werden. Dieses ist in jedem speichernden MatLab-Programm unter der Variablen *verz* abgelegt, welche jeweils zu Beginn gesetzt wird. Das Aussehen des Dateinamens muß

an dieser Stelle nicht explizit erklärt werden, da das nachfolgende Programm diesen automatisch anhand der vom Anwender gemachten Angaben ermittelt.
Es besteht die Möglichkeit, einen eigenen Speicherdateinamen anzugeben. Dafür gibt es bei den Eingaben den Punkt *Speicherdateinamen selbst angeben*. Hier wird der gewünschte Dateiname relativ zu **m/myfuncs/daten** angegeben. Die im nächsten Punkt beschriebene Grafik wird dann im Verzeichnis **m/myfuncs/plots** unter dem gleichen Namen als eps-Datei abgespeichert.
Es wird empfohlen, die automatische Vergabe der Dateinamen zu nutzen, um eventuellen Verwechslungen vorzubeugen.

2.3.3. Visualisierung

Nach erfolgreicher Beendigung der Berechnungen wird die Funktion der Filterkoeffizienten über der Grabennummer des entstehenden Rückens aufgetragen.
Die dargestellte Figur wird bei automatischer Vergabe der Dateinamen im Verzeichnis **m/daten/fxsfind** relativ zum Heimatverzeichnis des Benutzers als eps-Datei abgespeichert. Dabei ist der Dateiname der gleiche wie der der zugehörigen Datendatei.

2.4. Das Programm für die Fensterfunktion

Die bestimmten Filterkoeffizienten werden vom MatLab-Programm **fxswin.m** mit einer Fensterfunktion multipliziert, um das Abschneiden der Funktion auszugleichen (siehe auch Kapitel 2.2). Das ist eine gängige Methode bei der digitalen Filterauslegung. Der Quellcode dieses Programmes ist unter C.11 zu finden.
Zur problemlosen Ausführung sind zusätzlich folgende Skripten erforderlich:

- fxsfour.m (Durchführung der Transformation der Koeffizienten, siehe C.10)
- four.m (Allgemeine DFT, siehe C.7)

- kaiser.m (Bestandteil der MatLab Signal Processing Toolbox)
- saven.m (Speichern einer ASCII-Datei, siehe C.19)
- time.m (Ausgabe des Datums und der aktuellen Uhrzeit, siehe C.20)

2.4.1. Aufruf und Eingaben

Das Programm wird durch Eingabe von

```
fxswin
```

in der MatLab-Kommandozeile gestartet. Nach dem Start des Programmes und der Ausgabe der Überschrift werden einige Eingaben erwartet. Die Bedeutungen dieser Eingaben sind teilweise unter 2.3.1 beschrieben. Mit der zusätzlichen Frage nach dem *Kaiserkoeffizienten beta* ist der Koeffizient β_{Kaiser} der verwendenden Fensterfunktion nach Kaiser gemeint. Er entscheidet über die Wichtung der vorher berechneten si-Funktion für die Filterkoeffizienten.

2.4.2. Laden der Quelldaten und Speicherung der Ergebnisse

Auch hier besteht die Möglichkeit, den Daten- und Speicherdateinamen selbst anzugeben. Dies ermöglichen die Eingabepunkte *Datendateinamen selbst angeben* und *Speicherdateinamen selbst angeben*. Die Verfahrensweise und der zugrundeliegende Quell- bzw. Speicherpfad sind identisch mit der Bestimmung der Filterkoeffizienten durch **fxsfind** (siehe 2.3.2).
Es wird nochmals darauf hingewiesen, die automatische Generierung der Dateinamen zu verwenden, um eventuelle Fehler zu vermeiden.

Nach der Berechnung und der Visualisierung des Ergebnisses gibt es die Möglichkeit, die berechneten Filterkoeffizienten als ASCII-Datei zu speichern. Das ist für die spätere Umrechnung der soeben bestimmten Filterkoeffizienten in die entsprechenden Reflexionsfaktoren mit Hilfe der C++ Programme **ainr** (siehe 3.3) und

ainr2 (siehe 3.4) notwendig. Der Speicherdateiname wird automatisch anhand der Filterdaten bestimmt und entspricht der verwendeten Datenstruktur (siehe A).

2.4.3. Visualisierung

Nach Abschluß der Berechnungen wird in einer MatLab-Figur folgendes angezeigt:

- die kaisergewichteten Filterkoeffizienten,
- das relative Spektrum des Filters,
- die absolute Ausgangsleistung über der Freiraumwellenlänge,
- die relative Ausgangsleistung in dB.

Die dargestellte Figur wird im Verzeichnis **m/daten/fxswin** relativ zum Heimatverzeichnis des Benutzers als eps-Datei abgespeichert. Dabei ist der Dateiname der gleiche wie der der zugehörigen Datendatei.

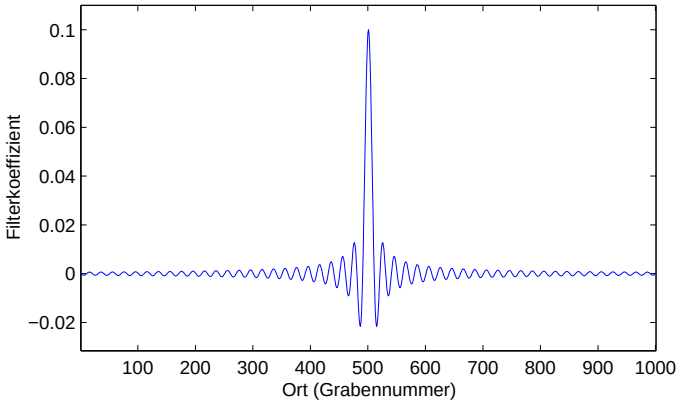


Abbildung 2.1.: Beispiel für eine *si*-Funktion

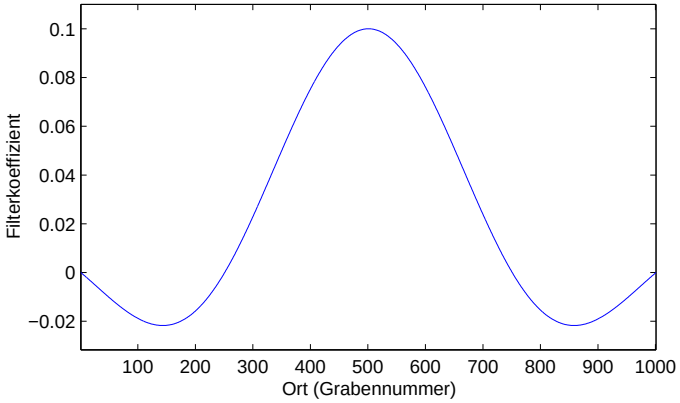


Abbildung 2.2.: Nach $\pm 2\pi$ abgebrochene *si*-Funktion

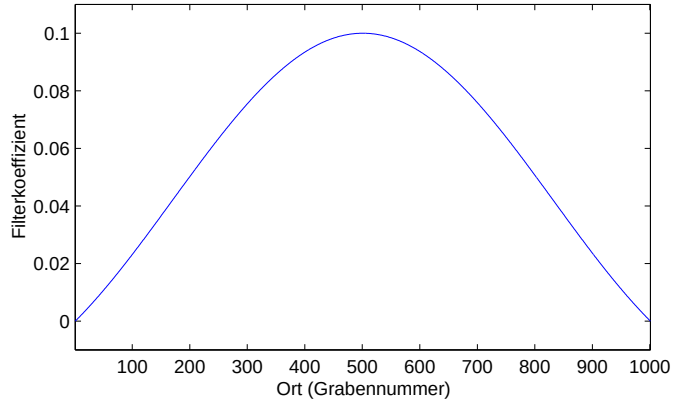
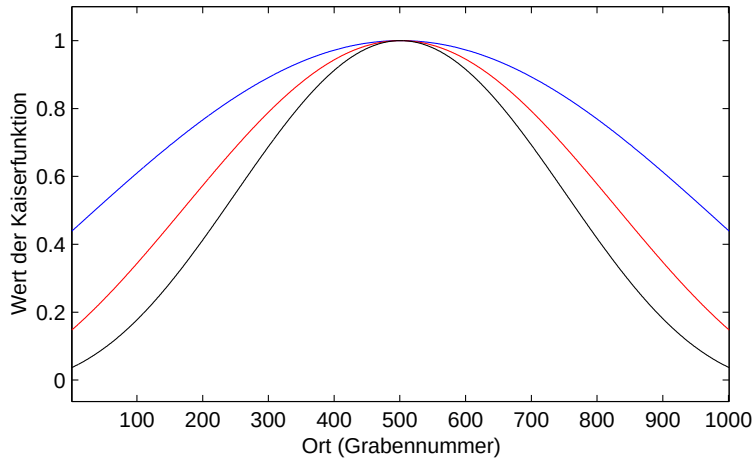
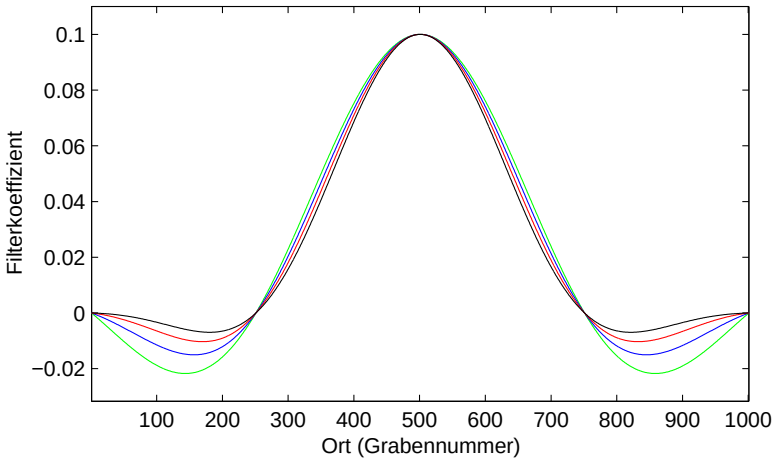


Abbildung 2.3.: Nach $\pm \pi$ abgebrochene *si*-Funktion



blau: $\beta_{Kaiser} = 2,0$ rot: $\beta_{Kaiser} = 3,4$ schwarz: $\beta_{Kaiser} = 5,0$

Abbildung 2.4.: Verschiedene Kaiserfenster



grün: ungewichtet blau: $\beta_{Kaiser} = 2,0$

rot: $\beta_{Kaiser} = 3,4$ schwarz: $\beta_{Kaiser} = 5,0$

Abbildung 2.5.: Mögliche Filterkoeffizienten (gewichtet und ungewichtet)

3. Berechnung der Reflexionsfaktoren

3.1. Übertragung der Filterkoeffizienten

Abbildung 3.1 zeigt eine Beispielfunktion für die Filterkoeffizienten (si-Funktion) bei einer kleinen Rückenlänge. Diese si-Funktion wurde nach $\pm 2\pi$ abgebrochen, sie hat ihren Maximalwert in der Mitte des Rückens. Im dargestellten Beispiel wurde eine Rückenlänge von 41 Gräben betrachtet. Der Maximalwert der Filterkoeffizienten beträgt $si_{max} = 0,1$ und liegt bei Graben Nummer 21.

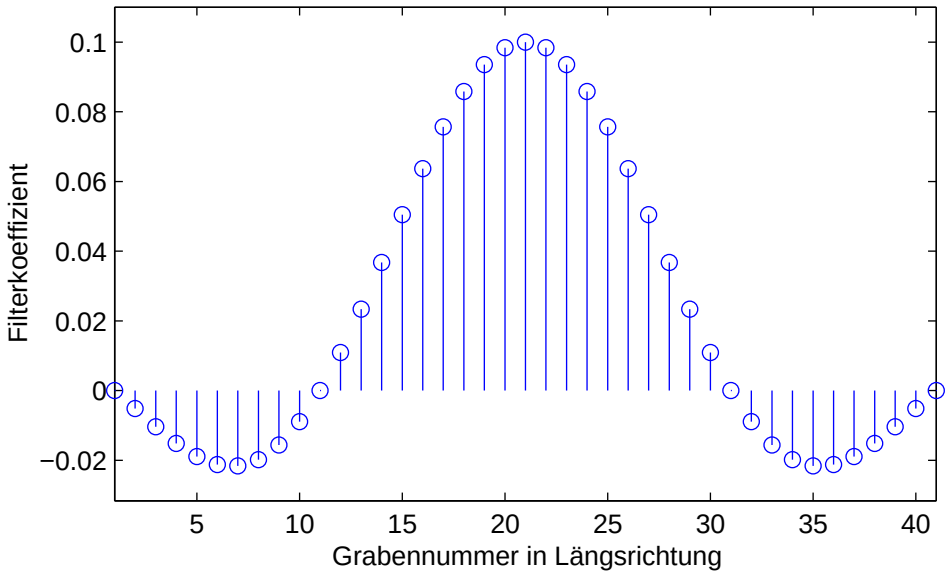
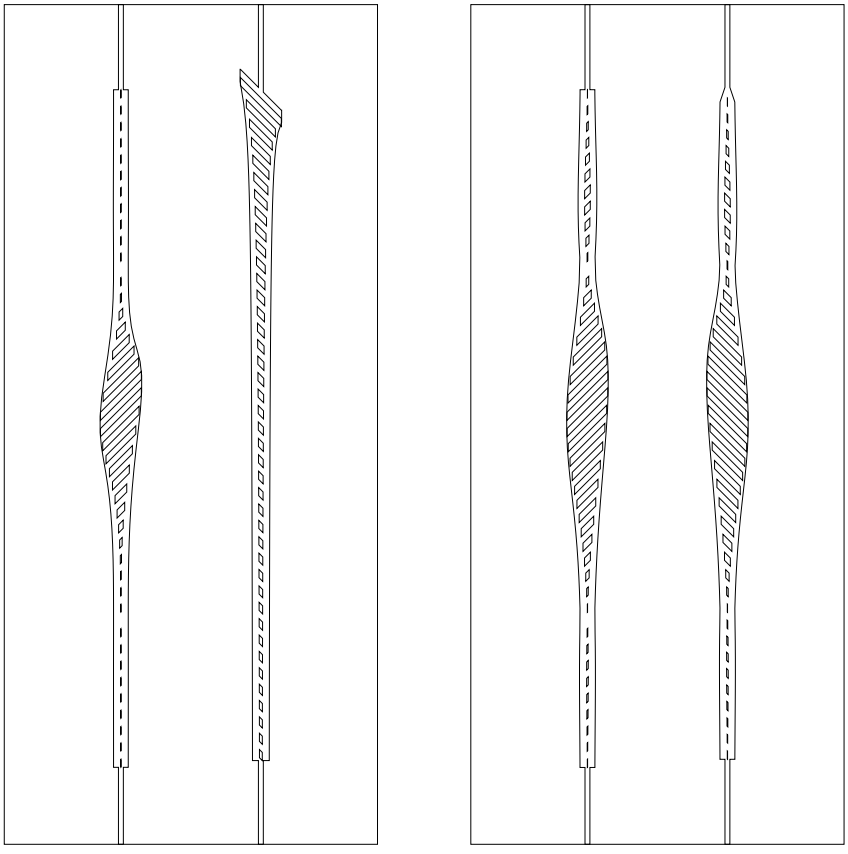


Abbildung 3.1.: Beispiel für die Filterkoeffizienten

Die bestimmten Koeffizienten müssen auf die Modellstruktur des Filters übertragen werden. Dazu sollen hier zwei Varianten betrachtet werden. Der schematische Aufbau des optischen Wellenfilters für beide Methoden ist in Abb. 3.2 dargestellt.

Methode 1: Der Empfangsrücken soll die Filterfunktion realisieren. Der Senderücken soll ausschließlich dazu dienen, daß in den Empfangsrücken eine Wellenfront mit örtlich gleicher Amplitude und konstanter Phase eingekoppelt wird. Näheres ist unter 3.3 zu finden.

Methode 2: Die Filterkoeffizienten werden auf Sende- und Empfangsrücken gleichzeitig übertragen (siehe 3.4).



links: Aufbau nach Methode 1 rechts: Aufbau nach Methode 2

Abbildung 3.2.: Schematischer Aufbau des Wellenfilters

Der Maximalwert der *si*-Funktion (si_{max}) für die Filterkoeffizienten ist wichtig für

den durch die Umrechnung entstehenden maximalen effektiven Reflexionsfaktor. Zu berücksichtigen ist, daß ein und derselbe *si*-Maximalwert bei unterschiedlichen Rückenlängen auch unterschiedliche maximale Reflexionsfaktoren liefert.

Der Maximalwert der zu berechnenden effektiven Reflexionsfaktoren[1] ist also abhängig von:

- der Rückenlänge,
- dem Maximalwert der *si*-Funktion für die Filterkoeffizienten,
- dem Kaiserkoeffizienten β der Fensterfunktion.

Der Zusammenhang zwischen dem Maximum der Filterkoeffizienten und dem Maximalwert der Reflexionsfaktoren ist nicht linear. Aus diesem Grund kann bei den im folgenden beschriebenen Berechnungen nur nach der „try-and-error“ Methode verfahren werden. Die Zusammenhänge der durch die Berechnungen gewonnenen Ergebnisse werden in Kapitel 7 diskutiert.

Das Finden des gewünschten Maximalwertes der so berechneten effektiven Reflexionsfaktoren für die einzelnen Gräben ist vor allem bei längeren Rücken sehr zeitaufwendig. Der Grund ist, daß alle auftretenden Mehrfachreflexionen berücksichtigt werden müssen.
Um den Zeitaufwand wesentlich zu reduzieren, wurde die Möglichkeit geschaffen, diese und weitere Berechnungen mit Hilfe der Programmiersprache Java auf mehrere Personalcomputer zu verteilen. Die Funktionsweise des verteilten Rechnens wird in Kapitel 5 beschrieben, deren Nutzung für diese und weitere Berechnungen ist in Kapitel 6 erläutert.
Die Verteilung ist nicht zwingend erforderlich, aber die Zeit bis zur Gewinnung der gewünschten Reflexionsfaktoren wird dadurch wesentlich verkürzt.

[1] Die effektiven Reflexionsfaktoren werden teilweise nur als Reflexionsfaktoren bezeichnet.

3.2. Die Funktion zum Vorbereiten der Filterkoeffizienten

3.2.1. Funktionsweise

Die nach Kapitel 2 bestimmten und mit einer Fensterfunktion multiplizierten Filterkoeffizienten bilden die Grundlage für deren Umrechnung in die effektiven Reflexionsfaktoren für die einzelnen Gräben. Aufgrund der unter 3.1 erwähnten Eigenheiten dieser Umrechnung müssen die Filterkoeffizienten mit verschiedenen Maximalwerten bereitgestellt werden. Dazu wird die bestimmte Ortsfunktion für die Filterkoeffizienten geladen. Jeder einzelne Koeffizient wird mit dem gleichen konstanten Faktor multipliziert. Anschließend wird die so erhaltene Ortsfunktion mit einem anderen Maximalwert in eine neue Datei gespeichert.
Das Vorbereiten des Maximums der Filterkoeffizienten übernimmt die MatLab-Funktion **ainr_v.m**. Der Quellcode für diese Funktion befindet sich unter C.3.
Zur problemlosen Ausführung von **ainr_v** sind folgende Skripten erforderlich:

- laden.m (Laden einer ASCII-Datei, siehe C.15)
- saven.m (Speichern einer ASCII-Datei, siehe C.19)

3.2.2. Anwendung der MatLab-Funktion

In der MatLab-Kommandozeile ruft man die Funktion wie nachstehend beschrieben auf.

```
ainr_v(rueckenbreite, anz_pi, kaiser_beta, si_original, si_max);
```

Die Bedeutung der einzelnen Parameter wird nun aufgezeigt.

- *rueckenbreite*

  Die Rückenlänge in Gräben, zum Beispiel `'1000'` für 1000 Gräben. Die Länge

ergibt sich aus der Berechnung der Filterkoeffizienten, welche unter 2.3 beschrieben wurde.

- *anz_pi*
  Die Anzahl der genutzten si-Perioden, wie bei der Bestimmung der Koeffizienten unter 2.3.1, z.B. `'2'`.

- *kaiser_beta*
  Der Koeffizient der benutzten Fensterfunktion unter 2.4.1, z.B. `'20'` für beta = 2,0.

- *si_original*
  Das Maximum der originalen si-Funktion, welche mit Hilfe des MatLab-Programmes **fxswin** (wie unter 2.4.2 beschrieben) gespeichert wurde, z.B. `'0015'` für ein Maximum von 0,0015.

- *si_max*
  Das neue Maximum der gewünschten si-Funktion, z.B. `'02'` für ein Maximum von 0,02.

Es wird angezeigt, aus welcher Datei die Koeffizienten geladen und in welche die daraus berechneten gespeichert wurden.
Eine Darstellung der neu berechneten Filterkoeffizienten findet nicht statt, da sich im Endeffekt nichts am äußeren Erscheinungsbild der Funktion geändert hat.

3.3. Die Berechnung nach Methode 1 – Nur Sammelrücken

3.3.1. Berechnungsvorschrift

Es wird davon ausgegangen, daß vom Senderücken eine Wellenfront mit gleicher Amplitude und konstanter Phase über die gesamte Rückenlänge eingekoppelt wird.

Die Filterkoeffizienten sollen nur auf den Sammelrücken übertragen werden (siehe Abb. 3.2 links).

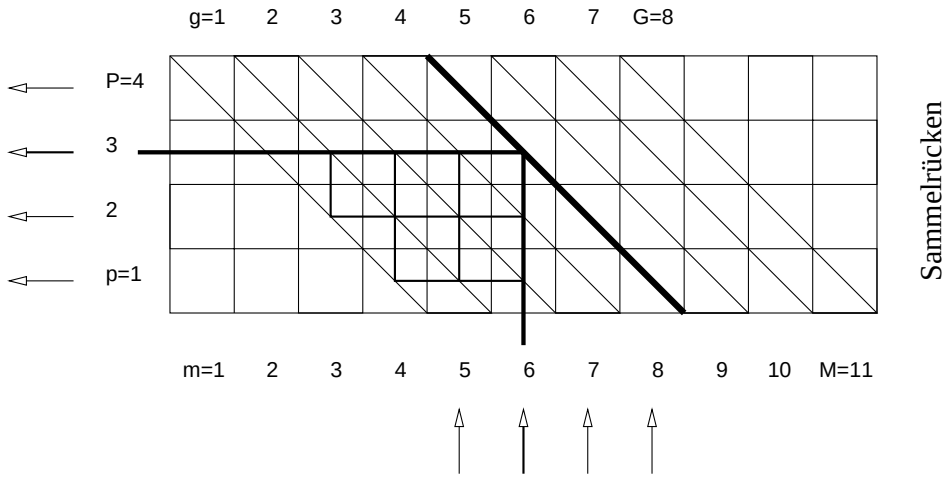


Abbildung 3.3.: Bestimmung der Reflexionsfaktoren (Sammelrücken)

Abbildung 3.3 zeigt die Gitterstruktur des Sammelrückens und die möglichen Wellenflüsse für die Berechnung des Ausgangs einer (von allen) Zeile(n) zur Bestimmung des Reflexionsfaktors für den fett dargestellten Graben.
In dieser Abbildung sind über der Struktur die Grabennummern (von links nach rechts) angetragen, unter der Struktur findet man die Nummerierung der Spalten. Graben- und Spaltennummer unterscheiden sich aufgrund der 45°-Neigung der Gräben. Links von der Gitterstruktur befindet sich die Nummerierung der einzelnen Zeilen.

Zur allgemeinen Berechnung der Struktur müssen alle Gitterzellen (p, m) durchgerechnet werden. Dabei stellt p die laufende Zeilennummer und m die laufende Spaltennummer dar.
Die Rückenlänge ist gleich der Anzahl der vorhandenen Gräben, diese soll fortan mit G bezeichnet werden. Die Anzahl der vorhanden Zeilen wird mit P bezeichnet und stellt gleichzeitig die Rückenbreite dar. Auf Grundlage dieser Bezeichnungen und der 45°-Neigung der Gräben ergibt sich für die verwendete Gitterstruktur eine Spaltenanzahl von $M = G + P - 1$. Die Definitionsbereiche für p und m lauten

demzufolge $1 \leq p \leq P$ bzw. $1 \leq m \leq M$.
In der dargestellten Figur (Abb. 3.3) beträgt die Rückenlänge $G = 8$ und Rückenbreite $P = 4$. Somit ergeben sich eine Zeilenanzahl von $P = 4$ und eine Spaltenanzahl von $M = 8 + 4 - 1 = 11$.

Die Bestimmung der Reflexionsfaktoren für die einzelnen Gräben erfolgt von links nach rechts, also von Graben 1 bis Graben G, im hier betrachteten Beispiel also von eins bis acht. In der gezeigten Abbildung sind die Reflexionsfaktoren der Gräben eins bis vier bereits bekannt. Nun soll der effektive Reflexionsfaktor für den fünften Graben ($g = 5$, fett gezeichnet) aus dem fünften Koeffizienten der Ortsfunktion für die Filterkoeffizienten (siehe Kapitel 2) berechnet werden.
Dazu wird jeder vertikale Ausgang (eine Zeile) des Sammelrückens einzeln betrachtet [1], damit immer gleich lange Wege durch die Struktur zusammengefaßt werden. Um dies zu erreichen wird nur ein horizontaler unterer Eingang des Empfangsrückens betragsmäßig mit Eins belegt. Die restlichen Eingänge werden auf Null gesetzt. Welcher Eingang als einziger aktiv ist, ergibt sich aus der Hauptreflexion an dem Graben, für welchen der Reflexionsfaktor berechnet werden soll.

Im dargestellten Fall soll als Beispiel der Ausgang für die 3. Zeile ($p = 3$) bestimmt werden. Die Spaltennummer für den aktiven Eingang ergibt sich zu $m = g + P - p$. Dabei ist g die Nummer des aktuellen Grabens, P die Anzahl der vorhandenen Zeilen und p die Nummer der Zeile, für welche der Ausgang berechnet werden soll. Hier ergibt sich also die Spaltennummer des aktiven horizontalen Eingang zu $m = 5 + 4 - 3 = 6$. Dieser Eingang wurde mit einem etwas dicker gezeichneten Pfeil kenntlich gemacht (unten), genauso verhält es sich auch mit dem zugehörigen vertikalen Ausgang (links).
Die anderen Pfeile unterhalb und links der Struktur kennzeichnen die ebenfalls einzeln auf betragsmäßig Eins zu setzenden Eingänge, um die restlichen vertikalen Ausgänge für diesen Graben zu berechnen. Dabei korrespondiert aufgrund der Grabenneigung ein weiter rechts liegender Eingang (höhere Spaltennummer) mit einem weiter unten liegenden Ausgang (niedrigere Zeilennummer).

Da die links vom betrachteten Graben liegenden Reflexionsfaktoren bereits bekannt

sind, kann dieser Teil der Gitterstruktur jeweils komplett berechnet werden. Dabei werden alle auftretenden Mehrfachreflexionen berücksichtigt.
Zu den auf diese Weise berechneten Ausgangsfeldstärken der einzelnen Zeilen wird im Anschluß jeweils die Hauptreflexion am Graben mit dem noch unbekannten Reflexionsfaktor addiert. Dabei sind sowohl die unteren Transmissionen in der jeweiligen Spalte (vor dem Auftreffen auf den Graben) als auch die links liegenden Transmissionen in der jeweiligen Zeile (nach der Reflexion am Graben) zu berücksichtigen.

Die Gleichung zur Berechnung der Ausgangsfeldstärke E für eine Zeile lautet aufgrund der oben erläuterten Betrachtungsweise:

$$E_p = aus_p + fakt_p \cdot r_g \cdot \prod_{i=1}^{g-1} t_i \tag{3.1}$$

Dabei ist r_g der zu bestimmende Reflexionsfaktor des g-ten Grabens. Die aus den bereits bekannten Reflexionsfaktoren berechnete Ausgangfeldstärke der p-ten Zeile ist mit aus_p bezeichnet. $fakt_p$ ist der Faktor, mit welchem der zu bestimmende Reflexionsfaktor multipliziert werden muß, damit die Transmissionen durch die Zellen der weiter unten liegenden Zeilen berücksichtigt werden. $\prod_{i=1}^{g-1} t_i$ ist das Produkt der Transmissionsfaktoren aller links vom aktuellen Graben liegenden Gräben in der entsprechenden Zeile.

Nun kann die Gleichung zur Bestimmung des gesuchten effektiven Reflexionsfaktors aufgestellt werden. Dazu werden die Ausgangsfeldstärken aller Zeilen aufsummiert. Die so erhaltene Gesamtfeldstärke muß gleich dem Filterkoeffizienten a_g für diesen Graben sein [1]. Die komplette Gleichung lautet dann:

$$\sum_{p=1}^{P} E_p = a_g \tag{3.2}$$

Dabei sind P die Rückenbreite der betrachteten Struktur und E_p die nach Gleichung 3.1 berechneten Ausgangsfeldstärken für die einzelnen Zeilen.

Durch Einsetzen der Gleichung 3.1 in Gleichung 3.2 läßt sich diese leicht zu

$$r_g = \frac{a_g - \sum_{p=1}^{P} aus_p}{\left(\sum_{p=1}^{P} fakt_p\right) \cdot \prod_{i=1}^{g-1} t_i} \tag{3.3}$$

umformen, denn sie ist rein reel und linear. Es gibt somit genau eine Lösung.

Die Formel 3.3 wird für jeden Graben des Sammelrückens mit den entsprechend bestimmten Werten für die Ausgänge aus_p und Faktoren $fakt_p$ angewendet und liefert somit den jeweiligen effektiven Reflexionsfaktor unter Berücksichtigung von allen auftretenden Effekten höherer Ordnungen.

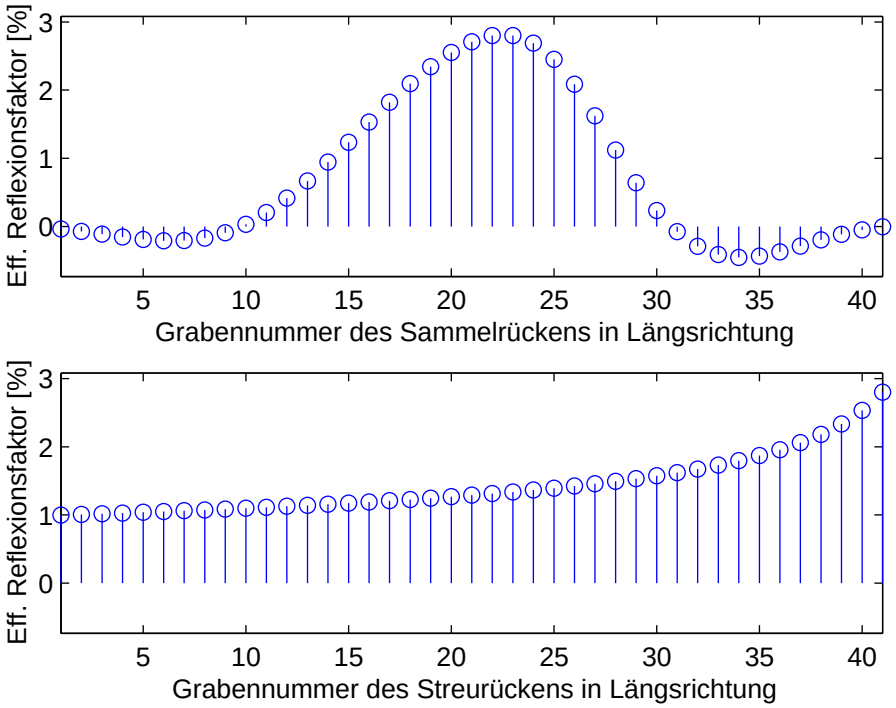


Abbildung 3.4.: Beispiel für eff. Reflexionsfaktoren nach Methode 1

Ist dies geschehen, ergibt sich eine diskrete Funktion für die effektiven Reflexionsfaktoren. Ein Beispiel ist in Abb. 3.4 oben dargestellt. Es ist zu erkennen, daß sich das Maximum der Funktion für den Sammelrücken nach rechts verschoben hat (vergleiche Abb. 3.1). Dieser Maximalwert der Reflexionsfaktoren befindet sich im dargestellten Fall beim 23. Graben, er beträgt hier $r_{max} = 2,8\%$.
Die dazugehörigen effektiven Reflexionsfaktoren für den Streurücken, welche nach [1] exponentiell gewichtet wurden, sind in Abb. 3.4 unten zu finden.

Die Darstellung der Zusammenhänge zwischen den durch die Umrechnung aus den Filterkoeffizienten bestimmten effektiven Reflexionsfaktoren wird in Kapitel 7 genauer diskutiert.

3.3.2. Die Realisierung in C++

Die Umrechnung der Filterkoeffizienten in die entsprechenden Reflexionsfaktoren wurde in Borland C++ realisiert, da hier die Abarbeitung der Berechnungen wesentlich schneller als unter MatLab erfolgt. Die Berechnung der Reflexionsfaktoren für den Sammelrücken übernimmt die ausführbare Datei **ainr.exe**. Der entsprechende Quellcode ist in der Datei **ainr.cpp** zu finden (siehe D.2).
Zum Compilieren und Linken dieser C-Quellcodedatei sind außerdem die folgenden C-Headerdateien von Nöten.

- sagrab.h (Berechnung eines Sammelgrabens, siehe D.6)
- sazelle.h (Berechnung einer Zelle des Sammelgrabens, siehe D.7)
- mytime.h (Ausgabe der aktuellen Uhrzeit, siehe D.5)

Wie bereits unter 3.1 erwähnt ist das Finden des maximal möglichen Maximums der Filterkoeffizienten sehr aufwendig.
Um herauszufinden, wie nah man am gewünschten Maximalwert liegt, muß man die Beendigung der von Hand gestarteten Berechnung abwarten. Danach läßt man sich das Ergebnis in MatLab anzeigen und erhöht den Maximalwert der Filterkoeffizienten, falls die gewonnenen Reflexionsfaktoren zu klein sind, bzw. macht diesen kleiner, falls die Reflexionsfaktoren zu groß oder sogar größer als Eins geworden sind. Bei dieser Berechnungsweise ist der Anwender immer gebunden. Er muß nach jedem Ergebnis den neuen Maximalwert bestimmen und eine neue Umrechnung starten. Mit den unter 6.2 beschriebenen Java Klassen wurde das Finden des maximal möglichen Reflexionsfaktors automatisiert, damit der Anwender entlastet wird.

3.3.3. Aufruf des Berechnungsprogrammes

Das Berechnungsprogramm wird von der MS-DOS Eingabeaufforderung aus gestartet. Zum Starten wechselt man in das Rootverzeichnis der Datenstruktur (siehe A).

Dort muß sich ebenfalls die Datei **ainr.exe** befinden oder man hat das Verzeichnis, in welchem sich diese exe-Datei befindet, dem `path` hinzugefügt. Der Aufruf lautet dann wie folgt.

```
ainr <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <lq>
```

Die Bedeutungen der einzelnen Parameter sind folgende:

- *rueckenbreite*
  Die Rückenlänge in Gräben, zum Beispiel `1000` für 1000 Gräben.

- *anz_pi*
  Die Anzahl der genutzten si-„Perioden“, z.B. `2` für $\pm 2\pi$.

- *kaiser_beta*
  Der Koeffizient der unter 2.4 genutzten Fensterfunktion, z.B. `20` für beta = 2,0.

- *si_max*
  Das Maximum der mit Hilfe der Funktion **ainr_v** in MatLab vorbereiteten Filterkoeffizienten (siehe 3.2), zum Beispiel `02` für ein Maximum von 0,02.

- *lq*
  Hier wird bestimmt, ob die Berechnung mit linearer (`l`) oder quadratischer (`q`) Betrachtung der Ausgänge durchgeführt werden soll. Linear bedeutet, daß die Ausgänge der einzelnen Zeilen direkt aufsummiert werden. Quadratisch bedeutet, daß die Ausgänge der Zeilen zuerst einzeln quadriert und anschließend aufsummiert werden.
  Im Laufe der Arbeit wurde die lineare Betrachtung favorisiert.

Alle Parameter werden durch ein Leerzeichen getrennt. Durch Bestätigen der Eingabe mit Return wird die Berechnung gestartet. Die Speicherung der Ergebnisse erfolgt automatisch entsprechend der in A beschriebenen Datenstruktur.

3.3.4. Visualisierung

Zur Visualisierung der mit Hilfe des eben beschriebenen C++ Programmes berechneten Reflexionsfaktoren wird wiederum MatLab verwendet. Die Funktion **ainr_l.m** lädt die entsprechenden Ergebnisse und zeigt diese in einer Figur an. Der Quellcode ist unter C.2 dargestellt.
Zur problemlosen Ausführung ist außerdem das MatLab-Skript **laden.m** (Laden einer ASCII-Datei, siehe C.15) erforderlich.

Zur Nutzung dieser Funktion gibt man folgendes in der MatLab-Kommandozeile ein.

```
ainr_l(rueckenbreite, anz_pi, kaiser_beta, si_max, lq);
```

Dabei haben die Parameter die gleichen Bedeutungen wie unter 3.3.3 beschrieben, allerdings müssen diese in Hochkommas gesetzt und durch Kommata getrennt werden.
Zum Laden und Anzeigen der berechneten Reflexionsfaktoren für den Sammelrücken mit 1000 Gräben Länge, $\pm 2\pi$ der si-Funktion, einem Kaiserkoeffizienten der Fensterfunktion von 3,4 sowie einem Maximum der Filterkoeffizienten von 0,1645 und linearer Betrachtung ist folgender Befehl einzugeben.

```
ainr_l('1000','2','34','1645','l');
```

3.4. Die Berechnung nach Methode 2 – Streu- und Sammelrücken

3.4.1. Berechnungsvorschrift

Hier werden beide Rücken mit gleich großen, aus den Filterkoeffizienten unter Berücksichtugung der Mehrfachreflexionen berechneten Reflexionsfaktoren gewichtet (siehe Abb. 3.2 rechts). Die schematische Darstellung der Vorgangsweise bei

dieser Methode ist in Figur 3.5 abgebildet. Sie ähnelt stark der unter 3.3.1 beschriebenen Berechnung, nur hier werden immer beide Rücken berechnet und die Gräben gleicher Nummer mit dem gleichen Reflexionsfaktor gewichtet.
Es ist zu beachten, daß der Reflexionsfaktor des Streugrabens positiv und der des entsprechenden Sammelgrabens negativ sein muß, falls der berechnete Reflexionsfaktor negativ ist. Das ist notwendig, damit nach Multiplikation beider Reflexionsfaktoren die Vorzeichenrichtigkeit bestehen bleibt.

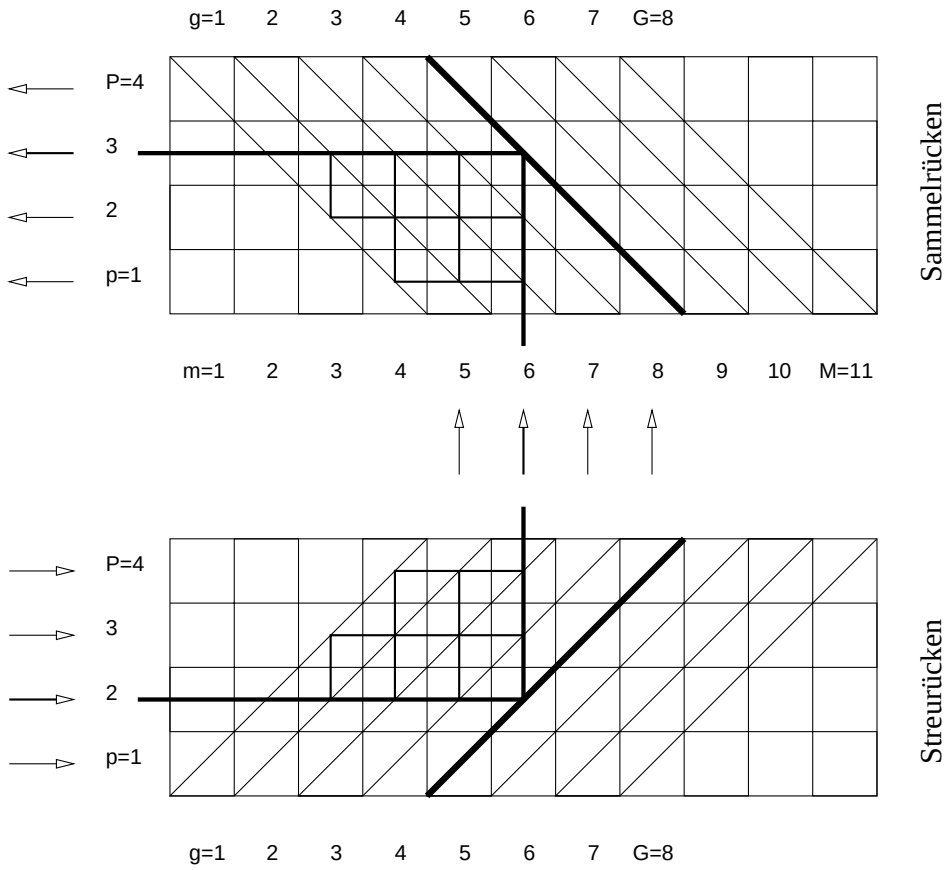


Abbildung 3.5.: Bestimmung der Reflexionsfaktoren (Streu- und Sammelrücken)

Es wird jeder vertikale Ausgang es Sammelrückens einzeln berechnet, wobei der jeweils in Hauptrichtung liegende vertikale Eingang des Senderückens mit Eins und alle anderen mit Null belegt werden. Dabei korrespondiert der in der p-ten Zeile liegende vertikale Eingang des Streurückens mit dem in der $(P-p+1)$-ten Zeile des Sammelrückens liegenden Ausgang.
Dies ergibt sich aus der Neigung der Gräben und der o.g. Betrachtung des Hauptwe-

ges. Aufgrund dessen sind Streu- und Sammelrücken nur über eine einzige Spalte der rechteckigen Gitterstruktur „verbunden“. Das ist immer die Spalte $m = g + p - 1$.

In Abb. 3.5 beträgt die Rückenlänge (Grabenanzahl) $G = 8$, die Rückenbreite (Zeilenanzahl) $P = 4$ und die erforderliche Spaltenanzahl der Gitterstruktur $M = G + P - 1 = 11$.
In der Grafik sind alle möglichen Eingänge, Ausgänge und Koppelspalten durch Pfeile gekennzeichnet. Für die aktuelle Zeile des Streurückens $p = 2$ und den aktuellen Graben $g = 5$ sind diese Pfeile fett gezeichnet und die möglichen Wellenflüsse durch die einzelnen Zellen sowohl im Sende- als auch im Empfangsrücken dargestellt. Bei diesem speziellen Fall ergibt sich der horizontale Ausgang der 6. Spalte ($m = 5+2-1 = 6$) des Senderückens als einziger Eingang des Empfangsrückens in genau der gleichen Spalte. Der zu betrachtende vertikale Ausgang des Sammelrückens ist jener in der 3. Zeile ($4 - 2 + 1 = 3$).

Nach einzelner Betrachtung der vertikalen Eingänge des Senderückens und der Berechnung aller entsprechenden vertikalen Ausgänge des Empfangsrückens ergibt sich die Gleichung zur Bestimmung des gesuchten Reflexionsfaktors zu

$$r_g = \pm \sqrt{\frac{\mid a_g \mid - \sum_{p=1}^{P} aus_p}{\left(\prod_{i=1}^{g-1} t_i\right)^2 \cdot \sum_{p=1}^{P} (fakt_p)^2}} \tag{3.4}$$

Dabei ist r_g der zu bestimmende Reflexionsfaktor des m-ten Grabens, a_g der zugrundeliegende Filterkoeffizient für diesen Graben. aus_p ist die vertikale Ausgangsfeldstärke der p-ten Zeile des Sammelrückens, und $\prod_{i=1}^{g-1} t_i$ ist das Produkt der Transmissionsfaktoren aller links vom aktuellen Graben liegenden Gräben in der entsprechenden Zeile. Durch die entsprechende Multiplikation mit $fakt_p$ werden die vertikalen Transmissionen für den Hauptweg in der bestimmten Koppelspalte berücksichtigt. Sowohl $fakt_p$ als auch das Produkt der links liegenden Transmissionsfaktoren gehen quadratisch ein, da diese im Streu- und im Sammelrücken durchlaufen werden.
Die Gleichung 3.4 hat zwei betragsmäßig gleiche reelle Lösungen. Es muß die Quadratwurzel gezogen werden, da auch der gesuchte Reflexionsfaktor quadratisch eingeht – zuerst findet die Reflexion am Graben im Streurücken statt und danach die

am entsprechenden Graben im Sammelrücken.

Um die gewünschte Filterwirkung zu erreichen, wird jeweils das r_g als effektiver Reflexionsfaktor verwendet, welches am nächsten beim jeweilig zugrundeliegenden Koeffizienten a_g liegt. Das ist erforderlich, damit die gewünschte Filterwirkung erhalten bleibt.

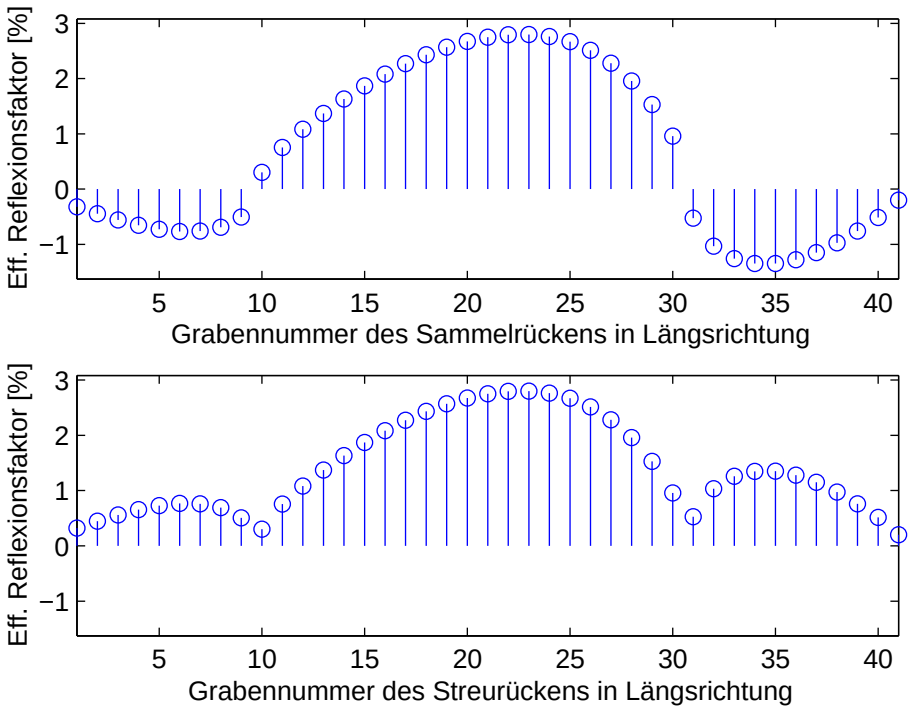


Abbildung 3.6.: Beispiel für eff. Reflexionsfaktoren nach Methode 2

Abb. 3.6 zeigt die auf Grundlage dieser Berechnung entstandenen effektiven Reflexionsfaktoren für den Streu- (unten) und den Sammelrücken (oben). Es ist zu erkennen, daß die Reflexionsfaktoren für beide Rücken in der gleichen Spalte betragsmäßig gleich groß sind. Beim Senderücken sind alle Reflexionsfakoren positiv, während beim Sammelrücken auch negative vorkommen.

Als wesentlicher Vorteil dieser Berechnungsmethode stellt sich die ca. doppelt so große Ausgangsleistung am Empfangsrücken heraus. Die Zusammenhänge zwischen Filterkoeffizienten und den daraus berechneten Reflexionsfaktoren bei dieser Methode werden in dieser Arbeit nicht genauer untersucht, sie verhalten sich jedoch

ähnlich wie bei Methode Eins.

3.4.2. Die Realisierung in C++

Auch diese Berechnung wurde in C++ realisiert, da sie hier wesentlich schneller abläuft. Für diese Berechnung ist das Programm **ainr2.exe** zuständig. Der entsprechende Quellcode ist in der Datei **ainr2.cpp** zu finden (siehe D.3).
Zum Compilieren und Linken dieser C-Quellcodedatei sind außerdem die folgenden C-Headerdateien von Nöten.

- stgrab.h (Berechnung eines Streugrabens, siehe D.8)
- stzelle.h (Berechnung einer Zelle des Streugrabens, siehe D.9)
- sagrab.h (Berechnung eines Sammelgrabens, siehe D.6)
- sazelle.h (Berechnung einer Zelle des Sammelgrabens, siehe D.7)
- mytime.h (Ausgabe der aktuellen Uhrzeit, siehe D.5)

3.4.3. Aufruf des Berechnungsprogrammes

Das Programm wird von der MS-DOS Eingabeaufforderung aus gestartet. Zum Starten wechselt man in das Rootverzeichnis der Datenstruktur (siehe A). Dort muß sich ebenfalls die Datei **ainr2.exe** befinden oder man hat das Verzeichnis, in welchem sich diese exe-Datei befindet, dem `path` hinzugefügt. Der Aufruf lautet dann wie folgt.

```
ainr2 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <lq>
```

Die Bedeutung der Parameter ist die gleiche wie unter 3.3.3 beschrieben.

3.4.4. Visualisierung

Zur Visualisierung der mit Hilfe des eben beschriebenen C++ Programmes berechneten Reflexionsfaktoren für den Streu- und Sammelrücken wird MatLab verwendet. Die Funktion **ainr2_l.m** (siehe C.4) lädt die entsprechenden Ergebnisse und zeigt diese in einer zweigeteilten Figur an.
Zur problemlosen Ausführung von **ainr2_l** ist zusätzlich das Skript **laden.m** (Laden einer ASCII-Datei, siehe C.15) erforderlich.

Zur Nutzung dieser Funktion gibt man folgendes in der MatLab-Kommandozeile ein.

```
ainr2_l(rueckenbreite, anz_pi, kaiser_beta, si_max, lq);
```

Dabei haben die Parameter die gleichen Bedeutungen wie unter 3.3.3 beschrieben, allerdings müssen diese in Hochkommas gesetzt und durch Kommata getrennt werden.
Zum Laden und Anzeigen der berechneten Reflexionsfaktoren für den Streu- und Sammelrücken mit 2205 Gräben Länge, $\pm 2\pi$ der si-Funktion, einem Kaiserkoeffizienten der Fensterfunktion von 3,4 sowie einem Maximum der Filterkoeffizienten von 0,00225781 und linearer Betrachtung ist die folgende Zeile in MatLab einzugeben.

```
ainr2_l('2205','2','34','00225781','l');
```

4. Berechnung der Ausgangsleistung

Die Berechnung der vertikalen Ausgänge des Sammelrückens wurde in C++ realisiert, denn auch diese Berechnung erfordert einen hohen Rechenaufwand, vor allem wenn die Rücken länger werden. Hierfür gibt es das Programm **aus.exe**. Der Quellcode ist in der Datei **aus.cpp** zu finden (siehe D.4).
Zum Compilieren und Linken dieser C-Quellcodedatei sind außerdem die folgenden C-Headerdateien von Nöten.

- stgrab.h (Berechnung eines Streugrabens, siehe D.8)
- stzelle.h (Berechnung einer Zelle des Streugrabens, siehe D.9)
- sagrab.h (Berechnung eines Sammelgrabens, siehe D.6)
- sazelle.h (Berechnung einer Zelle des Sammelgrabens, siehe D.7)
- mytime.h (Ausgabe der aktuellen Uhrzeit, siehe D.5)

4.1. Aufruf des Berechnungsprogrammes

Dieses Programm wird von der MS-DOS Eingabeaufforderung aus gestartet. Zum Starten wechselt man in das Rootverzeichnis der Datenstruktur (siehe A). Dort muß sich ebenfalls die Datei **aus.exe** befinden oder man hat das Verzeichnis, in welchem sich diese exe-Datei befindet, dem `path` hinzugefügt. Der Aufruf lautet wie folgt und ist in einer Zeile einzugeben.

```
aus <rueckenbreite> <rstart> <anz_pi>
    <kaiser_beta> <si_max> <lq>
```

Die Bedeutung der Parameter ist die gleiche wie unter 3.3.3 beschrieben. Für den zusätzlichen Parameter *rstart* gibt es drei Möglichkeiten.

- Berechnung der Ausgänge für die komplette Filterstruktur (Streu- und Sammelrücken), wobei der Streurücken mit einer e-Funktion gewichtet ist und die Reflexionsfaktoren von **ainr** (siehe 3.3.2) berechnet wurden. Dazu gibt man für *rstart* den Wert des Reflexionsfaktors in der ersten Spalte des Streurückens (ganz links) an, zum Beispiel `00985` für r = 0,00985.
- Berechnung der Ausgänge nur für den Sammelrücken. Hier wird davon ausgegangen, daß am unteren horizontalen Eingang des Streurückens ein konstantes E-Feld vorliegt. Will man diese Berechnungsart durchführen lassen, so gibt man für *rstart* `-1` ein.
- Um die Ausgänge für die Berechnung der Reflexionsfaktoren nach **ainr2** (siehe 3.4.2) bestimmen zu lassen, gibt man für *rstart* `-2` an.

Die Speicherung der Ergebnisse erfolgt automatisch entsprechend der in A beschriebenen Datenstruktur.

4.2. Visualisierung des Filterausgangs

Zur Visualisierung der mit Hilfe des eben beschriebenen C++ Programmes berechneten vertikalen Filterausgänge wird MatLab verwendet. Die Funktion **aus_l.m** (siehe C.5) lädt die entsprechenden Ergebnisse und zeigt diese in einer Figur an.
Zur problemlosen Ausführung sind folgende MatLab-Skripten erforderlich:

- laden.m (Laden einer ASCII-Datei, siehe C.15)

- time.m (Ausgabe des Datums und der aktuellen Uhrzeit, siehe C.20)

Der Aufruf dieser Funktion geht in MatLab folgendermaßen vonstatten.

```
aus_l(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
```

Dabei haben die Parameter die gleichen Bedeutungen wie unter 3.3.3 und 4.1 beschrieben, allerdings müssen diese in Hochkommas gesetzt und durch Kommata getrennt werden. Zum Laden und Anzeigen der Ausgangsleistungsdaten für eine Rückenlänge von 1000 Gräben, dem Startwert der Reflexionsfaktoren für den Streurücken (Wichtung mit e-Funktion) von 0,00985, $\pm 2\pi$ der si-Funktion, einem Kaiserkoeffizienten der Fensterfunktion von 5,0 sowie einem Maximum der Filterkoeffizienten von 0,1667 und linearer Betrachtung ist die folgende Zeile einzugeben.

```
aus_l('1000','00985','2','50','1667','l');
```

Die dargestellte MatLab-Figur enthält die folgenden Teil-Grafiken:

- die zugrundeliegenden Reflexionsfaktoren für den Streu- (falls verwendet) und Sammelrücken oben links,
- die absolute Ausgangsleistung in Abhängigkeit von der Freiraumwellenlänge des Lichts oben rechts,
- die Phase der Ausgangsleistung in der Mitte links,
- die relative Ausgangsleistung in logarithmischer Darstellung in der Mitte rechts,
- diverse bestimmte Werte und den Ort der Quelldateien in der dritten Zeile der Figur (unten).

4.3. Visualisierung des horizontalen Filterausgangs des Streurückens

Die Funktion **hori_l.m** (siehe C.13) lädt die Ergebnisse der Berechnung der horizontalen Ausgänge des Streurückens, falls beide Rücken bei der Ausgangsberechnung betrachtet wurden, und zeigt diese in einer Figur an.
Zur problemlosen Ausführung ist außerdem das MatLab-Skript **laden.m** (Laden einer ASCII-Datei, siehe C.15) erforderlich.

Der Aufruf dieser Funktion erfolgt in der MatLab-Kommandozeile und sieht folgendermaßen aus.

```
hori_l(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
```

Die Reihenfolge und Bedeutungen der erforderlichen Parameter sind die gleichen wie unter 4.2 beschrieben.

In der angezeigten MatLab-Figur werden im oberen Bereich Betrag und Phase der Ausgänge bei der Mittenfrequenz einzeln aufgezeigt. Zwei 3D-Grafiken zeigen den Betrags- und Phasenverlauf der horizontalen Ausgänge des Streurückens über dessen gesamter Länge für alle berechneten Frequenzen bzw. Lichtwellenlängen.

5. Verteiltes Rechnen

Mit Hilfe des Java Packages *mibec.share* können verschiedenste Berechnungen auf mehrere Computer verteilt werden.

Das Grundprinzip des „Verteilten Rechnens“ besteht darin, daß mehrere Personalcomputer als Clients agieren, und ein Server eine auszuführende Berechnung an einen freien Client vergibt. Alle Personalcomputer, von welchen aus über LAN[1] oder das Internet eine Verbindung zu diesem Server via TCP/IP[2] aufgebaut werden kann, können als Client fungieren.
Vom Server-Rechner aus können dann Berechnungen in Auftrag gegeben werden. Die für eine Berechnung erforderlichen Daten werden vom Server zum Client gesandt, dieser führt die Berechnung durch und sendet seine Ergebnisse nach Beendigung via Netzwerk zurück zum Server. Der Server speichert diese Ergebnisse ab oder leitet sie entsprechend weiter, so daß sie weiterverwendet werden können.

Alle benötigten *mibec* Klassen stehen mit dem Java Archiv **mibec.jar** zur Verfügung. Die Quellcodes der erforderlichen Java Klassen werden nicht im Anhang dargestellt, da sie nicht in direkter Beziehung zum Thema dieser Diplomarbeit stehen.

Die Umrechnung der Filterkoeffizienten in die entsprechenden Reflexionsfaktoren ist aufgrund des unbekannten Maximalwertes der Filterkoeffizienten sehr langwierig.

[1] Local Area Networt = Lokales Netzwerk
[2] Transport Control Protocol / Internet Protocol – Das Transport-Protokoll für Daten im Internet

Mit Hilfe der im folgenden beschriebenen Java Klassen können zum Beispiel genau diese Berechnungen auf mehrere Computer verteilt werden. In Abhängigkeit von der Anzahl verfügbarer Clients können mehrere Berechnungen gleichzeitig ausgeführt werden. Das hat eine enorme Zeitersparnis zur Folge.

5.1. Der Server

5.1.1. Funktionsweise

Der Server ist in der Klasse **mibec.share.Server** realisiert. Nach dem Start registriert sich dieser als RMI[3]-Objekt. Jetzt ist der Server über das Netzwerk unter der IP des Serverrechners erreichbar. Das ist notwendig, da sich jeder Client (siehe 5.2) unter anderem zuerst beim Server anmelden muß. Auch zum Starten von Berechnungen ist eine Verbindung zum Server erforderlich.

Wird eine Berechnung in Auftrag gegeben (siehe 5.3), so sucht der Server einen freien Client. Ist ein Client gefunden, sendet der Server den Berechnungsauftrag zu diesem Client, zusammen mit den erforderlichen Eingabedaten und allen zum Start der Berechnung auf dem Clientrechner notwendigen Parametern. Jeder Auftrag bekommt eine Identifikationsnummer (JobID). Das ist nötig, damit Statusmeldungen und Ergebnisse richtig zugeordnet werden können. Wurde eine Berechnung erfolgreich in Auftrag gegeben (das heißt ein Client hat begonnen, die Berechnung durchzuführen), so gibt der Server die IP des Clientrechners und die Identifikationsnummer an den Job-Starter zurück. In der Statusanzeige wird angegeben, daß der Auftrag mit der vergebenen JobID gestartet wurde.
Wurde stattdessen kein freier Client gefunden, so wird dies ebenfalls an den Job-Starter zurückgegeben. Falls aus irgendeinem Grund keine Verbindung zu einem angemeldeten Client aufgebaut werden kann, so wird dieser aus der Liste entfernt.

Ist eine Berechnung abgeschlossen, so empfängt der Server die berechneten Daten vom Client und speichert sie entsprechend der Identifikationsnummer auf dem Ser-

[3]Java Remote Method Invocation [11].

verrechner ab oder leitete sie weiter. Der betreffende Client ist nun bereit, einen weiteren Berechnungsauftrag entgegenzunehmen.
In der Statusanzeige werden im Anschluß daran die Beendigung des Auftrages, die Dauer der Berechnung und die Namen der Ausgabedateien angegeben.

5.1.2. Statusanzeige

Während des Serverbetriebes wird immer dann eine Statusanzeige ausgegeben, wenn sich ein Client aus einem der o.g. Gründe meldet. Dabei wird jeweils folgendes in einer Zeile ausgegeben.

- Die *IP* des Clientrechners.
- Die *Anzahl* der bereits von diesem Computer durchgeführten Berechnungen.
- Die *Uhrzeit* der letzten Meldung.
- Die Identifikationsnummer (*JobID*) der aktuellen Berechnung, falls ein Client gerade beschäftigt ist.
- Die letzte *Statusmeldung*. Das kann die Information über die Bereitschaft des Clients (idle), die Information über den Abschluß einer Berechnung oder die letzte Systemausgabe einer gerade laufenden Berechnung sein.

5.1.3. Aufruf

Der Server ist eine reine Konsolenanwendung, somit kann er auf jedem Computer und jeder Workstation gestartet werden. Zur Ausführung werden nur eine Java Runtime Environment (Ausführungsumgebung) und das Archiv der *mibec* Java Klassen benötigt. Der Start des Servers geht folgendermaßen von statten:

```
java -cp mibec.jar mibec.share.Server <pause>
```

Der Parameter *pause* gibt die Zeit zwischen den Statusmeldungen eines Clients während einer ausgeführten Berechnung in Sekunden an.
Als weitere Parameter können die IPs der bereits zur Verfügung stehenden Clients angegeben werden. Das ist wichtig, falls der Server einmal neu gestartet werden muß, wobei sich die Clients bereits angemeldet hatten.

5.2. Der Client

5.2.1. Funktionsweise

Um überhaupt Berechnungen ausführen zu können, werden Client-PCs benötigt. Auf diesen Rechnern werden die eigentlichen Arbeiten ausgeführt. Je mehr davon vorhanden sind, desto mehr Berechnungen können gleichzeitig durchgeführt werden.

Auch der Client registriert sich nach dem Start als Java RMI-Objekt, so daß er vom Server erreichbar ist. Vorher wird jedoch (falls möglich) die Version der *mibec* Java Klassen überprüft (siehe 5.4).
Ist die Registrierung erfolgt, so versucht der Client sich beim übergebenen Share-Server zu melden. Wenn das nicht auf Anhieb gelingt, wird in Abständen von einer Minute immer wieder ein erneuter Anmeldungsversuch durchgeführt. Der Client ist dann bereit Aufträge vom Server entgegenzunehmen, das heißt er ist idle.

Erteilt der Server einen Berechnungsauftrag, so prüft der Client zuerst mit Hilfe der unter 5.4 beschriebenen Updatefunktion das Vorhandensein der neuesten Version des Berechnungsprogrammes. Ist diese nicht vorhanden, so wird sie heruntergeladen. Anschließend wird die Berechnung gestartet. In bestimmten Zeitintervallen sendet der Client den Stand der ausgeführten Berechnung zum Server. Nach Abschluß der Rechnung werden die Ergebnisse zum Server geschickt. Ist die Verbindung zum Server gestört, so versucht der Client einmal pro Minute, seine Ergebnisse zu senden, bis dies schließlich gelingt. Wurden die Ergebnisse erfolgreich übertragen, so ist der Client bereit, den nächsten Auftrag vom Server anzunehmen.

5.2.2. Starten des Clients

Die Implementierung des Clients ist in der Klasse **mibec.share.Client** zu finden. Zum Starten des Clients auf einem beliebigen Rechner sind die Java Runtime Environment und das Archiv der *mibec* Java Klassen erforderlich. Der Aufruf erfolgt in der MS-DOS Eingabeaufforderung folgendermaßen (in einer Zeile):

```
java -cp mibec.jar mibec.share.Client
     <share_server> <update_server>
```

Für den Parameter *share_server* ist die IP Adresse des Rechners, auf dem der Server (siehe 5.1.3) gestartet wurde, anzugeben.
Der Parameter *update_server* ist optional. Dafür muß nur dann eine IP Adresse angegeben werden, wenn die unter 5.4 beschriebenen Server (Remote- und HTTP-Server) auf einem anderen Rechner als der Share-Server laufen.

5.3. Starten von Berechnungen

Zum Starten von Berechnungen, welche mit Hilfe des Servers auf einen Client verteilt werden, stehen zwei Möglichkeiten zur Verfügung.

Die erste Version startet nur die Berechnung und gibt anschließend aus, auf welchem Client die Berechnung läuft. Danach ist dieses Terminal wieder frei und kann zum Starten weiterer Aufträge genutzt werden. Mit dieser Methode können keine Aussagen über den Fortschritt und das Ende der gestarteten Berechnung gemacht werden, Berechnungen können nur vom Server-Computer aus in Auftrag gegeben werden, da die Speicherung der Ergebnisse der Server selbst übernimmt. Näheres ist unter 5.3.1 zu finden.

Manchmal möchte man allerdings eine Berechnung von einem anderen Rechner als dem Server-Computer aus oder mit Hilfe einer eigenen Java Klasse starten. Genau das ist mit der zweiten Version des Job-Starters möglich. Hier kann man auch

den Fortschritt der Berechnung verfolgen und auf deren Beendigung warten. Diese Startmethode wird unter 5.3.2 beschrieben.

5.3.1. Einfacher Start

Mit Hilfe der Klasse **mibec.share.Job** wird eine Berechnung in Auftrag gegeben. Diese Methode kann nur auf dem Server-Rechner selbst verwendet werden.
Sämtliche für die Ausführung der Berechnung benötigte Information wird in Form von Parametern an den Aufruf dieser Java Klasse angehängt. Der allgemeine Aufruf des Job-Starters sieht folgendermaßen aus.

```
java -cp mibec.jar mibec.share.Job <command>
     <anzahl_eingabedateien> <anzahl_ausgabedateien>
     <anzahl_zusatz> <...> <...> <...>
```

Dieser Befehl ist in einer einzigen Zeile einzugeben, das heißt die Eingabetaste ist nur nach dem letzten Parameter zu drücken.
Nun zur Erläuterung aller Parameter.

- *command*
  Das ist der komplette Dateiname des auszuführenden Berechnungsprogrammes, z.B. `ainr.exe`.

- *anzahl_eingabedateien*
  Hier wird die Anzahl der benötigten Eingabedateien für das Berechnungsprogramm angegeben. Bei **ainr** ist diese z.B. `1`.

- *anzahl_ausgabedateien*
  Die Anzahl der Ausgabedateien für die Speicherung der Ergebnisse muß hier angegeben werden. Bei unserem Beispiel **ainr** ist auch diese Anzahl `1`.

- *anzahl_zusatz*

  Hier wird die Anzahl der zusätzlichen Parameter für den Aufruf des Berechnungsprogrammes angegeben.

Im Anschluß an diese festen Parameter folgen alle Dateinamen der Eingabedateien, dann die Dateinamen der Ausgabedateien und zum Abschluß die zusätzlichen Argumente für das Berechnungsprogramm. Alle Parameter werden durch Leerzeichen voneinander getrennt.

5.3.2. Start und Beobachtung

Um Aussagen über den Fortschritt und die Beendigung einer gestarteten Berechnung machen zu können, wurde die Klasse **mibec.share.Job2** erstellt. Sie ermöglicht außerdem, Anwendungen von einem beliebigen Rechner aus zu starten, welcher den Share-Server über das Netzwerk erreichen kann.
Diese Klasse kann sowohl über die MS-DOS Eingabeaufforderung bzw. von einem Terminal aus als auch von anderen Java Klassen aufgerufen werden. Hier soll allerdings nur die erste Variante beschrieben werden. Der allgemeine Start einer Berechnung mit Beobachtungsfunktion sieht folgendermaßen aus:

```
java -cp mibec.jar mibec.share.Job2 <share_server> <command>
     <anzahl_eingabedateien> <anzahl_ausgabedateien>
     <anzahl_zusatz> <...> <...> <...>
```

Dieser Befehl ist in einer einzigen Zeile einzugeben, das heißt die Eingabetaste ist nur nach dem letzten Parameter zu drücken.
Für den Parameter *share_server* ist die IP des Share-Servers, welcher die Berechnungen verteilt, anzugeben. Die Bedeutungen der restlichen Parameter sind die gleichen wie unter 5.3.1 beschrieben.

5.4. Automatische Updatefunktion für die Clients

Um Änderungen an den *mibec* Java Klassen oder an den auszuführenden Berechnungsprogrammen schnell und unkompliziert an die Clients weiterzugeben, wurde eine automatische Updatefunktion integriert. Dazu ist es erforderlich, daß auf einem Rechner, sinnvollerweise auf dem Servercomputer, ein *mibec* Remote-Server und ein HTTP-Server laufen.

Die Clients prüfen beim Start auf eine neue Version des Java Archivs **mibec.jar** und vor jeder Berechnung auf das Vorhandensein des auszuführenden Berechnungsprogrammes in der aktuellen Version, indem sie zum oben genannten Remote-Server verbinden. Falls ein Update erforderlich ist, sendet der Remote-Server die URL der neuen Datei zum Client. Dieser lädt dann die aktuelle Version über den HTTP-Server selbständig herunter.

Der Remote-Server ist in der Klasse **mibec.remote.Server** realisiert. Auch er registriert sich als Java RMI-Objekt auf dem Computer, wo er gestartet wurde. Der Aufruf sieht folgendermaßen aus.

```
java -cp mibec.jar mibec.remote.Server <www-rootdirectory>
```

Für die *www-rootdirectory* ist das Rootverzeichnis des HTTP-Servers anzugeben. Die aktuelle Version des Java Archivs **mibec.jar** muß sich im Unterverzeichnis **download** und die aktuellen Berechnungsprogramme müssen sich im Unterverzeichnis **java/share** befinden. Die eben genannten Unterverzeichnisse verstehen sich relativ zur *www-rootdirectory*.

6. Berechnungsoptimierung

Für die optimale Nutzung des unter Kapitel 5 erläuterten „Verteilten Rechnens“ und zur Entlastung des Anwenders im Zusammenhang mit dem Thema dieser Diplomarbeit wurden einige Batch-Dateien und spezielle Java Klassen erstellt bzw. programmiert.

Alle im folgenden beschriebenen Eingaben finden in der MS-DOS Eingabeaufforderung statt. Man muß sich dazu im Rootverzeichnis der unter A beschriebenen Datenstruktur befinden, damit die Quelldateien (Eingabedaten für die Berechnung) gelesen und die berechneten Ergebnisse entsprechend abgespeichert werden können. Das Archiv der *mibec* Java Klassen, gespeichert in der Datei **mibec.jar**, muß sich ebenfalls im Hauptverzeichnis der Datenstruktur befinden.

6.1. Einzelne Berechnungen

Bei dieser Form der Berechnung kommt dem Menschen als Anwender eine wesentliche Rolle zu. Er koordiniert die Berechnungen und startet jede einzelne von Hand. Das Starten wird durch die im folgenden beschriebenen Batch-Dateien erleichtert.

6.1.1. Berechnung der Reflexionsfaktoren

Berechnung nur für den Sammelrücken

Das C++ Programm zu dieser Art der Berechnung der Reflexionsfaktoren wurde bereits unter 3.3 beschrieben. Zum Start einer Berechnung steht die Batch-Datei **sjainr.bat**, deren Inhalt in E.3 dargestellt ist, zur Verfügung. Der Aufruf einer solchen Berechnung sieht so aus:

```
sjainr <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <lq>
```

Alle Parameter haben die Bedeutungen wie unter 3.3.3 dargestellt.

Berechnung für Streu- und Sammelrücken

Das Programm, welches diese Berechnung durchführt, ist unter 3.4 beschrieben. Zur Vereinfachung des Starts einer solchen Berechnung wurde die Batch-Datei **sjainr2.bat** (siehe E.4) erstellt. Der Aufruf sieht folgendermaßen aus:

```
sjainr2 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <lq>
```

Die Bedeutungen der Parameter sind unter 3.3.3 beschrieben.

6.1.2. Berechnung der Ausgangsleistung

Das C++ Programm, welches die im folgenden beschriebenen Berechnungen ausführt, ist in Kapitel 4 beschrieben.

Berechnung nur für den Sammelrücken

Mit Hilfe der Batch-Datei **sjaus-1.bat**, deren Inhalt unter E.6 dargestellt ist, kann eine Berechnung der Ausgangsleistung nur für den Sammelrücken gestartet werden. Der Aufruf sieht so aus:

```
sjaus-1 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <lq>
```

Die Bedeutungen der Parameter sind unter 3.3.3 beschrieben.

Berechnung für die komplette Filterstruktur

Zum Start einer Berechnung der Ausgangsleistung für die komplette Filterstruktur steht die Batch-Datei **sjaus.bat**, deren Inhalt unter E.5 abgedruckt ist, zur Verfügung. Der Aufruf einer solchen Berechnung sieht so aus:

```
sjaus <rueckenbreite> <rstart> <anz_pi>
      <kaiser_beta> <si_max> <lq>
```

Alle Parameter haben die Bedeutungen wie unter 3.3.3 beschrieben. Für den Parameter *rstart* wird der Wert des Reflexionsfaktors in der ersten Spalte des Streurückens (ganz links) angegeben, z.B. `00985` für r = 0,00985.

Berechnung der Ausgänge für ainr2

Die Batch-Datei **sjaus-2.bat** (siehe E.7) dient der Erleichterung des Starts einer Ausgangsleistungsberechnung für die durch **ainr2** bestimmten Reflexionsfaktoren. Durch Eingabe des folgenden Befehls in der MS-DOS Eingabeaufforderung wird die Berechnung in Aufrag gegeben.

```
sjaus-2 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <lq>
```

Die Bedeutungen der Parameter sind unter 3.3.3 beschrieben.

6.2. Berechnung nach Methode 1 – Nur Sammelrücken

Mit der Möglichkeit, Share-Jobs auch direkt aus anderen Java Klassen heraus starten zu können (siehe 5.3.2), wurde die Bestimmung des maximal möglichen maximalen Reflexionsfaktors bei der Umrechnung von Filterkoeffizienten in die entsprechenden effektiven Reflexionsfaktoren für den Sammelrücken (**ainr.exe**, siehe 3.3) für eine bestimmte Rückenlänge vollständig automatisiert. Ist der maximale Wert gefunden, so werden direkt im Anschluß daran die Filterausgänge für die bestimmten Reflexionsfaktoren mit Hilfe des C++ Programmes **aus.exe** (siehe Kapitel 4) berechnet.

Die Automatisierung ist in den Java Klassen **mibec.diplom.Diplom** und **mibec.diplom.AinrFinder** realisiert. Während die Klasse Diplom allgemeine Funktionen zur Verfügung stellt (z.B. die Berechnung der Filterkoeffizienten mit einem neuen Maximalwert), wird die eigentliche Arbeit in der Klasse AinrFinder erledigt.

Der Aufruf einer solchen automatisierten Berechnung wurde mit Hilfe der Batch-Datei **ainrfind.bat** erleichtert und der bekannten Parameterübergabe angepaßt. Dabei wurde auch die IP des Servers fest in dieser Startdatei verankert, da sich dieser eher selten ändert. Der Inhalt dieser Batch-Datei ist unter E.1 zu finden.
Um eine problemlose Ausführung zu gewährleisten, ist es notwendig, daß man sich im Rootverzeichnis der unter A beschriebenen Datenstruktur befindet. Das Archiv der *mibec* Java Klassen (**mibec.jar**) muß sich ebenfalls in diesem Verzeichnis befinden. Der Start sieht folgendermaßen aus:

```
ainrfind <rueckenbreite> <anz_pi> <kaiser_beta>
         <si_max> <delta_start>
```

Diese Eingabe ist in einer Zeile vorzunehmen. Die Bedeutungen der Parameter sind

teilweise unter 3.3.3 beschrieben. Erweiterungen bzw. Abweichungen werden nun dargelegt:

- *si_max*

  Hier wird der Startwert für das Maximum der Filterkoeffizienten angegeben. Dabei ist zu beachten, daß genau diese Koeffizienten mit Hilfe der unter 3.2 beschriebenen MatLab-Funktion vorbereitet wurden. Im Vergleich zu sonstigen Aufrufen ist es wichtig, daß vor diesem Wert das führende Komma angegeben wird. Will man zum Beispiel bei einem Maximum von 0,07 beginnen, so ist `.07` für diesen Parameter anzugeben.

- *delta_start*

  Wichtig ist auch eine bestimmte Schrittweite. Der Startwert dieser wird hier angegeben, z.B. `.001` für 0,001.

Das Programm startet nun den ersten Share-Job zur Berechnung der Reflexionsfaktoren für das angegebene Maximum der Filterkoeffizienten. Dazu verbindet es zum in der Batch-Datei angegebenen Share-Server und teilt ihm seinen Berechnungswunsch mit. Die Reflexionsfaktoren werden vom beauftragten Client mit dem C++ Programm **ainr.exe** bestimmt.
Ist eine Berechnung vollendet, interpretiert das Programm das Ergebnis und ermittelt den neuen Maximalwert der zugrundeliegenden Filterkoeffizienten. Die nächste Berechnung wird nach Erstellung der neuen Filterkoeffizienten gestartet. Dieses Spiel wiederholt sich so lange, bis ein sinnvolles Ergebnis erzielt wurde und sich eine im Programm festgelegte Anzahl der Nachkommastellen des maximalen Filterkoeffizienten nicht mehr ändert.
Ist dieser Wert gefunden, so werden direkt die Filterausgänge für diese Reflexionsfaktoren berechnet. Vor Beendigung des Programmes wird noch eine Übersicht der bestimmten Werte im Verzeichnis, in dem auch die Reflexionsfaktoren gespeichert werden, in der Datei **info.txt** abgelegt.

Die Ergebnisse können dann mit den unter 3.3.4 (Reflexionsfaktoren) und 4.2 (Filterausgänge) beschriebenen MatLab-Funktion visualisiert werden.

6.3. Berechnung nach Methode 2 – Streu- und Sammelrücken

Auch die Bestimmung des maximal möglichen maximalen Reflexionsfaktors bei der Umrechnung von den Filterkoeffizienten in die Reflexionsfaktoren für den Streu- und Sammelrücken (**ainr2.exe**, siehe 3.4) für eine bestimmte Rückenlänge wurde vollständig automatisiert. Ist der maximale Wert gefunden, so werden direkt im Anschluß daran die Filterausgänge für die bestimmten Reflexionsfaktoren mit Hilfe des C++ Programmes **aus.exe** (siehe Kapitel 4) berechnet.

Die Automatisierung ist in der Java Klasse **mibec.diplom.Ainr2Finder** realisiert. Auch hier wird auf die allgemeinen Funktionen in der Klasse **mibec.diplom.Diplom** zurückgegriffen.

Der Aufruf einer solchen automatisierten Berechnung wurde mit Hilfe der Batch-Datei **ainr2find.bat** (siehe E.2) erleichtert und der bekannten Parameterübergabe angepaßt.
Um eine problemlose Ausführung zu gewährleisten, ist es notwendig, daß man sich im Rootverzeichnis der unter A beschriebenen Datenstruktur befindet. Das Archiv der *mibec* Java Klassen (**mibec.jar**) muß sich ebenfalls in diesem Verzeichnis befinden. Der Start sieht folgendermaßen aus:

```
ainr2find <rueckenbreite> <anz_pi> <kaiser_beta>
          <si_max> <delta_start>
```

Diese Eingabe ist in einer Zeile vorzunehmen. Die Bedeutungen der Parameter sind teilweise unter 3.3.3 beschrieben. Erweiterungen bzw. Abweichungen wurden bereits unter 6.2 erklärt.

Die Funktionsweise dieser Java Klasse ist die gleiche wie unter 6.2 beschrieben, allerdings wird zur Berechnung auf den Client-Computern das C++ Programm

ainr2.exe verwendet. Die Übersicht der bestimmten Werte wird im Verzeichnis, in dem auch die Reflexionsfaktoren für den Streurücken gespeichert werden, in der Datei **info.txt** abgelegt.

Die Ergebnisse können mit den unter 3.4.4 (Reflexionsfaktoren) und 4.2 (Filterausgänge) beschriebenen MatLab-Funktionen visualisiert werden.

7. Ergebnisse

Wie bereits erwähnt, wurden im Zusammenhang mit dieser Arbeit mehrere Programmiersprachen verwendet. Dies ist daraus entstanden, daß bestimmte Probleme in verschiedenen Programmiersprachen gar nicht oder nur mit einem höheren Aufwand lösbar sind.

Die Sprache C ist eine der ältesten und vor allem schnellsten Programmiersprachen. Aus dem zweiten Grunde wurde sie für die langwierigen Berechnungen eingesetzt.
Zur Darstellung der Berechnungsergebnisse eignet sich MatLab bestens. Hier sind genau solche Funktionen in verschiedener Art und Weise bereits vorhanden. Sie brauchen nur noch angewendet werden. In C bzw. C++ oder Java gestaltet sich die Visualisierung wesentlich schwieriger. Auch können die Darstellungen dabei nicht so einfach im gewünschten Format (Encapsulated Postscript) abgespeichert werden. In MatLab steht genau dafür einen Befehl bereit.
Die Kommunikation zwischen zwei oder mehreren Computer über ein Netzwerk gestaltet sich mit der neuen Programmiersprache Java relativ einfach. Hierzu sind diverse Klassen und vor allem die Remote Method Invocation (RMI) vorhanden. Auf der Basis dieser Java Klassen ist das verteilte Rechnen entstanden.

Die Vorteile der einzelnen Programmiersprachen (MatLab, C++ und Java) wurden bestmöglich ausgenutzt. Dies hat zu einer enormen Zeitersparnis geführt sowie schnelle und vielfältige Berechnungsergebnisse in Bezug auf den Entwurf der Filter ermöglicht.

7.1. Ergebnisse der Berechnungen

Anhand der Berechnungen zur Bestimmung der Reflexionsfaktoren für den Sammelrücken nach Methode 1 (siehe 3.3) bei Nutzung von $\pm 2\pi$ der *si*-Funktion für die Filterkoeffizienten werden nun die Zusammenhänge zwischen Rückenlänge, genutztem Koeffizienten β für das Kaiserfenster, Maximum der Filterkoeffizienten und dem daraus resultierenden Maximum der effektiven Reflexionsfaktoren dargestellt.

Nach der Formel für die Bestimmung der Reflexionsfaktoren für den Sammelrücken wurden u.a. aus den in B.1 aufgelisteten Maxima der Filterkoeffizienten die entsprechenden Reflexionsfaktoren berechnet. Die Ergebnisse aller durchgeführten Berechnungen sind im Anhang unter B.2 ($\pm 2\pi$) und B.3 ($\pm \pi$) zu finden.

Die Ortsfunktion der effektiven Reflexionsfaktoren

Bei jeder Umrechnung ergibt sich ein minimaler und ein maximaler Reflexionsfaktor. Der maximale effektive Reflexionsfaktor nach einer bestimmten Umrechnung befindet sich etwas rechts von der Mitte des Rückens. Der minimale effektive Reflexionsfaktor befindet sich im betrachteten Fall zwischen $+\pi$ und $+2\pi$ – also noch weiter rechts, er ist negativ. Zur besseren Verdeutlichung sind in Abb. 7.1 die effektiven Reflexionsfaktoren für eine Rückenlänge von 1000 Gräben dargestellt. Die o.g. Extremwerte sind durch Pfeile gekennzeichnet.

Deutung der Abbildungen

Um die im folgenden beschriebenen Grafiken besser deuten zu können, wurden die verwendeten Koeffizienten β_{Kaiser} für die Fensterfunktion nach Kaiser (siehe 2.2) fest einer bestimmten Linienfarbe zugeordnet.

Kaiserkoeffizient β	Linienfarbe
2,0	blau
3,4	rot
5,0	schwarz
ungewichtet	grün

Der maximale Reflexionsfaktor

Das Abbruchkriterium für die Bestimmung des maximalen Reflexionsfaktors liegt in der Tatsache, daß der zu bestimmende Reflexionsfaktor bei zu großem Maximum der Filterkoeffizienten größer als Eins wird. Das ist allerdings praktisch nicht möglich. Dieses Phänomen macht sich bei kleineren Rückenlängen (bis ca. 2000 Gräben) nicht bemerkbar, da hier der physikalisch bedingte Maximalwert für den effektiven Reflexionsfaktor von $r_{max} = 2,8\%$ [1] vorher erreicht wird.
Bei größeren Rückenlängen hingegen kann man diesen Wert ($r_{max} = 2,8\%$) nicht mehr erreichen. Um trotzdem die maximal mögliche Leistung am Filterausgang bereitstellen zu können, will man sich diesem Punkt (bevor der Reflexionsfaktor größer als Eins wird) möglichst gut annähern. In diesem kritischen Bereich machen sich infinitesimal kleine Änderungen des Maximums der Filterkoeffizienten sehr stark bemerkbar.

Der Zusammenhang zwischen dem maximalen bzw. minimalen effektiven Reflexionsfaktor und dem zugrundeliegenden Maximum der Filterkoeffizienten wird hier am Beispiel der Rückenlänge 2205 und dem Koeffizienten $\beta_{Kaiser} = 3,4$ des Kaiserfensters dargestellt. In Abb. 7.2 sind alle für diese Rückenlänge und dieses Kaiserfenster berechneten Ortsfunktionen der Reflexionsfaktoren eingezeichnet, Abb. 7.3 zeigt die Abhängigkeit des errechneten maximalen (blaue Kurve) und minimalen (rote Kurve) Refexionsfaktors vom Maximum der zugrundeliegenden Filterkoeffizienten. Es ist zu erkennen, daß dieser Zusammenhang nicht linear ist. Sehr deutlich wird dies am raschen Anwachsen des Betrages des minimalen effektiven Reflexionsfaktors.

Der kritische Bereich (dort, wo der errechnete Reflexionsfaktor größer als Eins wird) ist von der Rückenlänge (Anzahl der Gräben) abhängig. In Abb. 7.4 ist das zulässige Maximum der Filterkoeffizienten in Abhängigkeit von der Rückenlänge dargestellt, Abbildung 7.5 zeigt die Maximalwerte der bestimmten effektiven Reflexionsfaktoren für die entsprechenden Rückenlängen.
Dieser Zusammenhang stellt sich sowohl für das größtmögliche Maximum der Filterkoeffizienten als auch für den daraus bestimmten maximalen Reflexionsfaktor als eine abfallende Exponentialfunktion dar. Bei wachsender Rückenlänge ist das Filter

demzufolge nur noch mit kleineren Reflexionsfaktoren realisierbar.

Diese Tatsache ist darauf zurückzuführen, daß der Anteil der Mehrfachreflexionen dritter Ordnung mit wachsender Breite des Filterrückens einen immer größeren Anteil an der Ausgangsleistung einnimmt. Die Mehrfachreflexionen dritter Ordnung drehen die Phase um 270° (drei Reflexionen à 90°). Da der jeweils aktive horizontale Eingang mit $-j$ belegt wird[1], gehen die besagten Mehrfachreflexionen dritter Ordnung negativ in die Berechnung der Ausgangsfeldstärke ein. Dadurch errechnet sich mit fortschreitender Grabennummer ein immer größerer Reflexionsfaktor für den jeweiligen Graben.
Anfangs kann dieses Phänomen bei $\pm 2\pi$ der *si*-Funktion für die Filterkoeffizienten mit Hilfe eines betragsmäßig größeren negativen Reflexionsfaktors zwischen $+\pi$ und $+2\pi$ (rechts der Mitte) ausgeglichen werden. Der minimale Relfexionsfaktor wächst allerdings betragsmäßig sehr schnell an (siehe Abb. 7.3, rote Kurve) — bis das Überhandnehmen der Mehrfachreflexionen schließlich nicht mehr ausgeglichen werden kann, und der Reflexionfaktor größer als Eins wird, was eine Realisierung unmöglich macht.

Vorteilhaft für die Vorherbestimmung des möglichen Maximalwertes der Filterkoeffizienten für eine bestimmte Rückenlänge ist der lineare Zusammenhang zwischen möglichem Maximum der Filterkoeffizienten und dem Maximum der aus diesen Koeffizienten errechneten effektiven Reflexionsfaktoren bei gleicher Wichtung der Koeffizienten (Fensterfunktion). Die Grafik in Abb. 7.6 zeigt diesen Zusammenhang für drei Koeffizienten β_{Kaiser} des Kaiserfensters und für ungewichtete Filterkoeffizienten bei verschiedenen Rückenlängen.

Ausblick

Die hier dargelegten Werte und Zusammenhänge gelten stellvertretend für sämtliche Rückenbreiten und Kaiserkoeffizienten β. Die Tatsache der Nichtlinearität dieser Zusammenhänge macht die Berechnungen sehr aufwendig.

Aufgrund der dargestellten Zusammenhänge und der verwendeten Berechnungs- und Hilfsprogramme können der Rechenaufwand verringert und die Ergebnisgewinnung

[1] um rein reelle Ergebnisse am Ausgang zu erhalten.

beschleunigt werden. Es ist möglich, das erlaubte Maximum der Filterkoeffizienten für eine bestimmte Rückenlänge und einen bestimmten Kaiserkoeffizienten der Fensterfunktion mehr oder weniger genau vorherzusagen, und die Berechnung anschließend in der Nähe dieses Wertes zu starten.

7.2. Ausgewählte Filter

Die Filter wurden für eine Freiraumwellenlänge von 1300 nm ausgelegt. Sie werden mit Halbleitertechnologie in Indium-Phosphit (InP) und Indium-Gallium-Arsenid-Phosphit (InGaAsP) realisiert. Dabei werden die Gräben durch InP erzeugt, die Wellenführung und die physikalisch notwendige Korrekturzone sind aus InGaAsP.

In der heutigen Lichtwellentechnik werden Laser nach dem DWDM Standard verwendet. Diese haben u.a. eine Bandbreite von 0,4 nm. Ziel ist es, ein Filter zu bauen, welches diese geringe Bandbreite filtern kann. Für den Weg dorthin wurden drei Rückenlängen bestimmt und ausgewählt. Die Rückenbreite ist dabei jeweils konstant 8 Zellen.

- **1000** Gräben
  Eine relativ kurze Rückenlänge, um die allgemeinen Eigenheiten eines solchen Filters bestimmen zu können und zu sehen, wie gut Modellbildung und Simulation wirklich sind.

- **2205** Gräben
  Mit diesem Wert wird laut den durchgeführten Berechnungen eine Filterbandbreite von einem Nanometer erreicht.

- **5531** Gräben
  Hiermit soll die gewünschte Bandbreite von 0,4 nm erreicht werden.

Es werden für jede dieser Rückenlängen mehrere Filterarten gebaut:

- Wichtung der Reflexionsfaktoren für den Streurücken mit einer Exponentialfunktion, um an dessen horizontalen Ausgängen ein betrags- und phasenmäßig konstantes E-Feld zu erzielen. Die Reflexionsfaktoren wurden hier bei Nutzung von $\pm 2\pi$ der *si*-Filterkoeffizienten-Funktion nach Methode 1 bestimmt (siehe 3.3).

- Wichtung des Streurückens mit einer e-Funktion und Bestimmung der Reflexionsfaktoren für den Sammelrücken (Methode 1) bei $\pm\pi$ der Filterkoeffizientenfunktion.

- Gleiche Wichtung der Reflexionsfaktoren für den Streu- und Sammelrücken (siehe 3.4) bei $\pm 2\pi$ der *si*-Funktion für die Filterkoeffizienten.

In Abbildung 7.8 sind die Ausgangsleistungen für eine Rückenlänge von 2205 Gräben logarithmisch aufgetragen. Dabei zeigt Teil a) die Ausgangsleistung nach Methode 1 (siehe 3.3) der Reflexionsfaktorenberechnung ($\pm 2\pi$ der *si*-Funktion, nur Sammelrücken), Teil b) die Ausgangsleistung bei Verwendung von $\pm\pi$ der Orstfunktion für die Filterkoeffizienten (ebenfalls Methode 1) und Teil c) die Ausgangsleistung für die nach Methode 2 (siehe 3.4) gewichteten Rücken.

Leistungsbetrachtung

Die Ausgangsleistung[2] bei dem nach Methode 1 gewichteten Rücken mit $\pm 2\pi$ der *si*-Funktion (a) beträgt ca. 20%. Die Ausgangsleistung bei $\pm\pi$ der Ortsfunktion für die Filterkoeffizienten (b) sowie bei dem nach Methode 2 (c) gewichteten Filter ist mehr als doppelt so groß (ca. 45%).

Kurvenform

Die Kurve in Teil a) der Abbildung hat die steilsten Flanken. Es herrscht eine Einsattelung bei der Mittenfrequenz (1300 nm) von ungefähr 0,1 dB.
Die Kurve in den Teil b) ist wesentlich runder, es gibt keine Einsattelung. Bei gleicher Rückenlänge ergibt sich eine ungefähr halb so große Bandbreite im Vergleich zu den Teilen a) und c).

[2] bei Mittenfrequenz

Die Kurve in Teil c) der Abbildung hat nicht ganz so steile Flanken, aber eine bessere Rechteckform als die in Teil b).

Zusammenfassung

Es bleibt abzuwarten, welche der drei zu bauenden Varianten bei der Messung die besten Ergebnisse liefern wird. Höchstwahrscheinlich wird ein Kompromiß zwischen Leistungsausbeute und Flankensteilheit zu schließen sein.

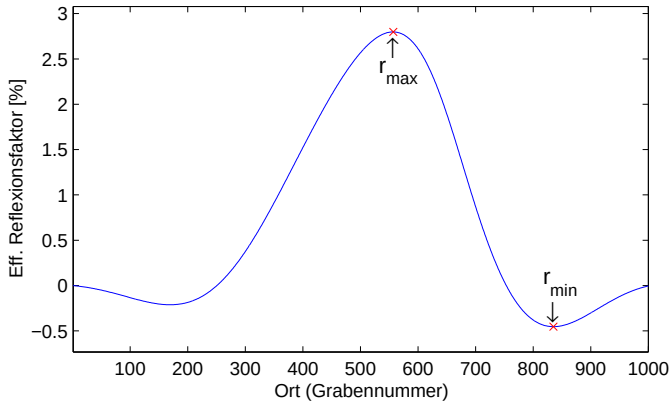


Abbildung 7.1.: Effektive Reflexionsfaktoren bei 1000 Gräben Rückenlänge

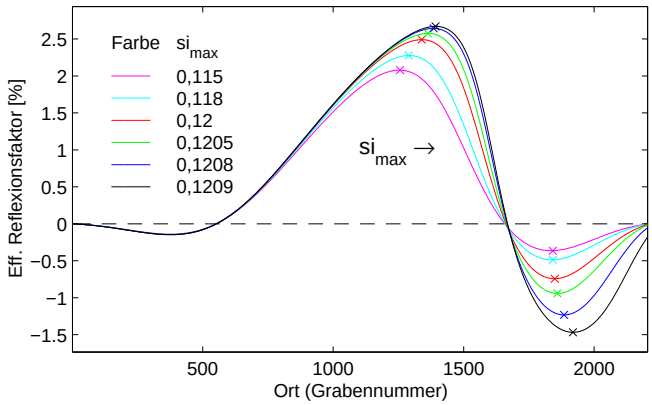
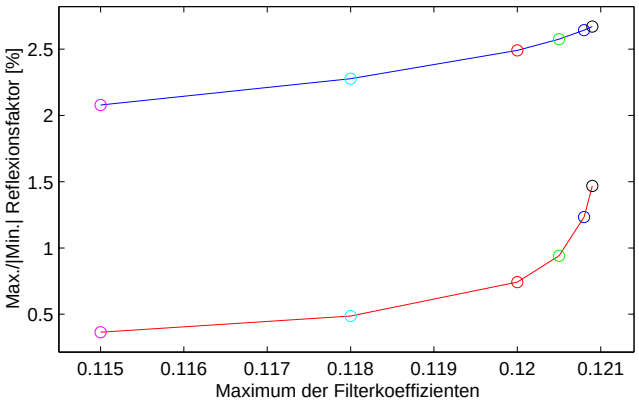


Abbildung 7.2.: Berechnete Reflexionsfaktoren für diverse si_{max}



blau: max. eff. Reflexionsfaktor rot: min. eff. Reflexionsfaktor

Abbildung 7.3.: Max./Min. Refelexionsfaktor über Maximum der Koeffizienten

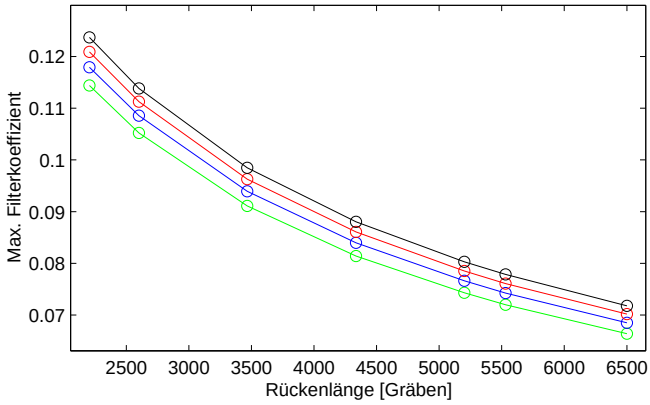


Abbildung 7.4.: Maximum der Filterkoeffizienten über der Rückenlänge

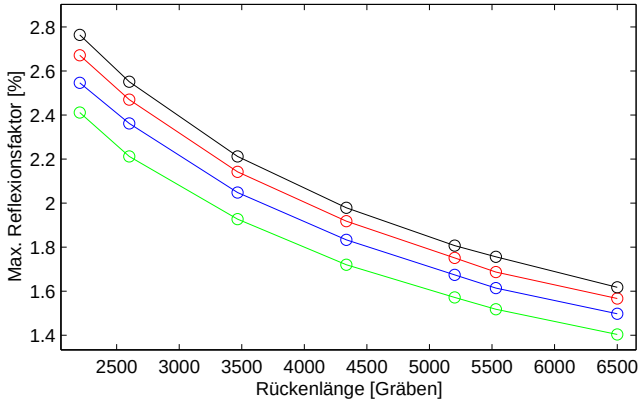


Abbildung 7.5.: Maximum der eff. Reflexionsfaktoren über der Rückenlänge

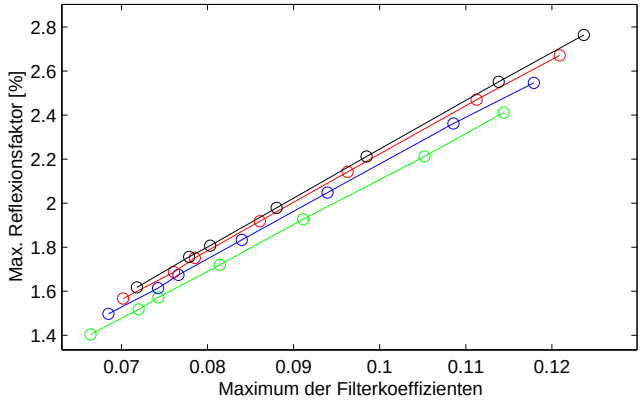


Abbildung 7.6.: Max. Reflexionsfaktoren über Maxima der Koeffizienten

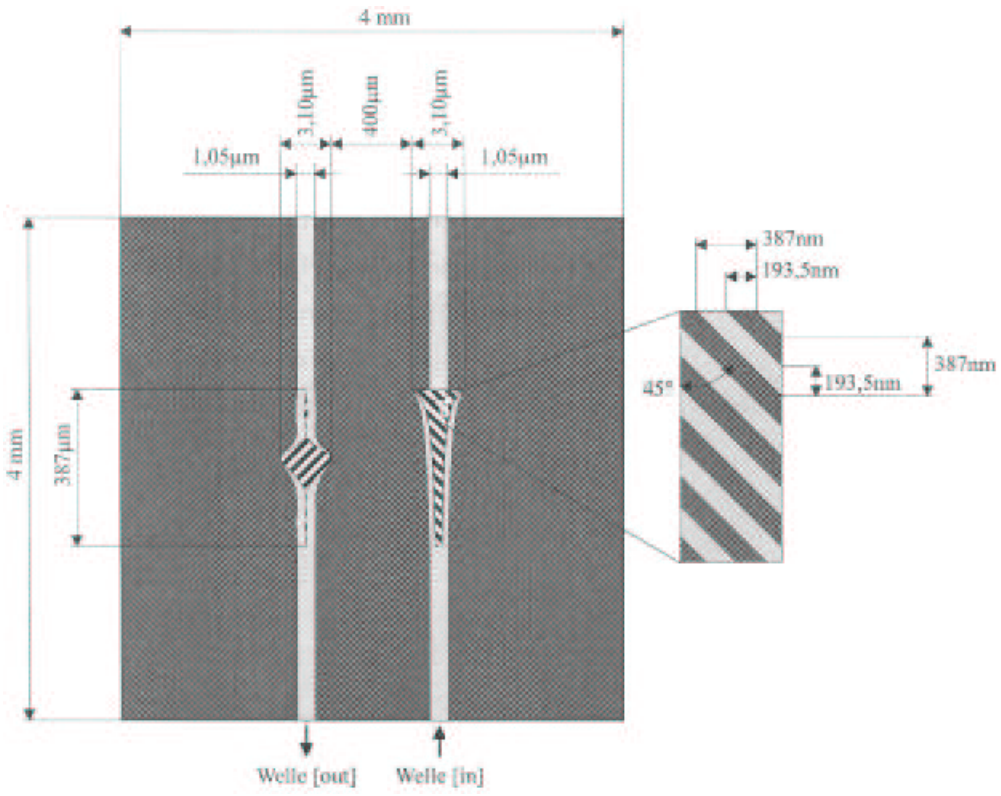
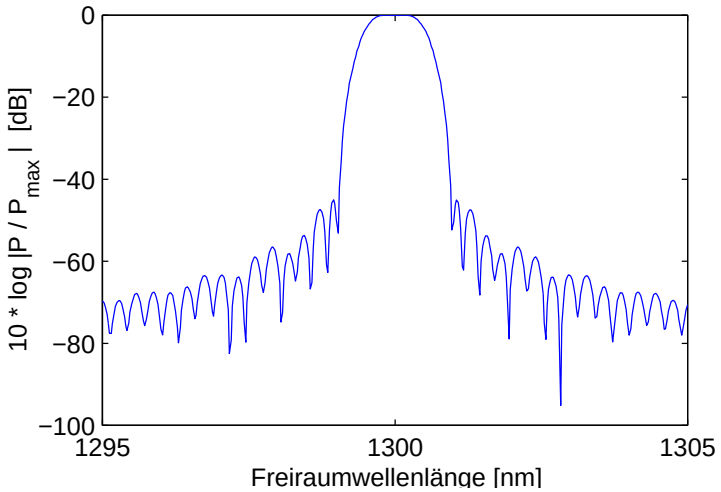
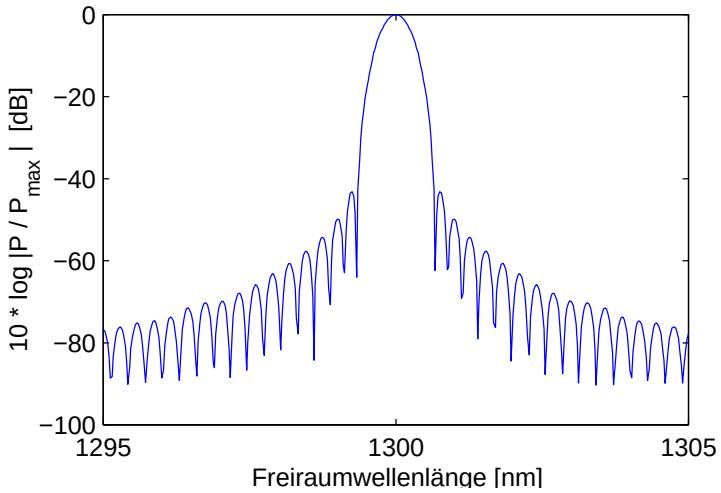


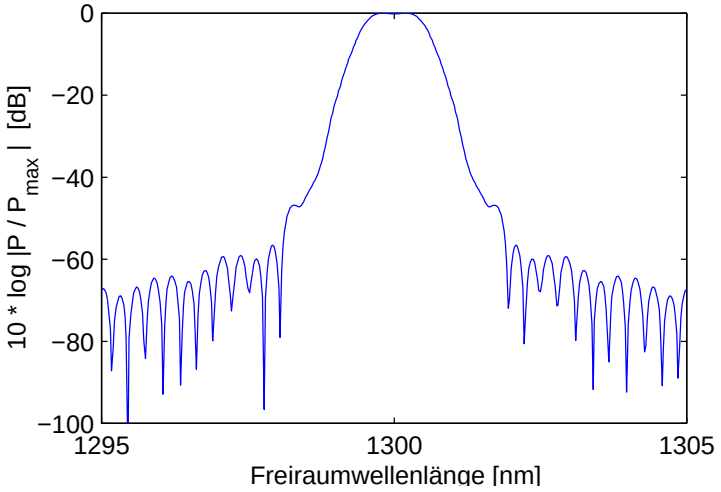
Abbildung 7.7.: Aufbau des integriert optischen Wellenfilters [1]



a)



b)



c)

a) Methode 1 mit $\pm\ 2\pi$ b) Methode 1 mit $\pm\ \pi$ c) Methode 2 mit $\pm\ 2\pi$

Abbildung 7.8.: Ausgangsleistungen bei einer Rückenlänge von 2205 Gräben

8. Berechnung der Zeichnungsdaten

8.1. Die Funktion zur Umrechnung der Reflexionsfaktoren

Die Wirkung der aus den Filterkoeffizienten errechneten effektiven Reflexionsfaktoren geht in die Größe des zugehörigen Grabens ein. Hier erfolgt der Übergang vom zweidimensionalen Modell in den Raum des Halbleitermaterials [6]. Dabei wird ebenfalls berücksichtigt, daß nur ein Teil der Welle den aktiven Filterbereich (die Gräben) durchläuft.
Für die Zeichnung der Masken in AutoCAD zur Herstellung des Filters müssen die Reflexionsfaktoren in Gitterbreiten und Weiten der Korrekturzone umgerechnet werden. Dafür ist die Funktion **r2gbwb.m** verantwortlich. Der Quellcode ist unter C.16 abgedruckt.
Zur problemlosen Ausführung sind zusätzlich folgende Skripten erforderlich:

- laden.m (Laden einer ASCII-Datei, siehe C.15)
- saven.m (Speichern einer ASCII-Datei, siehe C.19)

Der Aufruf dieser Funktion in der MatLab-Kommandozeile sieht folgendermaßen aus:

```
r2gbwb(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
```

Dabei haben die Parameter die gleichen Bedeutungen wie unter 3.3.3 und 4.1 beschrieben, allerdings müssen diese in Hochkommas gesetzt und durch Kommata getrennt werden.
Nach dem Start der Funktion werden noch einige Daten abgefragt, diese werden jetzt erläutert.

- *Liegen die Reflexionsfaktoren diskretisiert vor?*
  Falls die Reflexionsfaktoren mit **rdiskret** (siehe 8.2) teilweise zusammengefaßt wurden, muß an dieser Stelle eine `1` eingegeben werden. Ist dies nicht der Fall, reicht ein einfaches Drücken der Return-Taste aus.

- *GB diskretisieren?*
  Es wird nachgefragt, ob die Gitterbreiten diskretisiert werden sollen. Falls es gewünscht ist, daß die Gitterbreiten eine bestimmte Schwelle nicht unterschreiten, so muß an dieser Stelle eine `1` eingegeben werden. Falls das nicht erwünscht ist, drückt man einfach die Eingabetaste.

- *Schwelle*
  Wenn die Gitterbreiten diskretisiert werden sollen, so hat man hier die Möglichkeit, den Schwellwert, welcher nicht unterschritten werden darf, einzugeben. Die Einheit ist Nanometer. Der Standardwert beläuft sich auf 387 nm, das ist die Materialwellenlänge des Lichtes eines 1300 nm Lasers bei der Nutzung von Indium-Phosphit und Indium-Gallium-Arsenid-Phosphit als Halbleitermaterial.

Nach der Berechnung aller Werte werden diese in einer MatLab-Figur dargestellt. Diese enthält die folgenden Grafiken:

- die zugrundeliegenden Reflexionsfaktoren für den Streu- (falls verwendet) und Sammelrücken,
- den daraus berechneten Confinementfaktor Γ,
- die bestimmten Gitterbreiten,

- die errechneten Weiten der Korrekturzone.

Die Speicherung der Ergebnisse erfolgt automatisch entsprechend der in A beschriebenen Datenstruktur.

8.2. Die Funktion zur Diskretisierung der Reflexionsfaktoren

Falls es gewünscht ist, daß die Gitterbreiten einen bestimmten Wert nicht unterschreiten, so besteht mit Hilfe der Funktion **rdiskret.m** (siehe C.17) die Möglichkeit, mehrere zu kleine Reflexionsfaktoren zu einem genügend großen zusammenzufassen. Zur problemlosen Ausführung sind außerdem folgende MatLab-Skripten erforderlich:

- laden.m (Laden einer ASCII-Datei, siehe C.15)
- diskret.m (Eigentliche Diskretisierung, siehe C.6)
- saven.m (Speichern einer ASCII-Datei, siehe C.19)

Der Aufruf dieser Funktion in der MatLab-Kommandozeile sieht folgendermaßen aus:

```
rdiskret(rueckenbreite, rstart, anz_pi,
         kaiser_beta, si_max, lq);
```

Dabei haben die Parameter die gleichen Bedeutungen wie unter 3.3.3 und 4.1 beschrieben, allerdings müssen diese in Hochkommas gesetzt und durch Kommata getrennt werden.
Die Speicherung der Ergebnisse erfolgt automatisch entsprechend der in A beschriebenen Datenstruktur.

8.3. Die Funktion zur Rückrechnung diskretisierter Gitterbreiten

Falls bei der Erstellung der Zeichnungsdaten (siehe 8.1) die Gitterbreiten diskretisiert wurden, können diese mit Hilfe der Funktion **gb2r.m** (siehe C.12) in die entsprechenden Reflexionsfaktoren zurückgerechnet werden. Anhand dieser Reflexionsfaktoren kann anschließend die Wirkung einer Diskretisierung im Gitterbreitenbereich auf die Filtereigenschaften mit Hilfe des Programmes **aus.exe** (siehe Kapitel 4) bestimmt werden.
Zur problemlosen Ausführung von **gb2r** sind folgende MatLab-Skripten erforderlich:

- laden.m (Laden einer ASCII-Datei, siehe C.15)
- saven.m (Speichern einer ASCII-Datei, siehe C.19)

Der Aufruf dieser Funktion in der MatLab-Kommandozeile sieht folgendermaßen aus:

```
gb2r(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
```

Dabei haben die Parameter die gleichen Bedeutungen wie unter 3.3.3 und 4.1 beschrieben, allerdings müssen diese in Hochkommas gesetzt und durch Kommata getrennt werden.
Die Speicherung der Ergebnisse erfolgt automatisch entsprechend der in A beschriebenen Datenstruktur.

9. Zeichnen mit AutoCAD

9.1. Das Programm für die Vorbereitung der gesamten Struktur

Nachdem die Zeichnungsdaten für sämtliche Filter, welche gemeinsam in eine Struktur gezeichnet werden sollen, mit Hilfe der MatLab-Funktion **r2gbwb** (siehe 8.1) berechnet und gespeichert wurden, geht es an die Vorbereitung der zu zeichnenden Struktur. Für jede Filterstruktur wird genau eine Zeichnung angefertigt. Zu diesem Zweck gibt es das MatLab-Programm **acad_v.m**. Den Quellcode dieses Programmes zeigt C.1.
Zur problemlosen Ausführung sind zusätzlich folgende MatLab-Skripten erforderlich:

- laden.m (Laden einer ASCII-Datei, siehe C.15)
- saven.m (Speichern einer ASCII-Datei, siehe C.19)

Dieses Programm wird durch Eingabe von

```
acad_v
```

in der MatLab-Kommandozeile gestartet. Dabei sind einige Eingaben zu machen, welche im folgenden erläutert werden.

- *Anzahl der Filter pro Struktur*

  Hier wird angegeben, wie viele Filter zu einer Struktur (Zeichnung) zusammengefaßt werden sollen.

- *AutoCAD-Speicherverzeichnis*

  An dieser Stelle gibt man das Verzeichnis, in dem die zukünftigen AutoCAD Zeichnungen gespeichert werden sollen, an. Es wird relativ zu **m/daten/acad/** und in Hochkommas angegeben. In diesem Verzeichnis werden auch die Skripten für die Zeichnung gespeichert (siehe 9.2).

- *Verzeichnis für die Filterdaten*

  Für eine bestimmte Zeichnungsgröße können mehrere Strukturen mit unterschiedlichen Filtern erstellt werden (maximal fünf). Hier wird nun das Verzeichnis für die Zeichnungsdaten relativ zum AutoCAD-Speicherverzeichnis in Hochkommas angegeben. Jede Struktur benötigt ein eigenes Filterdatenverzeichnis.

Im Anschluß an diese allgemeinen Eingaben muß jedes einzelne Filter, welches in diese Struktur gezeichnet werden soll, angegeben werden. Hier werden die gleichen Eingaben verlangt wie beim C++ Programm **aus.exe** (siehe 3.3.3 und 4.1). Diese Angaben müssen hier in Hochkommas eingeschlossen werden.
Nach Abschluß aller Eingaben wird in das Filterdateienverzeichnis eine Datei namens **info.txt** gespeichert, welche Information über die verwendeten Filter enthält.

9.2. Das Programm zur Erstellung der Skripten

Eine Zeichnung in AutoCAD ist nicht sehr schnell zu erstellen, vor allem die für ein optisches Wellenfilter mit teilweise mehr als 1000 Gräben Länge. Um dem Anwender die Arbeit so weit wie möglich zu erleichtern, wurde die komplette Erstellung der Zeichnungen durch die Nutzung von Skripten automatisiert.

Aufgrund der vorbereiteten Zeichnungsdaten (siehe 9.1) werden mit Hilfe des Programmes **acad.exe** alle nötigen Skripten für AutoCAD erstellt. Der Quellcode ist in der Datei **acad.cpp** zu finden (siehe D.1).

9.2.1. Starten des Programmes

Dieses Programm wird von der MS-DOS Eingabeaufforderung aus gestartet. Zum Starten wechselt man in das Verzeichnis, in welchem sich die Datei **acad.exe** befindet. Falls man allerdings genau dieses Verzeichnis dem `path` hinzugefügt hat, ist es uninteressant, in welchem Verzeichnis sich der Anwender gerade befindet. Der Aufruf lautet wie folgt:

```
acad <datenverzeichnis>
```

Das *datenverzeichnis* ist genau das, welches bei der Vorbereitung in MatLab (unter 9.1 beschrieben) als *Verzeichnis für die Filterdaten* angegeben wurde.

Nach Drücken der Eingabetaste werden am oberen Bildschirmrand das Verzeichnis für die Filterdaten (*Daten-Verzeichnis*) und das AutoCAD-Speicherverzeichnis (*Skript-Verzeichnis*) angezeigt. Ein Auswahlmenü wird dargestellt, wo man zwischen

1 . . . Skripte für Grund- (Hilfsumrandung) und Pruefstruktur und
2 . . . Skripte für die einzelnen Filter sowie
0 . . . Ende

wählen kann. Durch Eingabe der entsprechenden Ziffer und anschließendes Drücken der Return-Taste wird ein Menüpunkt ausgewählt. Die Punkte 1 und 2 werden im folgenden genauer beschrieben.

9.2.2. Grund- und Prüfstruktur

Mit Hilfe dieses Menüpunktes werden die Skripten zum Zeichnen der jeweiligen Grundstruktur und der entsprechenden Prüfstruktur in AutoCAD erstellt.

Die Anzahl der Filter pro Struktur (Zeichnung) steht bereits fest, sie wurde bei der Vorbereitung unter 9.1 angegeben. Zusätzlich sind noch einige Angaben erforderlich, welche gleich erläutert werden. Bei jeder Angabe steht der Standardwert in eckigen Klammern, dieser wird durch Eingabe von `0` und anschließendes Drücken der Eingabetaste übernommen.

- *Breite eines Filters*
  Mit diesem Wert ist die Breite für ein einziges Filter in dieser Struktur gemeint. Die Angabe erfolgt in Mikrometer.
- *Höhe der Struktur*
  Hier wird die Höhe, das heißt die Länge eines Filters, angegeben. Auch diese Angabe erfolgt in Mikrometer.
- *Breite eines LWL*
  Damit ist die Breite eines Lichtwellenleiters inklusive Mantel gemeint, die Einheit ist Mikrometer.
- *Breite der Seele eines LWL*
  Mit dieser Breite ist der Durchmesser des Lichtwellenleiterkerns gemeint, die Einheit ist auch hier Mikrometer.
- *Abstand der LWL für Ein- und Auskopplung*
  Dieser Wert gibt den Abstand zwischen Streu- und Sammelrücken von Mitte zu Mitte an. Die Eingabe erfolgt in Mikrometer.
- *Wellenlänge im Material*
  Damit ist die Lichtwellenlänge des Lasers im Halbleitermaterial, in welchem die Filter realisiert werden, gemeint. Diese wird in Nanometer angegeben.

Im Anschluß an diese allgemeinen Angaben werden die Anzahl der Bragg-Gitter und die gewünschte Breite dieser Gitter für jedes Filter der Prüfstruktur einzeln abgefragt. Es wird jeweils zuerst nach dem linken und dann nach dem rechten Lichtwellenleiter gefragt. Soll kein Gitter gezeichnet werden, gibt man `0` als Gitteranzahl an.

Nun werden alle erforderlichen Skripten erstellt, diese werden im *AutoCAD-Speicherverzeichnis* (siehe 9.1) abgelegt und haben die folgenden Namen.

- **grund.scr**
  Zeichnung der Grundstruktur.
- **pruef.scr**
  Zeichnen der Prüfstruktur.

9.2.3. Einzelne Filter

Zu Beginn wird hier die *Nummer der Struktur* angegeben. Das ist eine Ziffer zwischen Eins und Fünf. Im Anschluß daran werden die Filterdaten, welche unter 9.1 vorbereitet wurden, geladen. Das Programm berechnet die Startkoordinate in y-Richtung und zeigt diese an. Diese Koordinate wird so berechnet, daß sich Sende- und Empfangsrücken genau in der Mitte der Struktur befinden. Dieser Wert wird durch Eingabe von `0` übernommen. Falls ein anderer Startpunkt gewünscht ist, kann dieser stattdessen eingegeben werden. Die Einheit Mikrometer ist zu beachten.
Im Normalfall wird die Führung des Lichtes bis zum Ende der Struktur fortgesetzt. Wenn der Streurücken eines Filters allerdings durch einen Wellensumpf abgeschlossen werden soll, so ist bei der entsprechenden Frage eine `1` einzugeben.
Nun wird das komplette Skript zur Zeichnung eines Filters erstellt und abgespeichert. Nach Erstellung aller Filterskripten wird das Skript zur Kennzeichnung der Struktur erstellt.
Die Skripten für die Struktur Nummer Eins, welche zum Beispiel aus drei Filtern besteht, haben die folgende Dateinamen.

- **flt1-1.scr**
  Zeichnen der Gräben des 1. Filters.
- **fhr1-1.scr**
  Zeichnen der Wellenführung und der Korrekturzone für das 1. Filter.

- **flt1-2.scr**

  Zeichnen der Gräben des 2. Filters.

- **fhr1-2.scr**

  Zeichnen der Wellenführung und der Korrekturzone für das 2. Filter.

- **flt1-3.scr**

  Zeichnen der Gräben des 3. Filters.

- **fhr1-3.scr**

  Zeichnen der Wellenführung und der Korrekturzone für das 3. Filter.

- **nr1.scr**

  Skript zur Kennzeichnung der Struktur.

Diese Skripten werden alle im *AutoCAD-Speicherverzeichnis* (siehe 9.1) gespeichert, so daß sie von AutoCAD aus einfach zu finden sind.

9.3. Aufruf der Skripten in AutoCAD

Die Grundstruktur ist Voraussetzung für jede Zeichnung. Dieses Skript (**grund.scr**, Erstellung: siehe 9.2.2) muß immer zuerst ausgeführt werden, nachdem in AutoCAD *Neue Zeichnung* aufgerufen wurde.

Zum Zeichnen der kompletten Prüfstruktur ist in AutoCAD im Anschluß daran das Skript zum Zeichnen dieser aufzurufen (**pruef.scr**, Erstellung: siehe 9.2.2).

Zum Zeichnen der Masken für die einzelnen Filter einer Struktur sind nach Ausführung von **grund.scr** alle unter 9.2.3 erstellten Skripten in beliebiger Reihenfolge aufzurufen.

10. Zusammenfassung

Die Forschung zur Herstellung eines integriert optischen Wellenfilters auf Halbleiterbasis am Institut für Nachrichtentechnik der Universität der Bundeswehr in München steht noch am Anfang. Theoretische Berechnungen wurden in nicht gerade geringem Umfang durchgeführt, aber ein auf der Grundlage dieser Theorien gebautes Filter steht momentan noch nicht zur Verfügung.
In dieser Arbeit wurde die Theorie zur Berechnung eines solchen optischen Wellenfilters kurz dargelegt. Auf Grundlage der z.T. erweiterten Modellbildung und mit Hilfe der in programmierbare Form gebrachten Gleichungen sowie gewonnener Erkenntnisse wurden mehrere Filter für eine Lichtwellenlänge von 1300 nm bei Variation verschiedener Parameter dimensioniert. Sämtliche Ergebnisse sind im Anhang abgedruckt, die vielversprechendsten Filterkonfigurationen wurden zum Bau ausgewählt. Zur Herstellung der Filter in Halbleitertechnologie sind sogenannte Masken erforderlich, diese werden mit einem CAD Programm gezeichnet. Die Komplexität des Filteraufbaus macht ein Zeichnen von Hand quasi unmöglich – so wurde diese Aufgabe mit Hilfe der Erstellung von Skripten zur Abarbeitung in AutoCAD vollständig automatisiert. Dem Bau der ersten Filter steht somit nichts mehr im Wege.

Aufgrund des hohen Rechenaufwands ist ein Programmpaket entstanden, welches es ermöglicht, die notwendigen Berechnungen auf mehrere Personalcomputer in einem Netzwerk zu verteilen – das verteilte Rechnen.

Verwendete Abkürzungen

Abb.	Abbildung
AutoCAD	das verwendete CAD-Programm
bzw.	beziehungsweise
C, C++	die Programmiersprachen C bzw. C++
ca.	cirka
DFT	Diskrete Fouriertransformation
DWDM	Dense Wavelength Division Multiplexing
d.h.	das heißt
eff.	effektiv
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JobID	Identifikationsnummer einer verteilten Berechnung
LAN	Local Area Network
LWL	Lichtwellenleiter
MS-DOS	Microsoft Disk Operating System
MatLab	die genutzte Berechnungs- und Simulationssoftware
max.	maximal
min.	minimal
μm	Mikrometer
nm	Nanometer
o.g.	oben genannt

PC	Personalcomputer
RMI	Remote Methode Invocation (Java)
TCP/IP	Transport Control Protocol / Internet Protocol
u.a.	unter anderem
usw.	und so weiter
vgl.	vergleiche
z.B.	zum Beispiel
z.T.	zum Teil

Verwendete Symbole

A	Vektor aller Eingangswellengrößen einer Zelle
A_x	eine Eingangswellengröße einer Zelle
a_g	der Filterkoeffizient für den g-ten Graben
aus_p	bekannter Anteil der Ausgangsfeldstärke in Zeile p
`anz_pi`	Anzahl π, nach denen die si-Funktion abgebrochen wurde (als Übergabeparameter)
B	Vektor aller Ausgangswellengrößen einer Zelle
B_x	eine Ausgangswellengröße einer Zelle
β, β_{Kaiser}	Koeffizient β des genutzten Kaiserfensters
E_p	gesamte Ausgangsfeldstärke in der p-ten Zeile
$fakt_p$	Produkt der zu berücksichtigenden Transmissionen in einer Spalte
G	Anzahl der Gräben eines Rückens (Rückenlänge)
g	laufende Grabennummer
j	die komplexe Zahl j, mit $j^2 = -1$
`kaiser_beta`	Koeffizient β des genutzen Kaiserfensters (als Übergabeparameter)
`lq`	Übergabeparameter; `l` bedeutet lineare Betrachtung, `q` bedeutet quadratische Betrachtung der Ausgänge
Λ_m	Wellenlänge des Lichts im Halbleitermaterial bei der Mittenfrequenz des Filters

M	Anzahl der Spalten der Gitterstruktur
m	laufende Spaltennummer
P	Rückenbreite (Anzahl der Zeilen)
p	laufende Zeilennummer
r	Reflexionsfaktor einer Gitterzelle
r_g	effektiver Reflexionsfaktor des g-ten Grabens
r_{max}	maximaler effektiver Reflexionsfaktor
r_{min}	minimaler effektiver Reflexionsfaktor
`rstart`	kleinster Reflexionsfaktor des exponentiell gewichteten Senderückens (als Übergabeparameter)
`rueckenbreite`	Rückenlänge G (als Übergabeparameter)
$\underline{S}$	Streumatrix einer Gitterzelle
si_{max}	Maximalwert der Filterkoeffizienten
`si_max`	Maximum der si-Funktion für die Filterkoeffizienten (als Übergabeparameter)
`si_original`	Maximalwert bereits gespeicherter Filterkoeffizienten (als Übergabeparameter)
t	Transmissionfaktor einer Gitterzelle
t_i	Transmissionfaktor des i-ten Grabens

Literaturverzeichnis

[1] Arbeitsunterlagen Dipl.-Ing. Peter Crassen Hruschka,
Universität der Bundeswehr München, Fakultät für Elektrotechnik,
Institut für Nachrichtentechnik, 1996-2000

[2] Oberdan W. Otto, Multiple Reflections in Acoustic Surface Wave
Reflective Arrays, IEEE Transactions on Sonics and Ultrasonics,
Vol. SU-22, No. 4, P. 251-257, 1975

[3] Eugene Hecht, Optik (Dt. Übersetzung von F. Siemsen),
2. durchges. Auflage, R. Oldenbourg Verlag, 1999

[4] Prof. Dr.-Ing. Ulrich Appel, Vorlesungsskript Digitale Signalverarbeitung,
Universität der Bundeswehr München, Fakultät für Elektrotechnik,
Institut für Mathematik und Datenverarbeitung, 1998

[5] Alan V. Oppenheim und Ronald W. Schafer,
Zeitdiskrete Signalverarbeitung, R. Oldenbourg Verlag, 1999

[6] Arbeitsunterlagen Dipl.-Ing. Seongjae Boo,
Universität der Bundeswehr München, Fakultät für Elektrotechnik,
Institut für Nachrichtentechnik, 1996-2000

[7] Prof. Dr.-Ing. Udo Barabas, Optische Signalübertragung,
R. Oldenbourg Verlag, 1993

[8] Katsuyuki Utaka, Ken-Ichi Kobayashi and Yasuharu Suematsu,
Lasing Characteristics of 1.5-1.6 μm GaInAsP/InP Integrated
Twin-Guide Lasers with First-Order Distributed Bragg Reflectors,
IEEE Journal of Quantum Electronics, Vol. QE-17, No. 5, P. 651-658, 1981

[9] Charles H. Henry, L. F. Johnson, Ralph A. Logan and D. P. Clarke,
Determination of the Refractive Index of InGaAsP
Epitaxial Layers by Mode Line Luminescence Spectroscopy,
IEEE Journal of Quantum Electronics, Vol. QE-21, No. 12, P. 1887-1892, 1985

[10] Ulrich Breymann, C++ : eine Einführung, 4. Auflage,
Carl Hanser Verlag München Wien, 1997

[11] Java 2 Platform, Standard Edition, v 1.3, API Specification,
Sun Microsystems, http://java.sun.com/, 1993-2000

A. Aufbau der Datenstruktur für die Speicherung der Filterdaten

A.1. Das Rootverzeichnis

Im Hauptverzeichnis der Datenstruktur sind die folgenden Ordner zu finden:

- ***X*pi**

  Das sind die Verzeichnis für sämtliche Filterdaten mit Nutzung von X Perioden der si-Funktion für die Filterkoeffizienten. Information über die Unterverzeichnisse eines solchen Ordners ist unter A.2 zu finden.

- **acad**

  In diesem Verzeichnis werden Unterverzeichnisse für die AutoCAD Zeichnungen angelegt. In diesen befinden sich wiederum Verzeichnisse für die jeweiligen zugrundeliegenden Filterdaten.

- **fxsfind**

  Hier werden sämtliche Daten der mit dem gleichnamigen MatLab-Programm bestimmten Filterkoeffizienten abgespeichert. Außerdem sind hier die jeweils zugehörigen MatLab-Figuren im Encapsulated-Postscript-Format zu finden.

- **fxswin**

  Die Daten der Filterkoeffizienten, welche mit einer Fensterfunktion gewichtet

wurden, werden hier gespeichert. Die bei der Bestimmung durch das gleichnamige MatLab-Programm gezeichneten MatLab-Grafiken werden ebenfalls hier abgelegt.

A.2. Ein π-Verzeichnis

In einem solchen Filterdatenverzeichnis gibt es jeweils einen Ordner für eine bestimmte Rückenlänge, z.B. **1000**.

Jedes Längenverzeichnis enthält ein Unterverzeichnis namens **streu** für die Daten des mit einer e-Funktion gewichteten Streurückens sowie Unterverzeichnisse für jedes genutzte Kaiserfenster, z.B. **34** für einen Kaiserkoeffizienten von $\beta = 3,4$.

Der oben genannte Ordner **streu** unterhalb des Längenverzeichnisses hat die im folgenden aufgeführten Unterverzeichnisse:

- **gb**
  Aus den jeweiligen Reflexionsfakoren berechnete Gitterbreiten
  für die AutoCAD-Zeichnung.

- **r**
  Berechnete Reflexionsfaktoren für ein bestimmtes Maximum
  der Filterkoeffizienten.

- **wb**
  Berechnete Weiten der Korrekturzone für die jeweiligen Reflexionsfakoren (erforderlich für die AutoCAD-Zeichnung).

In einem „Kaiserfenster“-Verzeichnis unterhalb des entsprechenden Längenverzeichnisses befinden sich die folgenden Verzeichnisse und Unterverzeichnisse:

- **a**
  Der Ordner für die Speicherung der Filterkoeffizienten
  mit einem bestimmten Maximum.

- **e**

  Ergebnisse der Filter-Ausgangsberechnung für die mit *ainr* bestimmten Reflexionsfaktoren (horizontale Ausgänge des Streurückens).

- **e2**

  Ergebnisse der Filter-Ausgangsberechnung für die mit *ainr2* bestimmten Reflexionsfaktoren (horizontale Ausgänge des Streurückens).

- **p**

  Ergebnisse der Filter-Ausgangsberechnung für die mit *ainr* bestimmten Reflexionsfaktoren (eigentlicher Filterausgang).

- **p2**

  Ergebnisse der Filter-Ausgangsberechnung für die mit *ainr2* bestimmten Reflexionsfaktoren (eigentlicher Filterausgang).

- **sammel**

  - **gb**

    Gitterbreiten des Sammelrückens für *ainr*-Reflexionsfaktoren.

  - **gb2**

    Gitterbreiten für die von *ainr2* bestimmten Reflexionsfaktoren.

  - **r**

    Von *ainr* bestimmte Reflexionsfaktoren.

  - **r2**

    Von *ainr2* berechnete Reflexionsfaktoren.

  - **wb**

    Weiten der Korrekturzone für die von *ainr* berechneten Reflexiosfaktoren.

  - **wb2**

    Weiten der Korrekturzone für *ainr2*-Reflexionsfaktoren.

- **streu**

  - **gb2**

    Gitterbreiten des Streurückens für die von *ainr2*
    berechneten Reflexionsfaktoren.

  - **r2**

    Die Reflexionsfaktoren für den Streurücken bei Verwendung von *ainr2*.

  - **wb2**

    Entsprechende Weiten der Korrekturzone bei Nutzung von *ainr2*.

- **trans**

  Zum Vergleich berechnete Ausgänge eines Transversalfilters mit den Original-Filterkoeffizienten.

B. Errechnete Werte

B.1. Ausgewählte Ergebnisse $(2 \cdot 2\pi)$

B.1.1. Ungewichtete Filterkoeffizienten

länge[1]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]
2205	0,1144	2,411	1363	-2,055	2060
2601	0,1052	2,212	1599	-1,718	2373
3467	0,0911	1,927	2140	-1,834	3388
4435	0,081425	1,720	2668	-1,425	4009
5201	0,0743	1,572	3207	-1,339	4833
5531	0,072	1,518	3409	-1,169	5017
6501	0,066407	1,404	4008	-1,143	5977

B.1.2. Gewichtete Filterkoeffizienten

kaiser[6]	länge[1]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]
2,0	2205	0,1179	2,546	1376	-1,575	1952
	2601	0,10855	2,362	1632	-1,702	2387
	3467	0,09393	2,048	2171	-1,489	3187
	4335	0,08398	1,833	2716	-1,343	4003
	5201	0,07662	1,674	3255	-1,228	4783
	5531	0,07425	1,614	3458	-1,018	4889
	6501	0,06849	1,498	4072	-1,076	5934
3,4	2205	0,1209	2,671	1392	-1,468	1919
	2601	0,11126	2,470	1647	-1,498	2312
	3467	0,09627	2,142	2195	-1,293	3068
	4335	0,08607	1,918	2752	-1,168	3852
	5201	0,07852	1,751	3294	-1,040	4571
	5531	0,0761	1,687	3486	-0,866	4745
	6501	0,07019	1,567	4114	-0,924	5703
5,0	2205	0,1237	2,764	1404	-1,385	1897
	2601	0,11381	2,551	1656	-1,352	2261
	3467	0,09847	2,212	2206	-1,162	2999
	4335	0,08803	1,979	2765	-1,013	3726
	5201	0,08031	1,807	3309	-0,910	4453
	5531	0,077875	1,756	3531	-0,952	4814
	6501	0,07179	1,618	4148	-0,829	5577

B.2. Alle Ergebnisse $(2 \cdot 2\pi)$

B.2.1. ainr für Sammelrücken

Lineare Betrachtung - Bandbreitenübersicht

länge[1]	angegeben[7]	kaiser[6]	3dB (qs[8])	3dB (sq[9])	6dB (qs[8])	6dB (sq[9])
501	5,20 nm	2,0	4,38 nm	4,40 nm	5,12 nm	5,12 nm
		3,4	4,32 nm	4,32 nm	5,18 nm	5,23 nm
		5,0	4,15 nm	4,18 nm	5,20 nm	5,23 nm
521	5,00 nm	3,4	3,10 nm	3,15 nm	3,98 nm	4,02 nm
1001	2,60 nm	-	2,15 nm	2,20 nm	2,45 nm	2,52 nm
1000	-,- nm	2,0	2,16 nm	2,20 nm	2,50 nm	2,56 nm
		3,4	2,16 nm	2,20 nm	2,56 nm	2,60 nm
		5,0	2,06 nm	2,10 nm	2,56 nm	2,60 nm
2205	1,18 nm	-	0,95 nm	1,00 nm	1,05 nm	1,15 nm
		2,0	0,95 nm	1,00 nm	1,05 nm	1,20 nm
		3,4	0,95 nm	1,00 nm	1,05 nm	1,20 nm
		5,0	0,90 nm	0,95 nm	1,05 nm	1,20 nm
2601	1,00 nm	-	0,80 nm	0,85 nm	0,90 nm	1,00 nm
		2,0	0,80 nm	0,85 nm	0,90 nm	1,00 nm
		3,4	0,80 nm	0,85 nm	0,90 nm	1,00 nm
		5,0	0,80 nm	0,80 nm	0,90 nm	1,00 nm
3467	0,75 nm	-	0,60 nm	0,65 nm	0,65 nm	0,75 nm
		2,0	0,60 nm	0,65 nm	0,65 nm	0,75 nm
		3,4	0,60 nm	0,65 nm	0,65 nm	0,75 nm
		5,0	0,60 nm	0,60 nm	0,65 nm	0,75 nm

länge[1]	angegeben[7]	kaiser[6]	3dB (qs[8])	3dB (sq[9])	6dB (qs[8])	6dB (sq[9])
4335	0,60 nm	-	0,50 nm	0,55 nm	0,55 nm	0,60 nm
		2,0	0,50 nm	0,55 nm	0,55 nm	0,60 nm
		3,4	0,50 nm	0,50 nm	0,55 nm	0,60 nm
		5,0	0,50 nm	0,50 nm	0,55 nm	0,60 nm
5201	0,50 nm	-	0,40 nm	0,45 nm	0,45 nm	0,50 nm
		2,0	0,40 nm	0,45 nm	0,45 nm	0,50 nm
		3,4	0,40 nm	0,45 nm	0,45 nm	0,50 nm
		5,0	0,40 nm	0,40 nm	0,45 nm	0,50 nm
5531	0,47 nm	-	0,40 nm	0,40 nm	0,45 nm	0,50 nm
		2,0	0,40 nm	0,40 nm	0,45 nm	0,50 nm
		3,4	0,40 nm	0,40 nm	0,45 nm	0,50 nm
		5,0	0,40 nm	0,40 nm	0,45 nm	0,50 nm
6501	0,40 nm	-	0,35 nm	0,35 nm	0,35 nm	0,40 nm
		2,0	0,35 nm	0,35 nm	0,35 nm	0,40 nm
		3,4	0,35 nm	0,35 nm	0,38 nm	0,40 nm
		5,0	0,35 nm	0,35 nm	0,38 nm	0,40 nm

Lineare Betrachtung - Ergebnisse

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
501	2,0	0,17	2,397	261	-0,389	423	x	
		0,18	2,588	263	-0,432	423	x	
		0,19	2,794	265	-0,481	423	x	x
501	3,4	0,17	2,379	261				
		0,18	2,565	262				
		0,19	2,764	264				
		0,192	2,805	264	-0,331	416	x	x
		0,195	2,869	265				
501	5,0	0,17	2,364	259	-0,177	411	x	
		0,18	2,546	261	-0,194	411	x	
		0,19	2,738	262	-0,215	411	x	
		0,191	2,758	263	-0,217	411	x	
		0,192	2,778	263	-0,219	411	x	
		0,193	2,799	263	-0,221	411	x	x
		0,195	2,840	263	-0,226	411	x	
521	3,4	0,0062	0,078	262	-0,005	466	x	
		0,01	0,125	262	-0,008	466	x	
		0,1	1,305	265	-0,083	466	x	
		0,15	2,091	272	-0,145	466	x	
		0,17	2,471	277	-0,181	466	x	
		0,18	2,687	280	-0,205	466	x	
		0,185	2,804	281	-0,219	468	x	x
		0,19	2,929	284	-0,235	468	x	
		0,2	3,207	289	-0,276	468	x	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
1001	-	0,159	2,763	565	-0,917	861	x	
		0,1595	2,786	567	-0,935	861	x	
		0,15975	2,797	567	-0,945	861	x	
		0,159812	2,800	567	-0.947	861	x	x
		0,159875	2,803	567	-0,950	861	x	
		0,16	2,808	567	-0,954	861	x	
1000	2,0	0,162	2,790	562	-0,680	849	x	x
1000	3,4	0,1645	2,800	557	-0,450	835	x	x
1000	5,0	0,1667	2,800	551	-0,300	823	x	x
2205	-	0,1	1,610	1204	-0,455	1890	x	x
		0,11	2,020	1269	-0,763	1890	x	
		0,113	2,243	1318	-1,118	1917	x	
		0,114	2,354	1343	-1,495	1953	x	
		0,1144	2,411	1363	-2,055	2060	x	x
		0,1145	¿100	2145				
2205	2,0	0,11	1,917	1241	-0,443	1862	x	
		0,115	2,204	1294	-0,663	1872	x	
		0,1175	2,473	1354	-1,169	1901	x	
		0,1178	2,526	1370	-1,418	1924	x	
		0,1179	2,546	1376	-1,575	1952	x	x
		0,118	2,568	1381	-4,528	2205	u	
		0,12	¿100	1498				
2205	3,4	0,115	2,079	1256	-0,364	1842	x	
		0,118	2,277	1290	-0,487	1843	x	
		0,12	2,491	1339	-0,743	1849	x	
		0,1205	2,576	1363	-0,941	1859	x	
		0,1208	2,644	1385	-1,233	1885	x	
		0,1209	2,671	1392	-1,468	1919	x	x
		0,121	¿100	2045				

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
2205	5,0	0,12	2,274	1276	-0,313	1820	x	
		0,123	2,593	1344	-0,597	1820	x	
		0,1235	2,700	1380	-0,881	1839	x	
		0,1237	2,764	1404	-1,385	1897	x	x
		0,1238	¿100	1926				
2601	-	0,105	2,183	1596	-1,457	2316	x	
		0,1052	2,212	1599	-1,718	2373	x	x
		0,1053	¿100	2595				
2601	2,0	0,1	1,712	1447	-0,380	2198	x	
		0,105	1,970	1508	-0,552	2198	x	
		0,106	2,045	1530	-0,626	2198	x	
		0,107	2,136	1551	-0,747	2210	x	
		0,108	2,260	1597	-1,017	2232	x	
		0,1085	2,351	1626	-1,507	2310	x	x
		0,10855	2,362	1632	-1,702	2387	x	x
		0,10856	2,365	1632	-2,279	2601	u	
		0,10857	¿100				u	
		0,10923	2,800	1863	-3,020	2174	u	x
2601	3,4	0,1	1,661	1427				
		0,11	2,239	1566				
		0,111	2,401	1619	-0,960	2198	x	x
		0,1111	2,426	1627	-1,070	2212	x	x
		0,1112	2,452	1637	-1,260	2240	x	
		0,11125	2,467	1644	-1,436	2287	x	
		0,11126	2,470	1647	-1,498	2312	x	x
		0,11127	2,474	1647	-3,952	2601	u	
		0,111275	¿100				u	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
2601	5,0	0,1	1,622	1414	-0,152	2146	x	
		0,11	2,065	1494	-0,275	2146	x	
		0,1135	2,457	1609	-0,701	2158	x	
		0,1136	2,483	1625	-0,787	2166	x	
		0,1138	2,545	1656	-1,267	2232	x	
		0,11381	2,551	1656	-1,352	2261	x	x
		0,11382	¿100				u	
3467	-	0,09	1,791	2072	-0,894	3017	x	
		0,0904	1,833	2090	-1,009	3028	x	
		0,0908	1,883	2113	-1,211	3078	x	
		0,091	1,911	2137	-1,424	3137	x	
		0,0911	1,927	2140	-1,834	3388	x	x
		0,09115	¿100	3353				
3467	2,0	0,09	1,652	1990	-0,432	2923	x	
		0,093	1,896	2090	-0,725	2948	x	
		0,0938	2,020	2153	-1,145	3026	x	
		0,0939	2,041	2166	-1,345	3101	x	
		0,09393	2,048	2171	-1,489	3187	x	x
		0,093935	2,049	2171	-1,667	3467	u	
		0,093938			¡ -4	3467		
3467	3,4	0,093	1,745	2003	-0,337	2897	x	
		0,095	1,918	2074	-0,486	2897	x	
		0,096	2,071	2155	-0,791	2925	x	
		0,0962	2,121	2181	-1,040	2971	x	
		0,09626	2,139	2195	-1,231	3040	x	
		0,09627	2,142	2195	-1,293	3068	x	x
		0,09628	¿100	3445				

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
3467	5,0	0,095	1,775	1987	-0,232	2856	x	
		0,098	2,085	2126	-0,501	2856	x	
		0,0984	2,185	2185	-0,846	2902	x	
		0,09846	2,207	2205	-1,078	2964	x	
		0,09847	2,212	2206	-1,162	2999	x	x
		0,09848	¿100	3317				
4335	-	0,08	1,559	2571	-0,709	3749	x	
		0,081	1,661	2640	-0,973	3805	x	
		0,08125	1,694	2641	-1,136	3860	x	
		0,081375	1,712	2668	-1,295	3924	x	
		0,081425	1,720	2668	-1,425	4009	x	x
		0,081475	¿100	4232				
4335	2,0	0,08	1,450	2475	-0,366	3677	x	
		0,0835	1,744	2646	-0,759	3713	x	
		0,0838	1,795	2685	-0,949	3754	x	
		0,0839	1,815	2701	-1,079	3804	x	
		0,08398	1,833	2716	-1,343	4003	x	x
		0,084	¿100	4187				
4335	3,4	0,085	1,723	2589	-0,442	3601	x	
		0,086	1,896	2725	-0,913	3708	x	
		0,08603	1,905	2736	-0,986	3726	x	
		0,08605	1,911	2741	-1,055	3759	x	
		0,08606	1,914	2741	-1,102	3788	x	
		0,08607	1,918	2752	-1,168	3852	x	x
		0,08608	¿100	4209				
4335	5,0	0,088	1,967	2742	-0,842	3650	x	
		0,08803	1,979	2765	-1,013	3726	x	x
		0,08804	¿100	4153				

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
5201	-	0,073	1,424	3079	-0,649	4500	x	
		0,074	1,528	3164	-0,938	4584	x	
		0,07425	1,564	3203	-1,190	4715	x	
		0,0743	1,572	3207	-1,339	4833	x	x
		0,07435	¿100	5010				
5201	2,0	0,075	1,460	3059	-0,457	4427	x	
		0,076	1,566	3148	-0,628	4427	x	
		0,0765	1,648	3231	-0,914	4520	x	
		0,0766	1,670	3250	-1,115	4660	x	
		0,07662	1,674	3255	-1,228	4783	x	x
		0,07663	¿100	5089				
5201	3,4	0,076	1,434	3002	-0,281	4334	x	
		0,077	1,515	3067	-0,343	4334	x	
		0,078	1,635	3172	-0,500	4334	x	
		0,0785	1,744	3283	-0,941	4500	x	
		0,07852	1,751	3294	-1,040	4571	x	x
		0,07853	¿100	5105				
5201	5,0	0,076	1,362	2924	-0,156	4323	x	
		0,08	1,717	3200	-0,430	4323	x	
		0,0803	1,803	3309	-0,840	4408	x	
		0,08031	1,807	3309	-0,910	4453	x	x
		0,08032	¿100	4983				
5531	-	0,07	1,321	3220	-0,535	4780	x	
		0,072	1,518	3409	-1,169	5017	x	x
		0,0721	¿100	5287				
5531	2,0	0,073	1,442	3285	-0,476	4690	x	
		0,074	1,566	3391	-0,738	4746	x	
		0,07425	1,614	3458	-1,018	4889	x	x
		0,074375	¿100	5032				

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
5531	3,4	0,075	1,503	3288	-0,365	4639	x	
		0,076	1,659	3453	-0,693	4670	x	
		0,0761	1,687	3486	-0,866	4745	x	x
		0,07615	¿100	5241				
5531	5,0	0,076	1,470	3218	-0,212	4598	x	
		0,077	1,573	3301	-0,286	4597	x	
		0,0775	1,651	3386	-0,390	4597	x	
		0,07775	1,711	3452	-0,550	4597	x	
		0,077875	1,756	3531	-0,952	4814	x	x
		0,077907	¿100	4821				
6501	-	0,065	1,252	3819	-0,543	5627	x	
		0,066	1,348	3927	-0,763	5692	x	
		0,06625	1,380	3965	-0,908	5760	x	
		0,066375	1,399	3993	-1,062	5882	x	
		0,066407	1,404	4008	-1,143	5977	x	x
		0,066438	¿100	6442				
6501	2,0	0,06	0,971	3538	-0,193	5503	x	
		0,065	1,171	3697	-0,290	5503	x	
		0,067	1,303	3826	-0,405	5503	x	
		0,068	1,410	3950	-0,580	5553	x	
		0,0684	1,478	4044	-0,836	5662	x	
		0,06845	1,489	4072	-0,924	5733	x	
		0,06848	1,496	4072	-1,018	5836	x	
		0,06849	1,498	4072	-1,076	5934	x	x
		0,0685	¿100				u	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
6501	3,4	0,07	1,515	4034	-0,580	5482	x	x
		0,0701	1,540	4078	-0,680	5514	x	
		0,07015	1,554	4097	-0,770	5564	x	
		0,07017	1,560	4097	-0,830	5604	x	
		0,07019	1,567	4114	-0,924	5703	x	x
		0,0702	¿100	6328				
6501	5,0	0,071	1,453	3885	-0,265	5408	x	
		0,0715	1,534	3998	-0,378	5408	x	
		0,0716	1,557	4026	-0,433	5408	x	
		0,0717	1,585	4074	-0,534	5408	x	
		0,07175	1,602	4111	-0,634	5408	x	
		0,07177	1,610	4122	-0,703	5480	x	
		0,07178	1,614	4135	-0,753	5507	x	
		0,07179	1,618	4148	-0,829	5577	x	x
		0,0718	¿100				u	

Quadratische Betrachtung - Ergebnisse

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
1000	2,0	0,0585	2,790	550	-0,580	846	x	x
1000	3,4	0,0593	2,800	547	-0,390	833	x	x
1000	5,0	0,0599	2,790	543	-0,260	823	x	x
2601	2,0	0,0435	2,800	1485	-0,540	2182	x	x

B.2.2. ainr2 für Streu- und Sammelrücken

Lineare Betrachtung - Ergebnisse

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
1000	2,0	0,00442	2,800	570	-1,800	872	x	x
1000	3,4	0,00457	2,794	559	-1,355	842	x	x
1000	5,0	0,0047	2,790	552	-1,050	827	x	x
2205	3,4	0,0019	1,742	1200	-0,718	1842	x	
		0,001925	1,760	1204	-0,736	1842	x	
		0,001975	1,798	1211	-0,774	1842	x	
		0,002	1,817	1215	-0,795	1842	x	
		0,002025	1,837	1218	-0,818	1842	x	
		0,002075	1,877	1224	-0,872	1849	x	
		0,002125	1,921	1247	-0,939	1849	x	
		0,002175	1,967	1252	-1,030	1864	x	
		0,002225	2,019	1270	-1,179	1888	x	
		0,00225	2,046	1284	-1,310	1912	x	
		0,00225781	2,056	1284	-1,376	1942	x	x
		0,00226562	¿8	2203	¡-7	2202	u	
		0,00228125	¿100	2084				
2601	2,0	0,001	1,183	1340	-0,490	2198	x	x
		0,0015	1,580	1423				
		0,0016	1,632	1431	-0,870	2217	x	x

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
2601	3,4	0,0015	1,550	1400			x	
		0,0016	1,596	1410			x	
		0,0017	1,677	1435			x	
		0,0019	1,876	1501			x	
		0,00191	1,890	1507			x	
		0,001911	1,890	1516			x	
		0,001912	1,891	1522			x	
		0,001913	1,893	1518			x	
		0,001914	1,894	1518			x	
		0,001915	1,897	1518			x	x
		0,001916	1,897	1522			u	
		0,001917	1,978	2598			u	
5531	3,4	0,00075	1,093	3007	-0,448	4639	x	
		0,000875	1,258	3151	-0,691	4670	x	x
		0,00089062	1,284	3185	-0,784	4745	x	
		0,00089843	1,299	3217	-0,875	4866	x	x
		0,00090233	¿100	5431				

Quadratische Betrachtung - Ergebnisse

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	r_{min}[5]	spalte[4]	fertig[10]	aus[11]
1000	2,0	0,00238	2,800	481	-1,130	161	x	x
1000	3,4	0,00234	2,800	481	-0,930	171	x	x
1000	5,0	0,00231	2,800	485	-0,750	184	x	x
2601	2,0	0,001	1,798	1234	-0,710	420	x	
		0,0015	2,124	1215	-0,880	420	x	x

B.3. Alle Ergebnisse $(2 \cdot 1\pi)$ – ainr für Sammelrücken

Lineare Betrachtung - Bandbreitenübersicht

länge[1]	angabe[7]	kaiser[6]	3dB (qs[8])	3dB (sq[9])	6dB (qs[8])	6dB (sq[9])
435	3,00 nm	3,4	2,25 nm	2,25 nm	3,10 nm	3,15 nm
501	2,60 nm	2,0	1,75 nm	1,80 nm	2,45 nm	2,50 nm
		3,4	1,95 nm	1,95 nm	2,70 nm	2,73 nm
		5,0	2,10 nm	2,15 nm	2,95 nm	3,00 nm
1001	1,30 nm	2,0	0,80 nm	0,90 nm	1,10 nm	1,25 nm
		3,4	0,90 nm	1,00 nm	1,25 nm	1,40 nm
		5,0	1,00 nm	1,10 nm	1,40 nm	1,50 nm
2205	0,59 nm	3,4	0,40 nm	0,45 nm	0,55 nm	0,65 nm
2601	0,50 nm	2,0	0,30 nm	0,35 nm	0,40 nm	0,50 nm
		3,4	0,30 nm	0,40 nm	0,40 nm	0,55 nm
		5,0	0,35 nm	0,45 nm	0,45 nm	0,60 nm
3251	0,40 nm	2,0	0,25 nm	0,30 nm	0,30 nm	0,40 nm
		3,4	0,25 nm	0,30 nm	0,35 nm	0,45 nm
		5,0	0,30 nm	0,35 nm	0,40 nm	0,50 nm
5533	0,235 nm	3,4	0,20 nm	0,20 nm	0,25 nm	0,25 nm

Lineare Betrachtung - Ergebnisse

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	fertig[10]	aus[11]
435	3,4	0,15	2,136	233	x	
		0,18	2,806	244	x	x
		0,185	2,947	247	x	
501	2,0	0,16	2,530	288	x	
		0,165	2,681	293	x	
		0,168	2,782	298	x	
		0,1685	2,799	298	x	x
		0,169	2,817	298	x	
		0,17	2,850	300	x	
501	3,4	0,17	2,682	284		
		0,173	2,770	287		
		0,174	2,801	287	x	x
		0,18	3,000	293		
501	5,0	0,16	2,353	270	x	
		0,17	2,580	275	x	
		0,175	2,706	277	x	
		0,176	2,732	277	x	
		0,177	2,759	279	x	
		0,178	2,786	279	x	
		0,1785	2,800	279	x	x
		0,179	2,814	279	x	
		0,18	2,842	279	x	
1001	2,0	0,13	2,602	683	x	
		0,131	2,733	707	x	
		0,1313	2,785	722	x	
		0,1314	2,804	727	x	x
		0,132	2,980	779	x	
		0,135	¿100		u	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	fertig[10]	aus[11]
1001	3,4	0,12	1,897	570		
		0,13	2,241	594		
		0,135	2,495	623	x	
		0,136	2,561	630		
		0,138	2,727	655	x	
		0,1385	2,780	660		
		0,1387	2,803	668	x	x
		0,14	3,014	711		
1001	5,0	0,138	2,391	588	x	
		0,139	2,436	592	x	
		0,14	2,484	592	x	
		0,141	2,535	600	x	
		0,145	2,796	628	x	x
		0,15	¿100		u	
2205	3,4	0,09	1,633	1352	x	
		0,0925	1,810	1422	x	
		0,09375	1,962	1507	x	
		0,094062	2,025	1559	x	
		0,094218	2,068	1597	x	x
		0,094374	¿100	2040		
2601	2,0	0,0816	1,854	2032	x	x
		0,08161	1,860	2045	x	
		0,08162	1,867	2064	x	
		0,08163	1,874	2076	x	
		0,08164	1,882	2088	x	
		0,081643	1,885	2099	x	
		0,081645	1,887	2120	x	x
		0,081647	2,155	2601	u	
		0,08166	¿80		u	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	fertig[10]	aus[11]
2601	3,4	0,082	1,463	1579	x	
		0,084	1,581	1634	x	
		0,086	1,773	1758	x	
		0,0865	1,865	1841	x	
		0,0866	1,892	1867	x	
		0,0867	1,927	1914	x	x
		0,08673	1,941	1947	x	
		0,086735	1,943	1947	x	
		0,086736	1,944	1947	x	
		0,086737	1,944	1947	x	x
		0,086738	1,945	1947	u	
		0,086739	2,027	2601	u	
		0,086745	¿100		u	
2601	5,0	0,086	1,512	1533	x	
		0,087	1,562	1550	x	
		0,089	1,684	1608	x	
		0,09	1,767	1640	x	
		0,091	1,886	1718	x	
		0,0915	1,986	1807	x	
		0,0916	2,019	1845	x	x
		0,09161	2,023	1845	x	
		0,091615	2,025	1845	x	
		0,091616	2,026	1873	x	x
		0,091617	2,771	2601	x	x
		0,091618	¿15	2601	u	
		0,09162	¿100		u	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	fertig[10]	aus[11]
3251	2,0	0,065	1,071	1905	x	
		0,07	1,293	2070	x	
		0,072	1,461	2224	x	
		0,0723	1,501	2270	x	
		0,0725	1,534	2324	x	
		0,0727	1,576	2375	x	
		0,0728	1,604	2431	x	
		0,0729	1,640	2505	x	
		0,07295	1,667	2558	x	
		0,07297	1,682	2601	x	x
		0,07298	3,498	3251	u	
		0,073	¿100		u	
3251	3,4	0,05	0,696	1709		
		0,07	1,169	1888		
		0,075	1,409	2040	x	
		0,077	1,606	2213	x	
		0,0775	1,726	2408	x	
		0,07752	1,736	2413	x	x
		0,07753	1,853	3251	x	
		0,07754	¿100		u	

länge[1]	kaiser[6]	si_{max}[2]	r_{max}[3]	spalte[4]	fertig[10]	aus[11]
3251	5,0	0,08	1,541	2028	x	
		0,0805	1,586	2056	x	
		0,081	1,640	2105	x	
		0,0815	1,714	2182	x	
		0,0816	1,733	2182	x	
		0,0817	1,755	2213	x	
		0,0818	1,782	2269	x	
		0,08185	1,798	2300	x	
		0,08186	1,802	2306	x	
		0,08187	1,806	2312	x	x
		0,08188	¿100		u	
5533	3,4	0,05	0,780	3111	x	
		0,055	0,945	3281	x	
		0,0575	1,075	3448	x	
		0,05875	1,182	3632	x	
		0,059375	1,285	3937	x	x
		0,059688	¿100	4643	u	

[1]Die Länge des Rückens in Gräben.
[2]Das Maximum der zugrundeliegenden Filterkoeffizienten (der *si*-Funktion).
[3]Das Maximum der berechneten Reflexionsfaktoren in Prozent.
[4]Die Grabennummer, in welcher der maximale bzw. minimale Reflexionsfaktor auftritt.
[5]Das Minimum der berechneten Reflexionsfaktoren in Prozent.
[6]Der bei **fxswin** (siehe 2.4) angegebene Koeffizient β der Kaiserfunktion (Fensterfunktion).
[7]Die bei **fxsfind** (siehe 2.3) angegebene Bandbreite (Wellenlängenabstand).
[8]Die Ausgänge werden zuerst einzeln quadriert und dann aufsummiert.
[9]Alle Ausgänge werden aufsummiert, und anschließend wird deren Summe quadriert.
[10]Ein `x` in dieser Spalte gibt an, daß die Reflexionsfaktoren komplett berechnet wurden.
[10]Steht ein `u` in dieser Spalte, so bedeutet dies, daß die berechneten Werte unbrauchbar sind.
[11]Ein `x` ist in dieser Spalte vorhanden, wenn die Filterausgänge berechnet wurden.

C. Quellcodes der MatLab-Skripten

C.1. acad_v.m

```
% programm zur vorbereitung der erforderlichen daten
% fuer die autocad-zeichnung.
%
%
% michael becker, 02.03.2000 - 06.04.2000


% -- vorarbeiten --
cd ~
clear
close all
clc

verz = 'm/daten/';


% -- ueberschrift --
disp(' vorbereitung der daten fuer c++, damit gezeichnet werden kann')
disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
disp(' ')

% -- anzahl filter --
anzahl = input('Anzahl der Filter pro Struktur [6] ? ');
if(isempty(anzahl)); anzahl = 6; end;

% -- speicherverzeichnis relativ zu m/daten/acad/
verz2 = input('acad-speicherverzeichnis relativ zu ''~/m/daten/acad/'' ?      ');
if(isempty(verz2))
   % direkt nach acad speichern
   scrverz = [verz 'acad/'];
else
   str = [verz 'acad/' verz2];
   if(exist(str)~=7)
      % verzeichnis gibt's noch nicht -> anlegen
      mkdir(str);
   end
   scrverz = [str '/'];
end

verz3 = input('verzeichnis fuer filterdaten (relativ zu o.g. verzeichnis) ? ');
str = [scrverz verz3];
if(exist(str)~=7)
   % verzeichnis gibt's noch nicht -> anlegen
   mkdir(str);
end
datverz = [str '/'];


% -- anzahl der filter pro struktur speichern --
```

```
disp(' ')
zd = [scrverz 'zeichng.dat'];
if(exist(zd)==2)
   % datei existiert schon -> pruefen
   disp(''zeichng.dat'' existiert bereits.')

   data = laden(zd);
   if(data(1)~=anzahl)
      % anzahl passt nicht
      disp('anzahl der filter pro struktur passt nicht.')
      disp('speicherung in diesem verzeichnis nicht moeglich.')
      return;
   end
else
   fid = fopen(zd, 'w');
   fprintf(fid, '%d\n', anzahl);
   fclose(fid);
   disp('anzahl der filter pro struktur nach')
   disp([''~/' scrverz 'zeichng.dat'' gespeichert.'])
end


% -- information --
disp(' ')
disp('------------------------------------------------------------')
disp(' alle folgenden werte muessen als zeichenketten eingegeben')
disp(' werden, also in einfachen anfuehrungszeichen (hochkommas)!')
disp('------------------------------------------------------------')


% -- vorbereitung infostring --
% spaltenbreite setzen
spb = 10;

% name der struktur
name = ['struktur in ''~/' datverz(1:length(datverz)-1) ''''];
for ii=length(name):7*spb-1
   name = [name ' '];
end
filterdata(1,:) = name;

% ueberschrift fuer infostring anfertigen
sp(1,:) = 'breite   |';
sp(2,:) = 'rstart   |';
sp(3,:) = 'anz_pi   |';
sp(4,:) = 'kaiser   |';
sp(5,:) = 'si_max   |';
sp(6,:) = 'lin/quad |';
sp(7,:) = 'bemerkung ';
filterdata(2,:) = [sp(1,:) sp(2,:) sp(3,:) sp(4,:) sp(5,:) sp(6,:) sp(7,:)];

% -- jeweils das gewuenschte filter auswaehlen --
for ii=1:anzahl
   disp(' ')
   disp(['filter nummer ' num2str(ii) ' :'])

   % -- eingaben --
   % rueckenbreite
   rueckenbreite = input('  rueckenbreite, z.b. ''6501''            : ');
   % rstart
   rstart        = input('  rstart (z.b. ''00985'', ''-2'' fuer ainr2) : ');
   % anz_pi
   anz_pi        = input('  anz_pi (z.b. ''2'' fuer -2pi bis +2pi)   : ');
   % kaiser_beta
   kaiser_beta   = input('  kaiser_beta (z.b. ''34'' fuer 3.4)       : ');
   % si_max
   si_max        = input('  si_max (z.b. ''0015'' fuer 0.0015        : ');
   % lq
   lq            = input('  linear (''l'') oder quadratisch (''q'')    : ');
```

```
% dis=0 bedeutet kontinuierlich
% dis=1 bedeutet gitterbreiten diskretisiert
% dis=2 bedeutet reflexionsfaktoren diskretisiert

dis    = input('Sind die Gitterbreiten diskretisiert [enter=nein] ?      ');
if(isempty(dis))
   dis = input('Sind die Reflexionsfaktoren diskretisiert [enter=nein] ? ');
   if(isempty(dis))
      dis = 0;
   else
      dis = 2;
   end
else
   dis = 1;
end

% -- dateinamen vorbereiten --
gemeinsam = sprintf('%s%spi/%s/', verz, anz_pi, rueckenbreite);

if(str2num(rstart)==-2)
   % daten von ainr2 berechnet
   streuGB = ...
      sprintf('%s%s/streu/gb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   streuWB = ...
      sprintf('%s%s/streu/wb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   sammelGB = ...
      sprintf('%s%s/sammel/gb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   sammelWB = ...
      sprintf('%s%s/sammel/wb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
else
   streuGB = ...
      sprintf('%sstreu/gb/%s', gemeinsam, rstart);
   streuWB = ...
      sprintf('%sstreu/wb/%s', gemeinsam, rstart);
   sammelGB = ...
        sprintf('%s%s/sammel/gb/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   sammelWB = ...
      sprintf('%s%s/sammel/wb/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end

% -- diskretisierte werte? --
bem = 'kontin.';

if(dis==1)
   % gitterbreiten diskretisiert
   streuGB = [streuGB 'd'];
   streuWB = [streuWB 'd'];
   bem = 'GB diskr.';
end

if(dis==2)
   % reflexionsfaktoren diskretisiert
   streuGB = [streuGB 'rd'];
   streuWB = [streuWB 'rd'];
   bem = 'r diskr.';
end

% -- infostring anfertigen --
rueckenbreite = [' ' rueckenbreite];
for jj=length(rueckenbreite):spb-1
   rueckenbreite = [rueckenbreite ' '];
end

if(str2num(rstart)==-2)
   rstart = '--';
else
   rstart = ['0,' rstart];
end;
for jj=length(rstart):spb-1
   rstart = [rstart ' '];
```

```
end

for jj=length(anz_pi):spb-1
   if(jj<spb/2)
      anz_pi = [' ' anz_pi];
   else
      anz_pi = [anz_pi ' '];
   end
end

if(length(kaiser_beta)==2)
   kaiser_beta = [kaiser_beta(1) ',' kaiser_beta(2)];
end
for jj=length(kaiser_beta):spb-1
   if(jj<spb/2)
      kaiser_beta = [' ' kaiser_beta];
   else
      kaiser_beta = [kaiser_beta ' '];
   end
end

si_max = ['0,' si_max];
for jj=length(si_max):spb-1
   si_max = [si_max ' '];
end

for jj=length(lq):spb-1
   if(jj<spb/2)
      lq = [' ' lq];
   else
      lq = [lq ' '];
   end
end

for jj=length(bem):spb-1
   bem = [bem ' '];
end

filterdata(ii+2,:) = [rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq, bem];


% -- eigentliches arbeiten --
% die daten werden nur kopiert, d.h. laden und wieder speichern
streuGBz = [datverz 'st' num2str(ii) '_gb.dat'];
streuWBz = [datverz 'st' num2str(ii) '_wb.dat'];
sammelGBz = [datverz 'sa' num2str(ii) '_gb.dat'];
sammelWBz = [datverz 'sa' num2str(ii) '_wb.dat'];
disp(' ')

% gitterbreiten streu
data = laden(streuGB);
disp(['gitterbreiten streu aus ''' streuGB ''' geladen.'])
saven(streuGBz, data);
disp(['gitterbreiten streu nach ''' streuGBz ''' gespeichert.'])

% weiten streu
data = laden(streuWB);
disp(['weiten streu aus ''' streuWB ''' geladen.'])
saven(streuWBz, data);
disp(['weiten streu nach ''' streuWBz ''' gespeichert.'])

% gitterbreiten sammel
data = laden(sammelGB);
disp(['gitterbreiten sammel aus ''' sammelGB ''' geladen.'])
saven(sammelGBz, data);
disp(['gitterbreiten sammel nach ''' sammelGBz ''' gespeichert.'])

% weiten sammel
data = laden(sammelWB);
disp(['weiten sammel aus ''' sammelWB ''' geladen.'])
```

```
   saven(sammelWBz, data);
   disp(['weiten sammel nach ''' sammelWBz ''' gespeichert.'])
end


% -- infostring speichern --
fid = fopen([datverz 'info.txt'], 'w');
fprintf(fid, '%s\n\n', filterdata(1,:));
shelp = size(filterdata);
for ii=2:shelp(1)
   fprintf(fid, '%s\n', filterdata(ii,:));
end
fclose(fid);

disp(' ')
disp(['filterinformation nach ''~/' datverz 'info.txt'' gespeichert.'])
```

C.2. ainr_l.m

```
function rsammel = ainr_l(rueckenbreite, anz_pi, kaiser_beta, si_max, lq);

% funktion zum laden der von c++ berechneten reflexionsfaktoren
% fuer den sammelruecken sowie deren visualisierung.
%
% rsammel = ainr_l(rueckenbreite, anz_pi, kaiser_beta, si_max, lq);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
%
%   rsammel         die berechneten reflexionsfaktoren fuer den
%                   sammelruecken
%
%
% michael becker, 10.02.2000 - 04.04.2000


% -- vorarbeiten --
close all

% relatives verzeichnis
verz = 'm/daten/';

if(isunix)
   % unix system
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end


% -- laden, sammel --
str = sprintf('%s%spi/%s/%s/sammel/r/%s%s', ...
   verz, anz_pi, rueckenbreite, kaiser_beta, si_max, lq);
%disp(str)
rsammel = laden(str);
disp(['reflexionsfaktoren sammel aus ''' str ''' geladen.'])
```

```
% -- zeichnen --
plot(rsammel*100)

[mi idmi] = min(rsammel);
[ma idma] = max(rsammel);
str = sprintf('rmin = %6.4f%% (spalte %d), rmax = %6.4f%% (spalte %d)', ...
   mi*100, idmi, ma*100, idma);
title(str)
```

C.3. ainr_v.m

```
function a = ainr_v(rueckenbreite, anz_pi, kaiser_beta, si_original, si_max);

% neuen maximalwert der filterkoeffizienten setzen.
%
% a = ainr_v(rueckenbreite, anz_pi, kaiser_beta, si_original, si_max);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_original     das maximum der original si-funktion,
%                   z.b. '01' fuer 0.01
%   si_max          maximum der zu berechnenden si-funktion,
%                   z.b. '02' fuer 0.02
%
%   a               die berechneten filterkoeffizienten
%
%
% michael becker, 14.02.2000 - 20.03.2000


% -- vorarbeiten --

% relatives verzeichnis
verz = 'm/daten/';

if(isunix)
   % unix system
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end

gemeinsam = sprintf('%s%spi/%s/%s/a/', ...
   verz, anz_pi, rueckenbreite, kaiser_beta);


% -- original laden --
str = [gemeinsam si_original];
data = laden(str);
disp(['original koeffizienten aus ''~/' str ''' geladen.'])


% -- neue berechnen --
a = data .* str2num(['.' si_max]) ./ max(data);


% -- neue speichern --
str = [gemeinsam si_max];
saven(str, a);
disp(['neue koeffizienten in ''~/' str ''' gespeichert.'])
```

C.4. ainr2_l.m

```
function [rstreu, rsammel] = ...
   ainr2_l(rueckenbreite, anz_pi, kaiser_beta, si_max, lq);

% funktion zum laden der von c++ (ainr2) berechneten reflexionsfaktoren
% fuer streu- und sammelruecken sowie deren visualisierung.
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
%
%   rsammel         die berechneten reflexionsfaktoren fuer den
%                   sammelruecken
%   rstreu          die berechneten reflexionsfaktoren fuer den
%                   streuruecken
%
%
% michael becker, 10.02.2000 - 04.04.2000


% -- vorarbeiten --
close all

% relatives verzeichnis
verz = 'm/daten/';

if(isunix)
   % unix system
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end


% -- laden, streu --
str = sprintf('%s%spi/%s/%s/streu/r2/%s%s', ...
   verz, anz_pi, rueckenbreite, kaiser_beta, si_max, lq);
rstreu = laden(str);
disp(['reflexionsfaktoren streu aus ''' str ''' geladen.'])

% -- laden, sammel --
str = sprintf('%s%spi/%s/%s/sammel/r2/%s%s', ...
   verz, anz_pi, rueckenbreite, kaiser_beta, si_max, lq);
rsammel = laden(str);
disp(['reflexionsfaktoren sammel aus ''' str ''' geladen.'])


% -- zeichnen --

% sammel
subplot(211)
plot(rsammel*100)
[mi idmi] = min(rsammel);
[ma idma] = max(rsammel);
str = sprintf('rmin = %6.4f%% (spalte %d), rmax = %6.4f%% (spalte %d)', ...
   mi*100, idmi, ma*100, idma);
title(str)

% streu
subplot(212)
plot(rstreu*100)
```

C.5. aus_l.m

```
function [lambdas, eqds, esdq] = ...
   aus_l(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq, nice, niceverz);

% darstellung der ausgangsleistung in abhaengigkeit von
% der frequenz/wellenlaenge des einfallenden lichts fuer
% eine bestimmte laserfrequenz bei schraeger berechnung
% der einzelnen graeben
%
% [lambdas, eqds, esdq] = ...
%    aus_l(rueckenbreite, rstart, kaiser_beta, si_max, lq);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   rstart          der startwert des reflexionsfaktors des streurueckens,
%                   z.b. '00985' fuer 0.00985 = 0.985% oder
%                   '-1' = nur berechnung des sammelrueckens oder
%                   '-2' = berechnung streu und sammel fuer ainr2
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
%   nice            optional:
%                   1. groesse fuer leistung in db, 1.5*[nice] x [nice] cm
%                   2. bei verwendung von niceverz titel der grafik
%   niceverz        optional: speicherverzeichnis fuer leistung in db,
%                   20x26cm mit titel, s.o.
%
%   lambdas         der vektor fuer die berechneten wellenlaengen
%   eqds            die ergebnisse fuer ausgaenge erst
%                   quadrieren und dann summieren
%   eqds            die ergebnisse fuer ausgaenge erst
%                   summieren und dann quadrieren
%
%
% michael becker, 10.02.2000 - 27.04.2000


% -- vorarbeiten --
clc
close all

% relatives verzeichnis
verz = 'm/daten/';
fineverz = 'diplom/doku/eps/';

if(isunix)
   % unix system
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
   fineverz = ['d:/studium/' fineverz];
end

gemeinsam = sprintf('%s%spi/%s/', verz, anz_pi, rueckenbreite);
l0 = 1300e-9;
P = 8;


% -- dateinamen vorbereiten --
if(str2num(rstart)==-1)
   % nur sammelruecken fuer ainr
   quelle_sammel = ...
      sprintf('%s%s/sammel/r/%s%s', gemeinsam, kaiser_beta, si_max, lq);
```

```
   quelle_ergebnis = ...
      sprintf('%s%s/p/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   plotdatei = ...
      sprintf('%s%s/p/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)==-2)
   % streu- und sammelruecken fuer ainr2
   quelle_streu = ...
      sprintf('%s%s/streu/r2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   quelle_sammel = ...
      sprintf('%s%s/sammel/r2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   quelle_ergebnis = ...
      sprintf('%s%s/p2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   plotdatei = ...
      sprintf('%s%s/p2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
   % streu- und sammelruecken unterschiedlich gewichtet
   quelle_streu = ...
      sprintf('%sstreu/r/%s', gemeinsam, rstart);
   quelle_sammel = ...
        sprintf('%s%s/sammel/r/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   quelle_ergebnis = ...
      sprintf('%s%s/p/%s%s-%s', gemeinsam, kaiser_beta, si_max, lq, rstart);
   plotdatei = ...
      sprintf('%s%s/p/%s%s-%s', gemeinsam, kaiser_beta, si_max, lq, rstart);
end


% -- ueberschrift --
disp('    Darstellung der Ausgangsleistung des Sammelgrabens in')
disp('  Abhaengigkeit von der Wellenlaenge des einfallenden Lichts')
disp(' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
disp(' ')

if(str2num(rstart)~=-1)
   % -- laden, streu --
   rstreu = laden(quelle_streu);
   disp(['reflexionsfaktoren streu aus ''' quelle_streu ''' geladen.'])
end

% -- laden, sammel --
rsammel = laden(quelle_sammel);
disp(['reflexionsfaktoren sammel aus ''' quelle_sammel ''' geladen.'])

% -- ergebnisse laden --
data = laden(quelle_ergebnis);
disp(['ergebnisse aus ''' quelle_ergebnis ''' geladen.'])

% zuweisen der ergebnis-daten
schritte = length(data) / 7;
eqds = zeros(3, schritte);
esdq = zeros(3, schritte);
lambdas = data(1:schritte);

for ii=1:3
   eq1 = (2*ii-1)*schritte+1;
   eq2 = (2*ii)*schritte;
   es1 = eq2 + 1;
   es2 = (2*ii+1)*schritte;

   eqds(ii, :) = data(eq1:eq2)';
   esdq(ii, :) = data(es1:es2)';
end

% -- transversalfilter ergebnis laden --
quelle_trans = ...
     sprintf('%s%s/trans/%d', gemeinsam, kaiser_beta, schritte);
```

```
if(exist(quelle_trans)==2)
   % transversalfilter laden
   trans = laden(quelle_trans);
   disp(['transversalfilterergebnis aus ''' quelle_trans ''' geladen.'])
else
   trans = [];
end


% -- Bestimmung der Einsattelung und der leistung bei l0 --

einsatt = 1000 * ones(2, 1);
pl0 = 1000 * ones(2, 1);
% aufbau: index 1 = eqds, index 2 = esdq

index = find(lambdas==l0*1e9);

if(isempty(index)==0)
   einsatt(1) = abs(eqds(2, index));
   einsatt(2) = abs(esdq(2, index));
   pl0(1) = eqds(1, index);
   pl0(2) = esdq(1, index);
end


% -- grafische darstellung des ergebnisses --

if(exist('nice')==0)
   % - grafische komplettdarstellung -

   % 1. zeile, links: reflexionsfaktoren streu- und sammelgraben
   subplot(321)
   plot((1:length(rsammel)), rsammel*100, 'b');
   if(str2num(rstart)~=-1)
      hold on
      plot((1:length(rstreu)), rstreu*100, 'r');
      hold off
      title('Streu- (rot) und Sammelruecken (blau)');
   else
      title('Sammelruecken');
   end
   xlabel('Rueckenzelle')
   ylabel('Reflexionsfaktor in %')
   ax = axis;
   axis([1, length(rsammel), ax(3) ax(4)]);

   % 1. zeile, rechts: absolute leistung
   %    eqds rot, esdq blau
   subplot(322), plot(lambdas, eqds(1, :), 'r')
   hold on
   subplot(322), plot(lambdas, esdq(1, :), 'b')
   hold off
   xlabel('Freiraumwellenlaenge lambda')
   ylabel('|P_s_a_m_m_e_l |')
   title('Ausgangsleistung (rot=eqds, blau=esdq)')

   % 2. zeile, links: phase der leistung
   %    eqds rot, esdq blau
   subplot(323), plot(lambdas, eqds(3, :), 'r')
   hold on
   subplot(323), plot(lambdas, esdq(3, :), 'b')
   hold off
   xlabel('Freiraumwellenlänge lambda');
   ylabel('phase(P_s_a_m_m_e_l ) in Grad');
   title('Phasengang (rot=eqds, blau=esdq)')

   % 2. zeile, rechts: relative leistung logarithmisch
   %    eqds rot, esdq blau
   subplot(324), plot(lambdas, eqds(2, :), 'r')
   hold on
```

```
subplot(324), plot(lambdas, esdq(2, :), 'b')
hold off
xlabel('Freiraumwellenlänge lambda');
ylabel('10.*log10|P_s_a_m_m_e_l / P_m_a_x | in dB');
title('Relative Leistung (rot=eqds, blau=esdq)')
% axen setzen
ax = axis;
axis([ax(1) ax(2) -100 0]);

% 3. zeile: datenwerte ausgeben
subplot(325), axis([0 10 0 14])

x1 = 1;
x2 = 9;

text(x1, 13, 'datum und uhrzeit :')
text(x2, 13, time)

text(x1, 12, 'berechnungsart :')
if(lq(1)=='q')
   str = 'quadratische betrachtung';
else
   str = 'lineare betrachtung';
end
text(x2, 12, str)

text(x1, 11, 'quelle streu :')
if(str2num(rstart)~=-1)
   str = sprintf('~/%s', quelle_streu);
   text(x2, 11, str)
end

text(x1, 10, 'quelle sammel :')
str = sprintf('~/%s', quelle_sammel);
text(x2, 10, str)

text(x1, 9, 'quelle ergebnis :')
str = sprintf('~/%s', quelle_ergebnis);
text(x2, 9, str)

text(x1, 8, 'laserwellenlaenge (l0):')
str = sprintf('%g nm', l0*1e9);
text(x2, 8, str)

text(x1, 7, 'rueckenhoehe :')
str = sprintf('%d', P);
text(x2, 7, str)

text(x1, 6, 'anzahl schritte :')
str = sprintf('%d', schritte);
text(x2, 6, str)

text(x1, 5, 'reflexionsfaktoren streu:')
if(str2num(rstart)~=-1)
   [mi idmi] = min(rstreu);
   [ma idma] = max(rstreu);
   str = sprintf('rmin = %5.3f%% (spalte %d), rmax = %5.3f%% (spalte %d)', ...
      mi*100, idmi, ma*100, idma);
   text(x2, 5, str)
end

text(x1, 4, 'reflexionsfaktoren sammel:')
[mi idmi] = min(rsammel);
[ma idma] = max(rsammel);
str = sprintf('rmin = %5.3f%% (spalte %d), rmax = %5.3f%% (spalte %d)', ...
   mi*100, idmi, ma*100, idma);
text(x2, 4, str)

text(x1, 3, 'leistung bei l0 :')
str = sprintf('eqds=%5.3f%%, esdq=%5.3f%%', pl0.*100);
```

```
   text(x2, 3, str)

   text(x1, 2, 'einsattelung :')
   str = sprintf('eqds=%5.3fdB, esdq=%5.3fdB', einsatt);
   text(x2, 2, str)

   text(x1, 1, 'plotdatei :')
   str = sprintf('~/%s.eps', plotdatei);
   text(x2, 1, str)

   % -- in datei drucken --
   prpeps(20,26);

   str = sprintf('print -depsc %s', plotdatei);
   eval(str)
end

% relative leistung logarithmisch allein in extra fenster
%    eqds rot, esdq blau

% wenn nur sammelruecken berechnet wurde, halbieren der bandbreite!
if(str2num(rstart)==-1)
   h_lang = length(lambdas);
   h_mitte = lambdas(fix(h_lang / 2));
   h_start = lambdas(1);
   h_stop = lambdas(h_lang);
   h_start_neu = h_mitte - (h_mitte - h_start) / 2;
   h_stop_neu = h_mitte + (h_stop - h_mitte) / 2;
   h_delta_neu = (lambdas(2) - lambdas(1)) / 2;
   lambdas = [h_start_neu : h_delta_neu : h_stop_neu];
end

if(exist('nice')==0)
   % bandbreiten bestimmen
   % eqds:
   gef = find(eqds(2, :)>-3);
   eqds_bb3 = lambdas(gef(length(gef))+1) - lambdas(gef(1)-1);
   gef = find(eqds(2, :)>-6);
   eqds_bb6 = lambdas(gef(length(gef))+1) - lambdas(gef(1)-1);
   % esdq:
   gef = find(esdq(2, :)>-3);
   esdq_bb3 = lambdas(gef(length(gef))+1) - lambdas(gef(1)-1);
   gef = find(esdq(2, :)>-6);
   esdq_bb6 = lambdas(gef(length(gef))+1) - lambdas(gef(1)-1);
end

% jetzt zeichnen
figure
plot(lambdas, esdq(2, :), 'b')
if(exist('nice')==0)
   hold on

   % eqds nur dann, wenn nicht schoenes format
   plot(lambdas, eqds(2, :), 'r')

   % drei- und sechs-db-linie
   plot(lambdas, -3 * ones(length(lambdas), 1), 'k')
   plot(lambdas, -6 * ones(length(lambdas), 1), 'k')

   if(isempty(trans)==0)
      % transversalfilter zeichnen
      plot(lambdas, trans, 'g')
   end

   hold off
else
   if(exist('niceverz')==1)
      title(nice)
   end
end
```

```
if(exist('nice')==0)
   % nicht schoen -> alle daten
   str = sprintf(['Freiraumwellenlänge; Bandbreiten: ' ...
         'eqds(3dB)=%4.2fnm, eqds(6dB)=%4.2fnm, ' ...
         'esdq(3dB)=%4.2fnm, esdq(6dB)=%4.2fnm'], ...
      eqds_bb3, eqds_bb6, esdq_bb3, esdq_bb6);
else
   str = 'Freiraumwellenlänge [nm]';
end

xlabel(str);
ylabel('10 * log |P / P_m_a_x |  [dB]');

if(exist('nice')==0)
   % titel nur bei nicht schoen
   str = 'Relative Leistung (rt=eqds, bl=esdq';
   if(isempty(trans)==0)
      str = [str ', gn=trans'];
   end
   str = [str sprintf(') M=%s kai=%3.1f simax=0.%s %spi', ...
      rueckenbreite, str2num(kaiser_beta)/10, si_max, anz_pi)];
   if(str2num(rstart)==-1)
      % nur sammelruecken fuer ainr
      str = [str ' nursammel'];
   end
   if(str2num(rstart)==-2)
      % streu- und sammelruecken fuer ainr2
      str = [str ' ainr2'];
   end
   if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
      % streu- und sammelruecken unterschiedlich gewichtet
      str = sprintf('%s rstart=0.%s', str, rstart);
   end
   if(lq(1)=='l')
      str = [str ' linear'];
   end
   if(lq(1)=='q')
      str = [str ' quadratisch'];
   end
   title(str)
end

% axen setzen
ax = axis;
axis([min(lambdas) max(lambdas) -100 0]);

% -- in datei drucken --
if(exist('nice')==0)
   % - komplett -
   prpeps(20, 26);

   str = sprintf('print -depsc %sz', plotdatei);
   eval(str)

   % -- gleich ausdrucken lassen --
   disp(' ')
   lp = input('Komplette Grafik gleich ausdrucken auf lj5 (0=nein, 1=ja) [0] ? ');
   if(isempty(lp)); lp = 0; end;

   if(lp==1)
       % drucken
       str = sprintf('!lp -dlj5 %s.eps', plotdatei);
      eval(str)
   end

   lp = input('Gezoomte Grafik gleich ausdrucken auf dj16 (0=nein, 1=ja) [0] ? ');
   if(isempty(lp)); lp = 0; end;

   if(lp==1)
```

```
      % drucken
      str = sprintf('!lp -ddj16 %sz.eps', plotdatei);
      eval(str)
   end
else
   if(exist('niceverz')==0)
      % - schoene grafik, nur leistung -
      prpeps(1.5*nice, nice);

      if(str2num(rstart)==-1)
         % nur sammelruecken fuer ainr
         art = 'nusa';
      else
         if(str2num(rstart)==-2)
            % streu- und sammelruecken fuer ainr2
            art = 'sisi';
         else
            art = 'esi';
         end
      end

      str = sprintf('print -depsc %s%s-%s!%s-%spi', ...
         fineverz, art, rueckenbreite, kaiser_beta, anz_pi);

      eval(str)
      disp(['grafik nach ''' str(14:length(str)) ''' gespeichert.'])
   else
      % - schoene grafik, nur leistung, 20x26cm -
      prpeps(20, 26);
      str = sprintf('print -depsc %s%s-%s-%s-%s-%s%s', ...
         niceverz, rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
      eval(str)
      disp(['grafik nach ''' str(14:length(str)) ''' gespeichert.'])
   end
end
```

C.6. diskret.m

```
function out = diskret(in, schwelle);

% funktion zur diskretisierung eines vektors, d.h.
% falls ein wert kleiner als eine bestimmte grenze ist,
% werden mehrere werte zusammengefasst.
%
%
% michael becker, 01.02.2000 - 03.02.2000


TEST = 0;
maxFEHLER = max(in) * 1e-10;

% -- pruefen --

lang = length(in);

if(sign(in(1))~=sign(in(2)))
  % ersten wert auf null setzen
  in(1) = 0;
end

if(sign(in(lang-1))~=sign(in(lang)))
  % letzten wert auf null setzen
  in(lang) = 0;
end
```

```
% -- erforderliche variablen vordefinieren --

if(TEST)
  fid1 = fopen('temp/verwendet', 'w');
  fid2 = fopen('temp/vonbis', 'w');
  fid3 = fopen('temp/gleich', 'w');
end

index = 0;
start = 0;
vz = 0;
ergebnis1 = zeros(lang, 1);
aktuell = 0;
value = 0;
destination = 0;


% -- erster durchlauf --

fprintf('\n----- diskretisierung beginnt -----\n')

% kompletten vektor durchgehen und zusammenfassen, falls noetig.
% dabei wird auf vorzeichenwechsel geachtet. es wird aber nicht
% geprüft, ob alle entstehenden werte groesser als die vorgegebene
% schwelle sind (dies geschieht im zweiten durchlauf).

while(index<=lang)
  while(value==0 & index<lang)
    % index incrementieren
    index = index + 1;

    % aktuellen wert isolieren
    value = in(index);
  end


  if(TEST)
    fprintf(fid1, '%d\n', index);
  end


  if(abs(value)>=schwelle)
    % wert ist groesser als der geforderte mindestwert
    % -> er kann sofort eingetragen werden
    destination = index;


    if(TEST)
      fprintf(fid3, '%d\n', index);
    end


  else
    % wir sollten lieber aufsummieren, damit wir das ziel erreichen
    vz = sign(value);
    start = index;
    index = index + 1;

    % wir gehen jeweils einen index weiter und prüfen,
    % ob wir die schwelle endlich überschritten haben
    while(index<=lang)
      % aktuellen wert isolieren
      aktuell = in(index);

      if(vz==sign(aktuell))
   % aufsummieren
   value = value + aktuell;


   if(TEST)
```

```
    fprintf(fid1, '%d\n', index);
  end


  if(abs(value)>=schwelle)
    % genuegend werte aufsummiert
    break
  else
    index = index + 1;
  end
     else
  % vorzeichenwechsel -> aufsummieren beenden
  index = index - 1;
  break
     end
   end

   % wir haben genügend werte aufsummiert oder wurden aufgrund
   % eines vorzeichenwechsels oder datenendes rausgekickt
   destination = (start + index) / 2;


   if(TEST)
     fprintf(fid2, '%d\n%d\n', start, index);
   end


   if(destination/2~=fix(destination/2))
     % ungerader index steht uns bevor

     if(index<lang/2)
  % links von der mitte -> wir runden auf
  destination = ceil(destination);
     else
  % rechts von der mitte -> wir runden ab
  destination = floor(destination);
     end
    end
 end

 % value an der destination. stelle eintragen
 % und index incrementieren
 ergebnis1(destination) = value;
 value = 0;

 if(destination==lang)
   break
 end
end


if(TEST)
 fclose(fid1);
 fclose(fid2);
 fclose(fid3);
end


% -- pruefen --
summe_alt = sum(in);
summe_neu = sum(ergebnis1);
differenz = summe_alt - summe_neu;
fprintf('\n1. durchlauf beendet  (gesamtfehler: %g)\n', differenz)

if(differenz<maxFEHLER)
 disp('glueck gehabt - es passt noch.')
else
 disp('tja micha, programmieren ist nicht einfach...')
 fprintf('du solltest noch etwas ueben!  (maxfehler: %g)\n', maxFEHLER)
%  return
```

```
end


% -- zweiter durchlauf --

% wir prüfen, ob noch werte vorhanden sind, welche kleiner als
% der erforderliche schwellwert sind. das kann z.b. aufgrund
% eines vorzeichenwechsels oder des datenendes passieren.
index = 1;
lastindex = 0;
lastvalue = 0;
vz = 0;
ergebnis2 = zeros(lang, 1);
first = 1;


if(TEST)
  fid1 = fopen('temp/usedby1', 'w');
  fid2 = fopen('temp/usedby2', 'w');
end


while(index<=lang)
  % aktuellen wert isolieren
  aktuell = ergebnis1(index);
  destination = 0;
  zusammengefasst = 0;

  if(aktuell~=0)


    if(TEST)
      fprintf(fid1, '%d\n', index);
    end


    % wir haben einen relevanten wert gefunden
    if(abs(aktuell)<schwelle | (first~=1 & ...
     abs(lastvalue)<schwelle))


      if(TEST)
   disp(sprintf('\nwert %d oder %d ist zu klein', lastindex, index));
      end


      % da ist ein wert, der noch zu klein ist
      if(sign(aktuell)==vz & first~=1)
   % das vorzeichen passt
   % -> wir addieren zum vorhergehenden wert hinzu
   value = lastvalue + aktuell;


   if(TEST)
     disp(sprintf('fasse %d und %d zusammen', lastindex, index));
   end


   destination = (lastindex + index) / 2;
   zusammengefasst = 1;
      else
   % das vorzeichen passt nicht
   % -> wir suchen den naechsten wert


   if(TEST)
     disp(sprintf('vz passt nicht bei %d und %d', lastindex, index));
   end
```

```
      end
    else
      % wert so wie er ist uebernehmen
      ergebnis2(index) = aktuell;


      if(TEST)
   fprintf(fid2, '%d\n', index);
      end


    end

    % erster wert bearbeitet
    first = 0;

    if(destination>0)
      % es gibt was einzutragen -> bestimmungsindex pruefen
      if(destination/2~=fix(destination/2))
   % ungerader index steht uns bevor

   if(index<lang/2)
     % links von der mitte -> wir runden auf
     destination = ceil(destination);
   else
     % rechts von der mitte -> wir runden ab
     destination = floor(destination);
   end
      end

      % neue werte setzen und alte leeren
      ergebnis2(destination) = value;
      ergebnis2(lastindex) = 0;
      ergebnis2(index) = 0;


      if(TEST)
   fprintf(fid2, '%d\n', destination);
      end
    end

    % letzten wert merken
    lastindex = index;
    vz = sign(aktuell);
    if(zusammengefasst==1)
      % wir haben zuletzt zwei werte addiert
      lastvalue = value;
    else
      lastvalue = aktuell;
    end
  end

  index = index + 1;
end


if(TEST)
  fclose(fid1);
  fclose(fid2);
end


% -- pruefen --
summe_neu = sum(ergebnis2);
differenz = summe_alt - summe_neu;
fprintf('\n2. durchlauf beendet  (gesamtfehler: %g)\n', differenz)

if(differenz<maxFEHLER)
  disp('glueck gehabt - es passt.')
else
```

```
  disp('tja micha, programmieren ist nicht einfach...')
  fprintf('du solltest noch etwas ueben!  (maxfehler: %g)\n', maxFEHLER)
%  return
end

fprintf('\n----- diskretisierung beendet -----\n')
out = ergebnis2;
```

C.7. four.m

Autoren: Thomas Ahrndt, Thomas Bregenzer

```
%FOUR(t, x)
%   Computing of the fast fourier transformation
%   Usage: X = FOUR(t, x)   or  [f X] = FOUR(t, x)

%------------------------------------------------------------------------
% MatLab - Function:    Fouriertransformation
%                       Zeit -> Spektralbereich
%
% Bearbeiter:    Thomas Ahrndt               22.08.1994
% Modifiziert:   Thomas Bregenzer            11.07.1995
%
% Aufruf:
%   [f,X] = four (t,x)      oder      X = four (t,x)
% Uebergabeparameter:
%   - t      = Zeitachse
%   - x      = Signal im Zeitbereich
% Rueckgabeparameter:
%   - X      = Signal im Spektralbereich
%   - f      = Frequenzachse
%
% Algorithmus:
%   - 1: ggf. durch Entfernen des letzten Elements aus dem Eingangsvektor x
%        einen Vektor mit geradzahliger Anzahl von Elementen erzeugen.
%   - 2: Transformation der Elemente (durch <fftshift>):
%        [ 1 2 3 4 5 6 ] ---> [ 4 5 6 1 2 3 ]
%   - 3: Die eigentliche Fouriertransformation: <fft>
%   - 4: Erneute Transformation der Elemente (durch <fftshift>),
%        wodurch das Ergebnis der FFT in der üblichen zweiseitigen
%        Darstellung erscheint: -f_A/2 ... 0 ... +f_A/2
%   - 5: ggf. durch Anhängen des ersten Elementes des Ausgangsvektors X
%        an diesen wieder einen Vektor mit einer ungeradzahliger
%        Anzahl von Elementen erzeugen
%
% Anmerkungen:
%   - diese Funktion ist getestet für eine ungerade Anzahl der Elemente
%     der Vektoren t und x; sie sollte auch für gerade Anzahl
%     korrekte Werte liefern, was aber nicht gewaehrleistet wird.
%
%   - Allg. Berechnung der FFT:
%     Zeitbereich      Spektralbereich
%          0                0
%
%         dt            1 / (N * dt)
%
%               N/2 - 1
%               -------
%               N * dt
%
%               +- dt/2
%
%
```

```
%               N/2 - 1
%             - -------
%               N * dt
%
%      (N-1) dt          -1 / (N * dt)
%
%
%   - Berechnung der FFT:
%     Zeitsignal : -2  -1   0     1  2
%   - gerade Anzahl Erzeugen und Swappen durch <fftshift>
%     ==>: FFTin :  0   1  2/-2  -1
%     ==>: FFTout:  0   1  2/-2  -1
%   - Swappen durch <fftshift> und evtl. ungerade Anzahl Erzeugen
%     Frequenz...: -2  -1   0     1  2
%
%------------------------------------------------------------------------

function [f,X] = four(t, x, test)
% der Parameter <test> ist nur für die
% Erprobung der Funktion implementiert

if length(t) ~= length(x)
   error('Die Vektoren x und t muessen die gleiche Laenge besitzen');
end

N=length(x);

t = t';
x = x';
t_A = t(2)-t(1);
f_0 = 1 / (2*t_A);
N_2 = fix(length(t) / 2);

if fix(N/2) == (N/2)
   %disp('gerade');
   f = -f_0 + f_0/N_2 : f_0/N_2 : f_0;
   z = x;
else
   %disp('ungerade');
   f = -f_0 : f_0/N_2 : f_0;
   z = x(1:N-1);
end;
f = f';

x = fftshift(z);
X = fft(x);
X = fftshift(X);
X = conj (X);

if fix(N/2) ~= (N/2)
   % ungerade
   X = [X X(1)];
end;
X = X';

%---------------------------------------------------------------------
% nur zum Testen

if (nargin==3)
   %whos
   Xr = real(X);
   Xi = imag(X);

   fig = figure;
   plot(f, Xi)
   hold on
   plot(f, Xr,'r')
   hold off

   disp('weiter mit beliebiger Taste');
```

```
   pause

   close(fig);
end
%-----------------------------------------------------------------------

% falls Aufruf in der Form: X = FOUR(t, x)
if (nargout == 1)
   f = X;
end
```

C.8. fxs.m

```
function [dt, al, f, X, t, x] = fxs(l0, dl, es, sh, anz_pi);

% parameter:
%    l0 ... laserwellenlaenge [m]
%    dl ... wellenlaengenabstand [m]
%    es ... anzahl erweiterungsstuetzstellen
%    sh ... 0 fuer keine ausgabe
%           1 fuer textausgabe
%           2 fuer textausgabe und grafikausgabe
%
%    falls ein viertes argument uebergeben wird,
%    werden die ergebnisse angezeigt und gezeichnet.
%
% rueckgabe:
%    dt ... abstand der zeitlinien
%    al ... anzahl der linien im zeitbereich
%
%
% programm zur gewinnung der si-ortsfunktion fuer den sammelgraben
% aus dem Spektralbereich mit hilfe der fouriertransformation
%
%
%    --------- ---------
%   |         |         |
%   |         |         |
%   |         |         |
%   |         |         |
% ---------------------------------------> freiraumwellenlaenge
%        | = l0    | = l0(nachbarkanal)
%   |<------->| = dl
%        |<------->| = dl
%
%
% lambda = geschwindigkeit / frequenz
% l = c / f  <=>  f = c / l  <=>  df = | - c / l^2 * dl |
% hier: b (banbreite) = c * dl / l^2
%
%
%              |||||||||
%              |||||||||
%             ||||||||||
%             ||||||||||
% ----------------------------------> relative frequenz
%         -b/2=|    |=0  |=+b/2
%                ->||<- = df
%
%
% Michael Becker, 09.11.1999 - 24.02.2000


% -- Anzahl Stuetzstellen im Durchlassbereich = const. --
% hat keinen einfluß auf die anzahl der stuetzstellen im
```

```
% zeitbereich, nur das anhaengen der erweiterungsstellen
% ist dafuer relevant
as = 11;


% -- berechnungen --

% mittenfrequenz
f0 = 3e8 / l0;

% bandbreite im frequenzbereich
b0 = 3e8 * dl / l0^2;

% spezialfall hruschka - wir erhalten effektiv nur die
% halbe bandbreite von der mit welcher wir rechnen,
% aufgrund der doppelten verzoegerungszeit
% -> d.h. hier fuer rechnung bandbreite verdoppeln
b = b0 * 2;


% -- erstellung der rel. frequenzachse --

% abstand der stuetzstellen fuer frequenzachse
df = b / (as - 1);

% erstmal von -b bis +b anlegen (spaltenvektor!)
f = [-b : df : b]';

% erweiterungsstuetzstellen vorn und
% hinten anhaengen
f = [ [-b - es * df : df : -b - df]'; f; ...
      [b + df : df : b + df * es]'];


% -- spektrum erstellen --

X = rect(f, 0, b);


% -- inverse fouriertransformation --

[t_alles, x_alles] = invfour(f, X);
x_alles = real(x_alles);

% indizes des zweiten nullduchganges finden
% fuer's einzoomen (relevanter teil)
%
% si(x) = sin(t * pi / delta_t) / ( t * pi / delta_t); si(0) := 1;
% delta_t = 1 / delta_f; hier: delta_f = b;
%
% zweiter nulldurchgang, wenn t * pi / delta_t = 2 * pi,
% d.h. t = 2 * delta_t;
% anzahl der perioden als parameter: anz_pi

delta_t = 1 / b;
min = 1;
idx_l = 0;
for ii=1:fix(length(t_alles)/2)
abstand = abs(t_alles(ii) + anz_pi * delta_t);
  if abstand <= min
    idx_l = ii;
    min = abstand;
  else
    break;
  end
end

min = 1;
idx_r = 0;
for ii=fix(length(t_alles)/2):length(t_alles)
```

```
abstand = abs(t_alles(ii) - anz_pi * delta_t);
  if abstand <= min
    idx_r = ii;
    min = abstand;
  else
    break;
  end
end

% relevanten teil der zeitfunktion zuweisen
t = t_alles(idx_l:idx_r);
x = x_alles(idx_l:idx_r);

% dt (stuetzstellenabstand zeitbereich)
% und anzahl linien berechnen
dt = t(2) - t(1);
al = length(t);


% -- werte ausgeben, falls erwuenscht --
if(sh>0)
  str = sprintf(['Mittenfrequenz:         f0 = %7.0f THz ' ...
   '(%16.0f Hz )'], f0*1e-12, f0);
  disp(str)
  str = sprintf(['Bandbreite:             b0 = %7.0f GHz ' ...
   '(%16.0f Hz )'], b0*1e-9, b0);
  disp(str)
  str = sprintf(['Bandbreite Rechnung:    b  = %7.0f GHz ' ...
   '(%16.0f Hz )'], b*1e-9, b);
  disp(str)
  str = sprintf(['Abstand (Spektrum):     df = %7.3f GHz ' ...
   '(%16.0f Hz )'], df*1e-9, df);
  disp(str)
  str = sprintf(['Abstand der Zeitlinien: dt = %7.3f ps  ' ...
   '(%5i Stuetzstellen )'], dt*1e15, al);
  disp(str)
end


% -- zeichnen, falls erwuenscht --

if(sh>1)
  figure(1), stairs(x);
end
```

C.9. fxsfind.m

```
% aufsatz fuer fxs.m
%
% finden des richtigen abstandes der zeitlinien
% (und somit auch die grabenlaenge m) fuer die
% si-funktion (gesamtkoeffizienten a des filters)
%
%
% Michael Becker, 09.11.1999 - 04.04.2000


% -- vorarbeiten --
save stack
close all
clear
clc

% relatives verzeichnis
verz = 'm/';
```

```
if(isunix)
   % unix system
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end


% -- ueberschrift --
disp('  Ermittlung der Filter-Koeffizienten')
disp(' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
disp(' ')


% -- eingaben von hand --

% mittenwellenlaenge
l0 = input(['Wellenlaenge des Lasers (l0) in nm [13' ...
      '00] :             ']);
if isempty(l0)
  l0 = 1300;
end
l0 = l0 * 1e-9;

% bandbreite (wellenlaenge)
dl = input(['Wellenlaengenabstand (dl) in nm [1] : ' ...
      '                  ']);
if isempty(dl)
  dl = 1;
end
dl = dl * 1e-9;

% genutzte perioden
disp('Anzahl der genutzten Perioden (einseitig), 1 bedeu-')
anz_pi = input('tet links und rechts bis zum ersten Nulldurchgang [2] : ');
if(isempty(anz_pi))
   anz_pi = 2;
end

% zwischenergebnisse anzeigen?
sh = input(['Zwischenergebnisse anzeigen (0=n, 1=j)' ...
      ' [0] ?            ']);
if isempty(sh)
  sh = 0;
end

% speicherdateiname
sd = input(['Speicherdateinamen selbst angeben (lee' ...
      'r=n) ?            ']);
if isempty(sd)
  sds = 0;
else
  sds = 1;
end

% ende der eingaben
disp(' ')


% -- vorberechnungen --

% rel. brechungsindex
%er = n^2;

% c (im medium) = c0 (im vakuum) / n (brechzahl)
% c (im medium) = c0 (im vakuum) / wurzel(er) (rel. brech.-index)
% wir rechnen aber erstmal im freiraum
c = 3e8;
```

```
% zielabstand berechnen
% zeit fuer durchlaufen einer zelle = abstand der zeitlinien,
% zeit = weg / geschwindigkeit
dt_end = l0 / c;

% ausgabe
%str = sprintf(['Wellenlaenge des Lasers:                ' ...
%      'l0     = %9.0f nm'], l0*1e9);
%disp(str)
%str = sprintf(['Wellenlaengenabstand:                   ' ...
%      'dl     = %9.1f nm'], dl*1e9);
%disp(str)
%str = sprintf(['Brechzahl:                              ' ...
%      'n      = %9.3f m/s'], n);
%disp(str)
%str = sprintf(['Relativer Brechungsindex:               ' ...
%      'er     = %9.3f m/s'], er);
%disp(str)
%str = sprintf(['Ausbreitungsgeschwindigkeit (im Medium): ' ...
%      'c      = %9.0f m/s'], c);
%disp(str)
str = sprintf(['Zielabstand:                        ' ...
      'dt_end = %9.5f ps'], dt_end*1e15);
disp(str)

% ab jetzt ausgaben fuer suche
disp(' ')

% startwerte setzen und dann loslegen
es = 1000;
delta = 500;
[dt, al] = fxs(l0, dl, es, 0, anz_pi);
schritte = 1;
gk = sign(dt_end - dt);
% gk = -1 -> abstand groesser als gewuenscht
% gk = +1 -> abstand kleiner als gewuenscht
if(gk==1)
  % abstand zu klein, d.h. es muss kleiner werden
  delta = - delta;
end

if(sh==1)
  str = sprintf(['%5i Erw.-St.stellen -> %5i Linien und ' ...
   '%9.5f ps Abstand'], es, al, dt*1e15);
  disp(str)
end

while (abs(dt_end - dt)>1e-18)
  if (gk ~= sign(dt_end - dt))
    % wir sind drueber
    delta = - fix(delta / 2);
    if (delta==0); break; end;
  end

  gk = sign(dt_end - dt);
  es = fix(es + delta);
  [dt, al] = fxs(l0, dl, es, 0, anz_pi);
  schritte = schritte + 1;

  if(sh==1)
    str = sprintf(['%5i Erw.-St.stellen -> %5i Linien und ' ...
     '%9.5f ps Abstand'], es, al, dt*1e15);
    disp(str)
  end
end

if(sh==1); disp(' '); end;
str = sprintf(' Ergebnis nach %2i Schritten', schritte);
str = [str, sprintf('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~')];
disp(str)
```

```
[dt, al, f, X, t, x] = fxs(l0, dl, es, 2, anz_pi);
disp(' ')


% -- relevante ergebnisse speichern --
% wunschspektrum: f, X
% ergebnis-si: t, x

if(sds==0)
  speicherdatei = sprintf('%sdaten/fxsfind/%gnm-%gpm-%dpi', ...
      verz, l0*1e9, dl*1e12, anz_pi);
else
  speicherdatei = sprintf('%smyfuncs/daten/%s', verz, sd);
end

str = sprintf('save %s f X t x', speicherdatei);
eval(str)
disp(sprintf('Daten in ''~/%s.mat'' gespeichert.', speicherdatei))


% -- speichern der grafik --
title(speicherdatei)
if(sds==0)
   plotdatei = sprintf('%sdaten/fxsfind/%gnm-%gpm-%dpi', ...
      verz, l0*1e9, dl*1e12, anz_pi);
else
  plotdatei = sprintf('%smyfuncs/plots/%s', verz, sd);
end

prpeps(20, 26);
str = sprintf('print -depsc %s', plotdatei);
eval(str)
disp(sprintf('Grafik in ''~/%s.eps'' gespeichert.', plotdatei))


% -- c++ daten speichern --
mx = num2str(max(x));
mx = mx(3:length(mx));
str = sprintf('%sdaten/%dpi/%d/-/a/%s', ...
   verz, anz_pi, length(x), mx);
disp(' ')
disp(['si-Daten fuer C++ Berechnung nach ''' str ''''])
cb = input('speichern (0=nein,1=ja) [0] ? ');
if(isempty('cb')); cb = 0; end;
if(cb)
  saven(str, x);
  disp('daten fuer c++ berechnung gespeichert')
end


% -- aufraeumen --
clear
load stack
```

C.10. fxsfour.m

```
function [df, f, X] = fxsfour(t, x, es);

% transformiert x(t) in den frequenzbereich
% haengt jedoch vorher es nullen (vorn und
% hinten) an die zeitfunktion an.
%
% gibt auch den abstand der frequenzwerte an
%
%
% Michael Becker, 12.11.1999 - 12.11.1999
```

```
%
% -> m/micha/si/fxsfour.m <-


% -- spektrum errechnen --

% zeitabstand bestimmen
dt = t(2) - t(1);

% nullen anhaengen
t = [ [t(1) - es * dt : dt : t(1) - dt]'; t; ...
      [t(length(t)) + dt : dt : t(length(t)) + dt * es]'];
x = [ zeros(es, 1); x; zeros(es, 1)];

% transformation
[f, X] = four(t, x);

% frequenzabstand bestimmen
df = f(2) - f(1);
```

C.11. fxswin.m

```
% laden der filterkoeffizienten
% (diese wurden mit hilfe von fxs_find.m bestimmt)
%
% hier: anwendung einer fensterfunktion
% und ruecktransformation in den frequenzbereich
% zur ueberpruefung des ergebnisses
%
%
% Michael Becker, 12.11.1999 - 14.03.2000


% -- vorarbeiten --
cd ~
clc
close all
save stack
clear
verz = 'm/';


% -- ueberschrift --
disp('  Window-Funktion ueber die Filterkoeffizientenkurve legen')
disp('     sowie Ueberpruefung der Filterkoeffizienten durch')
disp('   Ruecktransformation in Frequenzbereich und Darstellung')
disp(' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
disp(' ')


% -- eingaben von hand --

% mittenwellenlaenge
l0 = input(['Wellenlaenge des Lasers (l0) in nm [13' ...
      '00] :  ']);
if isempty(l0)
  l0 = 1300;
end
l0 = l0 * 1e-9;

% kaiser beta koeffizient
kb = input(['Kaiserkoeffizient beta [3.4] :        ' ...
      '       ']);
if isempty(kb)
  kb = 3.4;
```

```
end

% zwischenergebnisse anzeigen?
sh = input(['Zwischenergebnisse anzeigen (0=n, 1=j)' ...
      ' [0] ? ']);
if isempty(sh)
  sh = 0;
end

% datendateiname
dd = input(['Datendateinamen selbst angeben (leer=n' ...
      ') ?    ']);
if isempty(dd)
  dds = 0;
else
  dds = 1;
end

% speicherdateiname
sd = input(['Speicherdateinamen selbst angeben (lee' ...
      'r=n) ? ']);
if isempty(sd)
  sds = 0;
else
  sds = 1;
end

if(dds==0 | sds==0)
   % bandbreite (wellenlaenge)
   dl = input(['Wellenlaengenabstand (dl) in nm [1] : ' ...
      '       ']);
   if isempty(dl)
      dl = 1;
   end
   dl = dl * 1e-9;

   % genutzte perioden
   disp('Anzahl der genutzten Perioden (einseitig), 1 bedeu-')
   anz_pi = input('tet links und rechts bis zum ersten Nulldurchgang [2] : ');
   if(isempty(anz_pi))
      anz_pi = 2;
   end
end

% ende der eingaben
disp(' ')


% -- entsprechende si-daten laden --
if(dds==0)
  datendatei = sprintf('%sdaten/fxsfind/%gnm-%gpm-%dpi', ...
      verz, l0*1e9, dl*1e12, anz_pi);
else
  datendatei = sprintf('%smyfuncs/daten/%s', ...
      verz, dd);
end

str = sprintf('load %s', datendatei);
eval(str)
disp(sprintf('Daten aus Datei ''~/%s.mat'' geladen.', datendatei))

df_end = f(2) - f(1);
f_ori = f;
X_ori = X;


% -- plotdatei fuer entstehende grafik bestimmen --
if(sds==0)
  plotdatei = sprintf('%sdaten/fxswin/%gnm-%gpm-%dpi-%dkai', ...
      verz, l0*1e9, dl*1e12, anz_pi, kb*10);
```

```
else
  plotdatei = sprintf('%smyfuncs/plots/%s', ...
      verz, sd);
end


% -- rechnen --

% der si-fkt das kaiserfenster ueberbuegeln
w = kaiser(length(x), kb);
xw = x .* w;

% gewichtete funktion zeichnen
subplot(221), stairs(xw)
title(['aus ' datendatei])
xlabel('Grabenzelle')
ylabel('Koeffizient a=aS*aE (kaisergewichtet)')


% -- spektrum errechnen (noch relative frequenzachse) --
% dabei alten frequenzabstand wieder einstellen

% startwerte setzen und dann loslegen
es = length(x);
delta = 500;
[df] = fxsfour(t, xw, es);
schritte = 1;
gk = sign(df_end - df);
% gk = -1 -> abstand groesser als gewuenscht
% gk = +1 -> abstand kleiner als gewuenscht
if(gk==1)
  % abstand zu klein, d.h. es muss kleiner werden
  delta = - delta;
end

if(sh==1)
  str = sprintf(['%5i Erw.-St.stellen -> %g' ...
   ' Abstand'], es, df);
  disp(str)
end

while (abs(df_end - df)>1e6)
  if (gk ~= sign(df_end - df))
    % wir sind drueber
    delta = - fix(delta / 2);
    if (delta==0); break; end;
  end

  gk = sign(df_end - df);
  es = fix(es + delta);
  [df] = fxsfour(t, xw, es);
  schritte = schritte + 1;

  if(sh==1)
    str = sprintf(['%5i Erw.-St.stellen -> %g' ...
     ' Abstand'], es, df);
    disp(str)
  end
end

% geschafft
[df, f_alles, X_alles] = fxsfour(t, xw, es);

% relevanten spektrum-ausschnitt suchen und plotten

% nur werte nehmen ab da, wo
% |real(X_alles(i))| erstmals > schwelle ist

schwelle = 1e-3;
for ii=1:fix(length(X_alles)/2)
```

```
  if(abs(real(X_alles(ii))) > schwelle)
    idx_l = ii;
    break;
  end
end

idx_0 = find(f_alles==0);
idx_r = idx_l + 2 * (idx_0 - idx_l);

f = f_alles(idx_l:idx_r);
X = real(X_alles(idx_l:idx_r));

% zeichnen
subplot(222), plot(f, X)
title('Relatives Spektrum')
xlabel('relative Frequenz zur Laserfrequenz')
ylabel('Amplitude')

% entsprechende werte im originalspektrum finden
% und in die gleiche grafik zeichnen (relative frequenz)
idx_0 = find(f_ori==0);
idx_l = idx_0 - (length(f)-1)/2;
idx_r = idx_l + length(f) - 1;

%hold on;
%stairs(f_ori(idx_l:idx_r), X_ori(idx_l:idx_r), 'r')
%hold off;


% -- errechnen des absoluten spektrums fuer wellenlaengen --

% lambda-achse erstellen in nanometer
%                       | dieses durch zwei kommt von der
%                       | bandbreitenverringerung nach hruschka
l = (- l0^2 / 3e8 .* f / 2 + l0) * 1e9;


% absolute leistung zeichnen
P = X.*X;
subplot(223), plot(l, P)
title(time)
xlabel('Freiraumwellenlaenge in nm')
ylabel('Absolute Leistung P')

% leistung logarithmisch zeichnen
Plog = 10 * log10(P);
subplot(224), plot(l, Plog)
title(plotdatei)
xlabel('Freiraumwellenlaenge in nm')
ylabel('Leistung in dB')


% -- diverse brechnungen zu den erhaltenen grafen --

% abfall der mitteneinsattlung
%me = X(find(f==0));
%me = -20 * log10(me);
%str = sprintf('Mitteneinsattlung: %4.3f dB', me);
%disp(str)

% ueberschwinger
%ue = max(X(find(X>1)));
%ue = 20 * log10(ue);
%str = sprintf('Ueberschwinger:    %4.3f dB', ue);
%disp(str)


% -- relevante ergebnisse speichern --
% gewichtete si-funktion xw
```

```
if(sds==0)
  speicherdatei = sprintf('%sdaten/fxswin/%gnm-%gpm-%dpi-%dkai', ...
      verz, l0*1e9, dl*1e12, anz_pi, kb*10);
  str = sprintf('save %s xw', speicherdatei);
else
  speicherdatei = sprintf('%smyfuncs/daten/%s', ...
      verz, sd);

  num = (1:length(xw))';
  Datarve = [num, xw];
  str = sprintf('save %s Datarve', speicherdatei);
end

eval(str)
disp(sprintf('Daten in ''~/%s.mat'' gespeichert.', speicherdatei))


% -- speichern der grafik --
prpeps(20, 26);
str = sprintf('print -depsc %s', plotdatei);
eval(str)
disp(sprintf('Grafik in ''~/%s.eps'' gespeichert.', plotdatei))


% -- c++ daten speichern --
mx = num2str(max(xw));
mx = mx(3:length(mx));
str = sprintf('%sdaten/%dpi/%d/%d/a/%s', ...
   verz, anz_pi, length(xw), kb*10, mx);
disp(' ')
disp(['si-Daten fuer C++ Berechnung nach ''' str ''''])
cb = input('speichern (0=nein,1=ja) [0] ? ');
if(isempty('cb')); cb = 0; end;
if(cb)
  saven(str, xw);
  disp('daten fuer c++ berechnung gespeichert')
end

% -- aufraeumen --
clear
load stack
```

C.12. gb2r.m

```
function [rsammel, rstreu] = ...
   gb2r(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);

% funktion zur rueckgewinnung der reflexionsfaktoren
% aus den diskretisierten gitterbreiten.
%
% function [rsammel, rstreu] = ...
%    gb2r(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   rstart          der startwert des reflexionsfaktors des streuurueckens,
%                   z.b. '00985' fuer 0.00985 = 0.985% oder
%                   '-1' = nur berechnung des sammelrueckens oder
%                   '-2' = berechnung streu und sammel fuer ainr2
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
```

```
%
%   rsammel         die errechneten reflexionsfaktoren fuer den sammelruecken
%   rstreu          die errechneten reflexionsfaktoren fuer den streuruecken
%
%
% michael becker, 03.02.2000 - 20.03.2000


% -- vorarbeiten --
clc

% verzeichnis fuer unix
verz = 'm/daten/';

if(isunix)
   % unix system
   close all
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end

gemeinsam = sprintf('%s%spi/%s/', verz, anz_pi, rueckenbreite);


% -- dateinamen vorbereiten --
if(str2num(rstart)==-1)
   % nur sammelruecken von ainr
   datendatei2 = sprintf('%s%s/sammel/gb/%s%sd', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei2 = sprintf('%s%s/sammel/r/%s%sgbd', ...
      gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)==-2)
   % streu- und sammelruecken von ainr2
   datendatei1 = sprintf('%s%s/streu/gb2/%s%sd', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei1 = sprintf('%s%s/streu/r2/%s%sgbd', ...
      gemeinsam, kaiser_beta, si_max, lq);
   datendatei2 = sprintf('%s%s/sammel/gb2/%s%sd', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei2 = sprintf('%s%s/sammel/r2/%s%sgbd', ...
      gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
   % streu- und sammelruecken unterschiedlich gewichtet
   datendatei1 = sprintf('%sstreu/gb/%sd', gemeinsam, rstart);
   speicherdatei1 = sprintf('%sstreu/r/%sgbd', gemeinsam, rstart);
   datendatei2 = sprintf('%s%s/sammel/gb/%s%sd', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei2 = sprintf('%s%s/sammel/r/%s%sgbd', ...
      gemeinsam, kaiser_beta, si_max, lq);
end


% -- laden und zuweisen --

if(str2num(rstart)~=-1)
   % -- laden, streu --
   GB1 = laden(datendatei1);
   disp(['gitterbreiten streu aus ''' datendatei1 ''' geladen.'])
end

% -- laden, sammel --
GB2 = laden(datendatei2);
disp(['gitterbreiten sammel aus ''' datendatei2 ''' geladen.'])
```

```
% -- konstanten und vorbereitende rechnungen --
fprintf('\nberechne')

% brechungsindex indium-phosphid
nInP = 3.2;

% brechungsindex indium-gallium-arsenid-phosphid
nInGaAsP = 3.456;

% einfallswinkel
phiE = 45 * pi / 180;

% ausfallswinkel
phiA = asin(nInGaAsP / nInP * sin(phiE));

% reflexionsfaktor des uebergangs
help1 = cos(phiE) / cos(phiA);
help2 = nInP / nInGaAsP;
rueb = abs((help1 - help2) / (help1 + help2));

% fertig mit vorbereiten
fprintf('.')


% -- berechnung confinementfaktor (in %) --

a = [ -2.319e-4;
  1.941e-3;
  -1.072e-6;
  7.82e-10;
  -3.766e-13;
  9.348e-17;
  -9.138e-21];

if(str2num(rstart)~=-1)
   % streu
   G1 = zeros(length(GB1), 1);
   for ii=1:length(GB1)
     help = zeros(length(a), 1);

     for jj=1:length(a)
       help(jj) = a(jj) * abs(GB1(ii)) ^ (jj-1);
     end

     G1(ii) = sum(help);
   end

   fprintf('.')
end

% sammel
G2 = zeros(length(GB2), 1);
for ii=1:length(GB2)
  help = zeros(length(a), 1);

  for jj=1:length(a)
    help(jj) = a(jj) * abs(GB2(ii)) ^ (jj-1);
  end

  G2(ii) = sum(help);
end

fprintf('.')


% -- berechnung der reflexionsfaktoren --

if(str2num(rstart)~=-1)
   % streu
```

```
   rstreu = sign(GB1) .* sqrt(G1 * rueb * rueb / 25);
   fprintf('.')
end

% sammel
rsammel = sign(GB2) .* sqrt(G2 * rueb * rueb / 25);
fprintf(' fertig\n')


% -- zeichnen --

fprintf('zeichne')

subplot(311)
%sammel
plot((1:length(GB2)), GB2/1000, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(GB1)), GB1/1000, 'r')
   hold off
   fprintf('.')
end
ylabel('Gitterbreite [mikrometer]');
title(['Gitterbreite -> Gamma -> Reflexionsfaktor (' time ')'])

fprintf('.')

subplot(312)
% sammel
plot((1:length(GB2)), G2, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(GB1)), G1, 'r')
   hold off
   fprintf('.')
end
ylabel('Confinementfaktor [%]');

fprintf('.')

subplot(313)
% sammel
plot((1:length(GB2)), 100*rsammel, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(GB1)), 100*rstreu, 'r')
   hold off
   fprintf('.')
end
ylabel('Reflexionsfaktor [%]');

fprintf(' fertig\n\n')


% -- speichern --

if(str2num(rstart)~=-1)
   % streu
   saven(speicherdatei1, rstreu);
   disp(['reflexionsfaktoren streu in ''' speicherdatei1 ''' gespeichert.'])
end

% sammel
saven(speicherdatei2, rsammel);
disp(['reflexionsfaktoren sammel in ''' speicherdatei2 ''' gespeichert.'])
```

C.13. hori_l.m

```
function [lambdas, aushori] = ...
   hori_l(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);

% darstellung der vertikalen ausgangaenge des streurueckens
% in abhaengigkeit von der frequenz/wellenlaenge des einfallenden
% lichts und der zellen-nummer fuer eine bestimmte laserfrequenz
% bei schraeger berechnung der einzelnen graeben
%
% function [lambdas, aushori] = ...
%    hori_l(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   rstart          der startwert des reflexionsfaktors des streurueckens,
%                   z.b. '00985' fuer 0.00985 = 0.985% oder
%                   '-1' = nur berechnung des sammelrueckens oder
%                   '-2' = berechnung streu und sammel fuer ainr2
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
%
%   lambdas         der vektor fuer die berechneten wellenlaengen
%   aushori         die matrix der einzelnen ausgaenge. eine zeile
%                   enthaelt die komplexen werte fuer die entsprechende
%                   wellenlaenge im lambdas-vektor, wobei die spalte
%                   die entsprechende zelle (von links nach rechts) angibt.
%
%
% michael becker, 27.04.2000 - 27.04.2000


% -- vorarbeiten --
clc
close all

% relatives verzeichnis
verz = 'm/daten/';
if(isunix)
   % unix system
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end

gemeinsam = sprintf('%s%spi/%s/', verz, anz_pi, rueckenbreite);
l0 = 1300e-9;


% -- dateinamen vorbereiten --
if(str2num(rstart)==-1)
   % nur sammelruecken fuer ainr
   quelle_ergebnis = ...
      sprintf('%s%s/e/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   plotdatei = ...
      sprintf('%s%s/e/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)==-2)
   % streu- und sammelruecken fuer ainr2
   quelle_ergebnis = ...
      sprintf('%s%s/e2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   plotdatei = ...
```

```
      sprintf('%s%s/e2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
   % streu- und sammelruecken unterschiedlich gewichtet
   quelle_ergebnis = ...
      sprintf('%s%s/e/%s%s-%s', gemeinsam, kaiser_beta, si_max, lq, rstart);
   plotdatei = ...
      sprintf('%s%s/e/%s%s-%s', gemeinsam, kaiser_beta, si_max, lq, rstart);
end


% -- ueberschrift --
disp('     Darstellung der Ausgangsleistung des Streugrabens in')
disp('  Abhaengigkeit von der Wellenlaenge des einfallenden Lichts')
disp('           fuer die einzelnen Zellen (Spalten)')
disp(' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
disp(' ')


% -- ergebnisse laden --
fprintf('lade ergebnisse aus ''%s''...', quelle_ergebnis)
data = laden(quelle_ergebnis);
fprintf(' ok\n')


% -- zuweisen der ergebnis-daten --
fprintf('weise ergebnisse in matrix zu...')

data_laenge = length(data);
P = data(data_laenge);
M = str2num(rueckenbreite);
schritte = (data_laenge - 1) / (1 + 2 * (M + P - 1));
aushori = zeros(schritte, M + P - 1);
lambdas = data(data_laenge-schritte:data_laenge-1);

for ii = 1:schritte
   % index des allerersten wertes fuer diese wellenlaenge bestimmen
   % ( zwei vorher )
   idx0 = (ii - 1) * 2 * (M + P - 1) - 1;

   for jj = 1:M+P-1
      % index des realteilwertes bestimmen
      idx_re = idx0 + 2 * jj;
      idx_im = idx_re + 1;

      % wert in matrix eintragen
      aushori(ii, jj) = data(idx_re) + i * data(idx_im);
   end

   % pruefen des auslesens
%   fprintf('%fnm: (last real idx) = %d, (last imag idx) = %d\n', ...
%      lambdas(ii), idx_re, idx_im)
end

fprintf(' ok\n')


% -- zeichnen --
fprintf('zeichne bildchen...')

% - erstmal nur fuer die mittenfrequenz -
% betrag:
subplot(411), plot((1:M+P-1), abs(aushori(fix((schritte+1)/2),:)))
ax = axis;
axis([1 M+P-1 ax(3) ax(4)])

% titel basteln und setzen
str = 'horizontale ausgaenge (mittenfrequenz)';
str = [str sprintf(' M=%s kai=%3.1f simax=0.%s %spi', ...
```

```
   rueckenbreite, str2num(kaiser_beta)/10, si_max, anz_pi)];
if(str2num(rstart)==-1)
   % nur sammelruecken fuer ainr
   str = [str ' nursammel'];
end
if(str2num(rstart)==-2)
   % streu- und sammelruecken fuer ainr2
   str = [str ' ainr2'];
end
if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
   % streu- und sammelruecken unterschiedlich gewichtet
   str = sprintf('%s rstart=0.%s', str, rstart);
end
if(lq(1)=='l')
   str = [str ' linear'];
end
if(lq(1)=='q')
   str = [str ' quadratisch'];
end
title(str)

% phase:
subplot(412), plot((1:M+P-1), phase(aushori(fix((schritte+1)/2),:)))
ax = axis;
axis([1 M+P-1 ax(3) ax(4)])

% - jetzt alle frequenzen -
% betrag:
subplot(413), mesh(abs(aushori))
ax = axis;
axis([0 M+P-1 1 schritte ax(5) ax(6)])

% phase:
subplot(414), mesh(phase(aushori))
ax = axis;
axis([0 M+P-1 1 schritte ax(5) ax(6)])

fprintf(' ok\n')


% -- ausdrucken ? --
disp(' ')
lp = input('Grafik ausdrucken auf dj16 (0=nein, 1=ja) [0] ? ');
if(isempty(lp)); lp = 0; end;

if(lp==1)
   % speichern
   prpeps(20, 26);
   str = sprintf('print -depsc %s', plotdatei);
   eval(str)

   % drucken
   str = sprintf('!lp -ddj16 %s.eps', plotdatei);
   eval(str)
end
```

C.14. invfour.m

Autoren: Thomas Ahrndt, Thomas Bregenzer

```
%INVFOUR(f, X)
%   Computing of the inverse fast fourier transformation
%   Usage: x = INVFOUR(f, X)   or  [t x] = INVFOUR(f, X)

%------------------------------------------------------------------------
```

```
% MatLab - Function:    Fouriertransformation
%                       Spektral --> Zeitbereich
%
% Bearbeiter:    Thomas Ahrndt               22.08.1994
% Modifiziert:   Thomas Bregenzer            02.02.1995
%
% Aufruf:
%   [t,x] = invfour (f,X)      oder      x = invfour (f,X)
% Uebergabeparameter:
%   - f      = Frequenzachse
%   - X      = Signal im Spektralbereich
%   - test      = !!! Nur für die Erprobung der Funktion !!!
% Rueckgabeparameter:
%   - t      = Zeitachse
%   - x      = Signal im Zeitbereich
%
% Algorithmus:
%   - 1: ggf. durch Entfernen des letzten Elements aus dem Eingangsvektor X
%        einen Vektor mit geradzahliger Anzahl von Elementen erzeugen.
%   - 2: Transformation der Elemente (durch <fftshift>):
%        [ 1 2 3 4 5 6 ] ---> [ 4 5 6 1 2 3 ]
%   - 3: Die eigentliche inverse Fouriertransformation: <ifft>
%   - 4: Erneute Transformation der Elemente (durch <fftshift>),
%        wodurch das Ergebnis der FFT in der üblichen zweiseitigen
%        Darstellung erscheint: -t_0 ... 0 ... +t_0
%   - 5: ggf. durch Anhängen des ersten Elementes des Ausgangsvektors x
%        an diesen wieder einen Vektor mit einer ungeradzahliger
%        Anzahl von Elementen erzeugen
%
% Anmerkungen:
%   - diese Funktion ist getestet für eine ungerade Anzahl der Elemente
%     der Vektoren t und x; sie sollte auch für gerade Anzahl
%     korrekte Werte liefern, was aber nicht gewaehrleistet wird.
%
%------------------------------------------------------------------------
function [t,x] = invfour(f, X, test)
% der Parameter <test> ist nur für die
% Erprobung der Funktion implementiert

if length(f) ~= length(X)
   error('Die Vektoren X und f muessen die gleiche Laenge besitzen');
end

% ----------------------------------
N=length(X);

f = f';
X = X';
f_A = f(2)-f(1);
t_A = 1 / (f_A * N);
N_2 = fix(length(f) / 2);

if N_2 == (N/2)
   %disp('gerade');
   t = -N_2*t_A : t_A : N_2*t_A - t_A;
   Z = X;
else
   %disp('ungerade');
   t = -N_2*t_A : t_A : N_2*t_A;
   Z = X(1:N-1);
end
t = t';

X = fftshift(Z);
x = ifft(X);
%x = real(x);
x = fftshift(x);

if fix(N/2) ~= (N/2)
   % ungerade
```

```
   x = [x x(1)];
end
x = x';

%-----------------------------------------------------------------------
% nur zum Testen

if (nargin==3)
   whos
   fig = figure;
   plot(t, x)

   disp('weiter mit beliebiger Taste');
   pause

   close(fig);
end
%-----------------------------------------------------------------------

% falls Aufruf in der Form: x = INVFOUR(f, X)
if (nargout == 1)
   t = x;
end
```

C.15. laden.m

```
function data = laden(datei);

% programm zum laden der werte eines vektors
% (entweder einzeilig oder einspaltig!) aus einer datei,
% und zuweisen an die variable
%
% die elemente des vektors "data" wurden in der datei
% "datei" im ascii-format gespeichert. dabei wird pro
% zeile genau ein wert gelesen.
%
%
% Michael Becker, 11.02.2000 - 02.03.2000


fid = fopen(datei, 'r');
data = fscanf(fid, '%f');
fclose(fid);
```

C.16. r2gbwb.m

```
function [GB2, WB2, GB1, WB1] = ...
   r2gbwb(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);

% funktion zur berechnung der gitterbreiten GB sowie der
% weiten der korrekturzone WB ueber den confinementfaktor
% aus den reflexionsfaktoren eines grabens.
%
% function [GB2, WB2, GB1, WB1] = ...
%    r2gbwb(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   rstart          der startwert des reflexionsfaktors des streurueckens,
%                   z.b. '00985' fuer 0.00985 = 0.985% oder
```

```
%                   '-1' = nur berechnung des sammelrueckens oder
%                   '-2' = berechnung streu und sammel fuer ainr2
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
%
%   GB2             die berechneten gitterbreiten fuer den sammelruecken
%   WB2             die errechneten weiten der korrekturzone fuer den sammelruecken
%   GB1             die berechneten gitterbreiten fuer den streuruecken
%   WB1             die errechneten weiten der korrekturzone fuer den streuruecken
%
%
% michael becker, 02.02.2000 - 20.03.2000


% -- vorarbeiten --
clc

% verzeichnis fuer unix
verz = 'm/daten/';

if(isunix)
   % unix system
   close all
   cd ~
else
   % verzeichnis fuer windows
   verz = ['h:/' verz];
end

gemeinsam = sprintf('%s%spi/%s/', verz, anz_pi, rueckenbreite);


% -- anfang --

r_dis = input('Liegen die Reflexionsfaktoren diskretisiert vor [enter=nein] ? ');

if(isempty(r_dis))
   r_dis = 0;

   gb_dis = input('GB diskretisieren [enter=nein] ? ');
   if(isempty(gb_dis))
      gb_dis = 0;
   else
      gb_dis = 1;

      schwelle = input('Schwelle in nm [387] : ');
      if(isempty(schwelle))
         schwelle = 387;
      end
   end
else
   r_dis = 1;
   gb_dis = 0;
end


% -- dateinamen vorbereiten --
if(str2num(rstart)==-1)
   % nur sammelruecken von ainr
   datendatei2 = ...
      sprintf('%s%s/sammel/r/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiGB2 = ...
      sprintf('%s%s/sammel/gb/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiWB2 = ...
      sprintf('%s%s/sammel/wb/%s%s', gemeinsam, kaiser_beta, si_max, lq);
```

```
end

if(str2num(rstart)==-2)
   % streu- und sammelruecken von ainr2
   datendatei1 = ...
      sprintf('%s%s/streu/r2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiGB1 = ...
      sprintf('%s%s/streu/gb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiWB1 = ...
      sprintf('%s%s/streu/wb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   datendatei2 = ...
      sprintf('%s%s/sammel/r2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiGB2 = ...
      sprintf('%s%s/sammel/gb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiWB2 = ...
      sprintf('%s%s/sammel/wb2/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end

if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
   % streu- und sammelruecken unterschiedlich gewichtet
   datendatei1 = ...
      sprintf('%sstreu/r/%s', gemeinsam, rstart);
   speicherdateiGB1 = ...
      sprintf('%sstreu/gb/%s', gemeinsam, rstart);
   speicherdateiWB1 = ...
      sprintf('%sstreu/wb/%s', gemeinsam, rstart);
   datendatei2 = ...
        sprintf('%s%s/sammel/r/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiGB2 = ...
        sprintf('%s%s/sammel/gb/%s%s', gemeinsam, kaiser_beta, si_max, lq);
   speicherdateiWB2 = ...
        sprintf('%s%s/sammel/wb/%s%s', gemeinsam, kaiser_beta, si_max, lq);
end


% diskrete reflexionsfaktoren richtig laden
% und daraus resultierende ergebnisse richtig speichern
if(r_dis)
   if(str2num(rstart)~=-1)
      % streu
      datendatei1 = [datendatei1 'd'];
      speicherdateiGB1 = [speicherdateiGB1 'rd'];
      speicherdateiWB1 = [speicherdateiWB1 'rd'];
   end

   % sammel
   datendatei2 = [datendatei2 'd'];
   speicherdateiGB2 = [speicherdateiGB2 'rd'];
   speicherdateiWB2 = [speicherdateiWB2 'rd'];
end


% wenn gittereiten diskretisiert werden sollen
% speicherdateinamen richtig setzen
if(gb_dis)
   if(str2num(rstart)~=-1)
      % streu
      speicherdateiGB1 = [speicherdateiGB1 'd'];
      speicherdateiWB1 = [speicherdateiWB1 'd'];
   end

   % werte sind diskretisiert
   speicherdateiGB2 = [speicherdateiGB2 'd'];
   speicherdateiWB2 = [speicherdateiWB2 'd'];
end


% dateinamen testen
if(0)
   if(str2num(rstart)~=-1)
```

```
      disp('-- streu --')
      disp(datendatei1)
      disp(speicherdateiGB1)
      disp(speicherdateiWB1)
   end
   disp('-- sammel --')
   disp(datendatei2)
   disp(speicherdateiGB2)
   disp(speicherdateiWB2)
   return
end


% -- laden und zuweisen --

if(str2num(rstart)~=-1)
   % -- laden, streu --
   rstreu = laden(datendatei1);
   disp(['reflexionsfaktoren streu aus ''' datendatei1 ''' geladen.'])
end

% -- laden, sammel --
rsammel = laden(datendatei2);
disp(['reflexionsfaktoren sammel aus ''' datendatei2 ''' geladen.'])


% -- konstanten und vorbereitende rechnungen --

% brechungsindex indium-phosphid
nInP = 3.2;

% brechungsindex indium-gallium-arsenid-phosphid
nInGaAsP = 3.456;

% einfallswinkel
phiE = 45 * pi / 180;

% ausfallswinkel
phiA = asin(nInGaAsP / nInP * sin(phiE));

% reflexionsfaktor des uebergangs
help1 = cos(phiE) / cos(phiA);
help2 = nInP / nInGaAsP;
rueb = abs((help1 - help2) / (help1 + help2));

% fertig mit vorbereiten
fprintf('.')


% -- berechnung confinementfaktor (in %) --

if(str2num(rstart)~=-1)
   % streu
   G1 = 100 * rstreu .* rstreu / (4 * rueb * rueb);
end

% sammel
G2 = 100 * rsammel .* rsammel / (4 * rueb * rueb);

fprintf('.')


% -- berechnung der gitterbreite --

a = [ .1215;
   510.8793;
   157.4392;
   44.8953;
   -55.6924;
   23.8364;
```

```
   -2.8863 ];

if(str2num(rstart)~=-1)
   % streu
   GB1 = zeros(length(rstreu), 1);
   for ii=1:length(rstreu)
      help = zeros(length(a), 1);

      for jj=1:length(a)
         help(jj) = a(jj) * G1(ii) ^ (jj-1);
      end

      GB1(ii) = sign(rstreu(ii)) * sum(help);
   end

   fprintf('.')
end

% sammel
GB2 = zeros(length(rsammel), 1);
for ii=1:length(rsammel)
   help = zeros(length(a), 1);

   for jj=1:length(a)
      help(jj) = a(jj) * G2(ii) ^ (jj-1);
   end

   GB2(ii) = sign(rsammel(ii)) * sum(help);
end

fprintf('.')

if(gb_dis)
   % diskretisieren
   if(str2num(rstart)~=-1)
      % streu
      GB1 = diskret(GB1, schwelle);
     end

   % sammel
   GB2 = diskret(GB2, schwelle);
end

fprintf('.')


% -- berechnung der weite der korrekturzone --

a = [ 1.05e3;
  7.912e-1;
  -3.023e-4;
  3.289e-7;
  -1.792e-10;
  4.841e-14;
  -5.250e-18 ];

if(str2num(rstart)~=-1)
   % streu
   WB1 = zeros(length(rstreu), 1);
   for ii=1:length(rstreu)
      help = zeros(length(a), 1);

      for jj=1:length(a)
         help(jj) = a(jj) * abs(GB1(ii)) ^ (jj-1);
      end

     WB1(ii) = sum(help);
   end
end
```

```
% sammel
WB2 = zeros(length(rsammel), 1);
for ii=1:length(rsammel)
  help = zeros(length(a), 1);

  for jj=1:length(a)
    help(jj) = a(jj) * abs(GB2(ii)) ^ (jj-1);
  end

  WB2(ii) = sum(help);
end

fprintf(' fertig\n')


% -- zeichnen --

fprintf('zeichne')

subplot(411)
% sammel
plot((1:length(rsammel)), 100*rsammel, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(rstreu)), 100*rstreu, 'r')
   hold off
   fprintf('.')
end
ylabel('Reflexionsfaktor [%]');
title(['Reflexionsfaktor -> Gamma -> Gitterbreite -> ' ...
      'Weite InGaAsP (' time ')'])

fprintf('.')

subplot(412)
% sammel
plot((1:length(rsammel)), G2, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(rstreu)), G1, 'r')
   hold off
   fprintf('.')
end
ylabel('Confinementfaktor [%]');

fprintf('.')

subplot(413)
% sammel
plot((1:length(rsammel)), GB2/1000, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(rstreu)), GB1/1000, 'r')
   hold off
   fprintf('.')
end
ylabel('Gitterbreite [mikrometer]');

fprintf('.')

subplot(414)
% sammel
plot((1:length(rsammel)), WB2/1000, 'b')
if(str2num(rstart)~=-1)
   % streu
   hold on
   plot((1:length(rstreu)), WB1/1000, 'r')
```

```
   hold off
   fprintf('.')
end
xlabel('Graben in vertikaler Richtung');
ylabel('Breite InGaAsP [mikrometer]');

fprintf(' fertig\n\n')


% -- speichern --

if(str2num(rstart)~=-1)
   saven(speicherdateiGB1, GB1);
   disp(['gitterbreiten streu in ''' speicherdateiGB1 ''' gespeichert.'])
   saven(speicherdateiWB1, WB1);
   disp(['weiten streu in ''' speicherdateiWB1 ''' gespeichert.'])
end

saven(speicherdateiGB2, GB2);
disp(['gitterbreiten sammel in ''' speicherdateiGB2 ''' gespeichert.'])
saven(speicherdateiWB2, WB2);
disp(['weiten sammel in ''' speicherdateiWB2 ''' gespeichert.'])
```

C.17. rdiskret.m

```
function [rsammel_d, rstreu_d] = ...
   rdiskret(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);

% funktion zur diskretisierung der reflexionsfaktoren.
%
% function [rsammel_d, rstreu_d] = ...
%    rdiskret(rueckenbreite, rstart, anz_pi, kaiser_beta, si_max, lq);
%
%   rueckenbreite   die rueckenbreite in zellen,
%                   z.b. '1000' fuer 1000 zellen
%   rstart          der startwert des reflexionsfaktors des streuruecken,
%                   z.b. '00985' fuer 0.00985 = 0.985% oder
%                   '-1' = nur berechnung des sammelruecken oder
%                   '-2' = berechnung streu und sammel fuer ainr2
%   anz_pi          die anzahl der genutzen si-abschnitte, z.b. '2'
%   kaiser_beta     der kaiserkoeffizient der window funktion,
%                   z.b. '20' fuer 2.0
%   si_max          maximum der zugrunde liegenden si-funktion,
%                   z.b. '02' fuer 0.02
%   lq              'l' = lineare betrachtung der ausgaenge
%                   'q' = quadratische betrachtung der ausgaenge
%
%   rsammel_d       die diskretisierten reflexionsfaktoren fuer den sammelruecken
%   rstreu_d        die diskretisierten reflexionsfaktoren fuer den streuruecken
%
%
% michael becker, 01.02.2000 - 20.03.2000


% -- vorarbeiten --
clc

% verzeichnis fuer unix
verz = 'm/daten/';

if(isunix)
   % unix system
   close all
   cd ~
else
   % verzeichnis fuer windows
```

```
   verz = ['h:/' verz];
end

gemeinsam = sprintf('%s%spi/%s/', verz, anz_pi, rueckenbreite);


% -- dateinamen vorbereiten --
if(str2num(rstart)==-1)
   % nur sammelruecken von ainr
   datendatei2 = sprintf('%s%s/sammel/r/%s%s', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei2 = [datendatei2 'd'];
end

if(str2num(rstart)==-2)
   % streu- und sammelruecken von ainr2
   datendatei1 = sprintf('%s%s/streu/r2/%s%s', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei1 = [datendatei1 'd'];
   datendatei2 = sprintf('%s%s/sammel/r2/%s%s', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei2 = [datendatei2 'd'];
end

if(str2num(rstart)~=-1 & str2num(rstart)~=-2)
   % streu- und sammelruecken unterschiedlich gewichtet
   datendatei1 = sprintf('%sstreu/r/%s', gemeinsam, rstart);
   speicherdatei1 = [datendatei1 'd'];
   datendatei2 = sprintf('%s%s/sammel/r/%s%s', ...
      gemeinsam, kaiser_beta, si_max, lq);
   speicherdatei2 = [datendatei2 'd'];
end


% -- laden und zuweisen --

if(str2num(rstart)~=-1)
   % -- laden, streu --
   rstreu = laden(datendatei1);
   disp(['reflexionsfaktoren streu aus ''' datendatei1 ''' geladen.'])
end

% -- laden, sammel --
rsammel = laden(datendatei2);
disp(['reflexionsfaktoren sammel aus ''' datendatei2 ''' geladen.'])


% -- und los --
if(str2num(rstart)~=-1)
   % streu
   rstreu_d = diskret(rstreu, 0.0133);
end

% sammel
rsammel_d = diskret(rsammel, 0.0133);


% -- speichern --
if(str2num(rstart)~=-1)
   % streu
   saven(speicherdatei1, rstreu_d);
   disp(['reflexionsfaktoren streu in ''' speicherdatei1 ''' gespeichert.'])
end

% sammel
saven(speicherdatei2, rsammel_d);
disp(['reflexionsfaktoren sammel in ''' speicherdatei2 ''' gespeichert.'])
```

C.18. rect.m

Autoren: Thomas Ahrndt, Rolf Matzner

```
%RECT(t, t_0, delta_t)   or  RECT(f, f_0, delta_f)
%   Computing of a rectangle impulse
%   Usage: x = RECT(t, t_0, delta_t)   or   X = RECT(f, f_0, delta_f)

%------------------------------------------------------------------------
% MatLab - Function:    Rechteck-Impuls
%
% 30.01.95/Matzner   Flankenpunkte auf 0.5 gesetzt
%                    (erforderlich fuer korrekte Integration)
% Bearbeiter:   Thomas Ahrndt               12.8.1994
%
% Aufruf:
%   y = rect (t, t_0, delta_t)
% Uebergabeparameter:
%   - t        = Zeitachse
%   - t_0      = Verschiebung
%   - delta_t  = aequivalente Impulsdauer
% Rueckgabeparameter:
%   - y        = Rechteck-Impuls
%------------------------------------------------------------------------

function y = rect (t, t_0, delta_t)

%t_A=t(2)-t(1);

N=length(t);

if (N == 0)
   error('Kein gueltiger Zeitvektor');
end

a=max(find(t <= t_0));
b=max(find(t <= (t_0 + delta_t/2))) - a;
y=zeros(size(t));

if ((a-b) <= 0)
   error('Bereichsueberschreitung');
end

if ((a+b) > N)
   error('Bereichsueberschreitung');
end

if b == 0
   error('delta_t darf nicht 0 sein');
else
   y (a-b : a+b) = ones (1, 2*b+1);
   y (a-b) = 0.5;
   y (a+b) = 0.5;
end
```

C.19. saven.m

```
function saven(datei, data);

% programm zur speicherung der werte eines vektors
% (entweder einzeilig oder einspaltig!) in einer datei,
% so dass diese daten dann von einem anderen programm
% eingelesen werden koennen.
%
```

```
% die elemente des vektors "data" werden in der datei
% "datei" im ascii-format gespeichert. dabei wird pro
% zeile genau ein wert geschrieben.
%
%
% Michael Becker, 20.01.2000 - 20.01.2000


fid = fopen(datei, 'w');

for ii=1:length(data)
  fprintf(fid, '%f\n', data(ii));
end

fclose(fid);
```

C.20. time.m

```
function t = time(art);

% rueckgabewert in abhaengigkeit von art:
%
% art = 0: aktuelles datum und uhrzeit
%          in der form tt.mm.jjjj ss:mm
%
% art = 1: aktuelles datum in der form tt.mm.jjjj
%
% art = 2: aktuelle zeit in der form ss:mm:ss
%
%
% Michael Becker, 18.11.1999 - 09.02.2000


if(exist('art')==0)
   art = 0;
end


cl = fix(clock);

day = num2str(cl(3));
if length(day)==1
  day = ['0' day];
end

month = num2str(cl(2));
if length(month)==1
  month = ['0' month];
end

hours = num2str(cl(4));
if length(hours)==1
  hours = ['0' hours];
end

minutes = num2str(cl(5));
if length(minutes)==1
  minutes = ['0' minutes];
end

seconds = num2str(cl(6));
if length(seconds)==1
  seconds = ['0' seconds];
end
```

```
switch(art)
   case 0
      t = sprintf('%s.%s.%i %s:%s', day, month, cl(1), hours, minutes);
   case 1
      t = sprintf('%s.%s.%i', day, month, cl(1));
   case 2
      t = sprintf('%s:%s:%s', hours, minutes, seconds);
   otherwise
      t = 'falsches argument';
end
```

D. Quellcodes der C-Programme

D.1. acad.cpp

```
/*  programm zur erstellung der skripten fuer das zeichnen der
    masken fuer die filterherstellung mit autocad.

    ------------------------------------------------------------
    |   |       |       |       |       |       |       |   |
    | 0 |   1   |   2   |   3   |   4   |   5   |   6   | 7 |
    |   |       |       |       |       |       |       |   |
    ------------------------------------------------------------

    0 ..... kennzeichnung links (simples dreieck)
    1-6 ... filter 1 bis filter 6
    7 ..... kennzeichnung rechts (nummerierung)

    hinweis: gerechnet wird alles in NANOMETER !

    michael becker, 14.01.2000 - 27.04.2000 */

#include <vcl\condefs.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>

#pragma hdrstop
USERES("acad.res");

//---------------------------------------------------------------------------
// globale variablen

// maximale anzahl filter pro struktur (fuer dateinamenzuweisung wichtig)
const int ANZAHL_MAX = 6;

// maximale anzahl graeben pro ruecken
const int GRABEN_MAX = 8000;

// zeichnungsmassstab
const int M = 1000;

// breite der wellenfuehrung (1.05 mikrometer)
const double welle = 1050;

/*  anzahl der zu zeichnenden rueckenpaare
    (streu + sammel = 1 == ein filter) */
int anzahl;

// breite einer struktur
double breite;

// begrenzungen der struktur
```

```
double hoehe;

// breite einer zelle (materialwellenlaenge)
double lambda;

// abstand der beiden lwl in einer struktur
double lwl_abstand;

// (effektive) breite der seele eines lwl
double lwl_seele;

// verzeichnis fuer ruecken- und zeichnungsdaten
char datverz[80];

// verzeichnis fuer scripte
char scrverz[80];

// minima und maxima fuers zoomen nach gitterzeichnung
double xmin, ymin, xmax, ymax;

// breite der brechstruktur in sio2 (160 mikrometer)
double sio2_breite = 160 * 1000;

// groesse des rechtecks fuer lwl kennzeichnung (100 x 500 mikrometer)
double kenn_breit = 100 * 1000;
double kenn_hoch = 500 * 1000;

// abstand (1 mm) und groesse (10 x 10 mikrometer) fuer layermarkierung
double mark_abstand = 1000 * 1000;
double mark = 10 * 1000;

// laenge der bragg-pruef-gitter [(P=8)*lambda = 3096 nanometer]
double pruef_laenge = 3096;


//--------------------------------------------------------------------------
// definition der genutzen funktionen

// erstellen der filter- bzw. struktur-skripte
void filter();

// erstellen des scripts fuer ein filter
void gitters(int nummer, int nr);

// erstellen der hilfs- und pruefskripte
void grund();

// kennzeichnung der lwl
void kennzeichnung(FILE *datei, double x0, double y0, int lire, int obun);

// ein datenarray einlesen
int liesArray(char *source, double *dest, double *verify, char *ruecken);

// marken fuer justierung der einzelnen layer setzen
void markierung(FILE *datei);

// punkte fuer eine justiermarke schreiben
void marke(FILE *datei, double x0, double y0);

// einen punkt in eine datei schreiben
void schreibePunkt(FILE *datei, double x, double y);

// einen punkt in eine datei schreiben
void schreibePunktNoY(FILE *datei, double x, double y);

// 0 + enter zum fortfahren
void taste();

/*  fehlermeldung fuer unterschiedliche laengen
    in den datendateien fuer ein filter */
```

```
void vl_fehler(char *welche);

// punkte fuer den wellensumpf erstellen
void wellensumpf(FILE *datei, double x0, double y0, double b0);

// fehlermeldung bei ueberschreitung der maximalen grabenlaenge
void zl_fehler();


//------------------------------------------------------------------------
// hauptfunktion zum anzeigen des menues
int main(int argc, char **argv) {

    // wert der menuewahl
    int wahl = -1;

    /*  ein argument, verzeichnis der daten files -
        also das, wo die rueckdaten stehen als
        st1_gb.dat, st1_wb.dat sa1_gb.dat, sa1_wb.dat */
    if(argc==2) {
        // abweichendes verzeichnis angegeben
        // wichtig: kein verzeichnisseparator am ende!
        strcpy(datverz, argv[1]);
    } else {
        // keine angabe -> standardverzeichnis nutzen
        strcpy(datverz, "../dat");
    }
    strcpy(scrverz, datverz);
    strcat(scrverz, "/..");

    while(wahl!=0) {
        clrscr();
        printf("   daten-verzeichnis:  \"%s\"\n", datverz);
        printf("   script-verzeichnis: \"%s\"\n\n\n", scrverz);

        printf("     Optisches Filter auf Halbleiterbasis:\n");
        printf("     Programm zur Erstellung der Scripte fuer ");
        printf("das Zeichnen mit AutoCad14\n");
        printf("    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printf("~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
        printf("                                                  ");
        printf("(c) 2000, Michael Becker\n\n");

        printf("\n   1 ... Skripte fuer Grund- (Hilfsumrandung) ");
        printf("und Pruefstruktur");
        printf("\n   2 ... Skripte fuer einzelne Filter");
        printf("\n   0 ... Ende");

        printf("\n\n   Ihre Wahl: ");
        scanf("%d", &wahl);

        if(wahl==1) grund();
        if(wahl==2) filter();
    }

    return 0;
}


//------------------------------------------------------------------------
/*  laden aktueller daten und veranlassen der
    skripterstellung fuer die einzelnen filter */
void filter() {

    // die struktur nummer
    int nummer = 0;

    // zeiger auf datei
    FILE *datei;
```

```
// hilfsvariablen
int i;
char name[80];
double start, starty, ende;

// bildschirm leeren
clrscr();
printf("\n");

// ----- laden der von grund() gespeicherten aktuellen zeichnungsdaten -----

// dateinamen vorbereiten
strcpy(name, scrverz);
strcat(name, "/zeichng.dat");

// datei oeffnen und anschliessend werte einlesen
datei = fopen(name, "r");

// anzahl der strukturen
fscanf(datei, "%d", &anzahl);

// breite einer struktur
fscanf(datei, "%lf", &breite);

// hoehe einer struktur
fscanf(datei, "%lf", &hoehe);

// abstand der lwl
fscanf(datei, "%lf", &lwl_abstand);

// materialwellenlaenge
fscanf(datei, "%lf", &lambda);

// datei schliessen
fclose(datei);

// ----- filternummer abfragen -----
while(nummer<1 || nummer>5) {
    printf("nummer der strktur : ");
    scanf("%d", &nummer);
}

// ----- erstellung veranlassen -----
for(i=0; i<anzahl; i++)
    gitters(nummer, i);

// ----- script fuer strukturnummer in sio2 schicht -----

// dateinamen vorbereiten
strcpy(name, scrverz);
strcat(name, "/nr");

switch(nummer) {
case 1:
    strcat(name, "1");
    break;
case 2:
    strcat(name, "2");
    break;
case 3:
    strcat(name, "3");
    break;
case 4:
    strcat(name, "4");
    break;
case 5:
    strcat(name, "5");
    break;
}
```

```
strcat(name, ".scr");

// ausgabedatei oeffnen
datei = fopen(name, "w");

// aktuellen layer setzen
fprintf(datei, "-layer SE sio2\n\n");

// farbe setzen (weiss)
fprintf(datei, "farbe 7\n");

// brechhilfe und kennzeichnung der struktur (rechts)
start = breite * (anzahl + .5);

// allgemeiner anfang der brechstruktur
fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f",
    start / M, (-sio2_breite / 2) / M,
    (start + breite / 2 + sio2_breite / 2) / M, (-sio2_breite / 2) / M,
    (start + breite / 2 + sio2_breite / 2) / M, (hoehe / 2) / M,
    (start + breite / 2 - sio2_breite / 2) / M, (hoehe / 2) / M,
    (start + breite / 2 - sio2_breite / 2) / M, (sio2_breite / 2) / M,
    (start + sio2_breite / 2) / M, (sio2_breite / 2) / M,
    (start + sio2_breite / 2) / M, (hoehe - sio2_breite / 2) / M,
    (start + breite / 4) / M, (hoehe - sio2_breite / 2) /M);

// einzelne kennzeichnungen (linke haelfte)
for(i=0; i<nummer; i++) {
    starty = hoehe + hoehe / 20 - (hoehe / 5) * i;

    fprintf(datei, " %f,%f %f,%f %f,%f %f,%f",
        (start + breite / 4) / M, (starty - hoehe / 10) / M,
        (start + hoehe / 10) / M, (starty - hoehe / 10) / M,
        (start + hoehe / 10) / M, (starty - hoehe / 5) / M,
        (start + breite / 4) / M, (starty - hoehe / 5) / M);
}

// einzelne kennzeichnungen (rechte haelfte)
for(i=nummer-1; i>=0; i--) {
    starty = hoehe + hoehe / 20  - (hoehe / 5) * i;

    fprintf(datei, " %f,%f %f,%f %f,%f",
        (start + breite / 2 - hoehe / 10) / M, (starty - hoehe / 5) / M,
        (start + breite / 2 - hoehe / 10) / M, (starty - hoehe / 10) / M,
        (start + breite / 4) / M, (starty - hoehe / 10) / M);

    if(i==0)
        ende = hoehe - sio2_breite / 2;
    else
        ende = starty;

    fprintf(datei, " %f,%f", (start + breite / 4) / M, ende / M);
}

// allgemeines ende der brechstruktur
fprintf(datei, " %f,%f %f,%f %f,%f %f,%f %f,%f s\n",
    (start + breite / 2 - sio2_breite/2) / M, (hoehe - sio2_breite/2) / M,
    (start + breite / 2 - sio2_breite / 2) / M, (hoehe / 2) / M,
    (start + breite / 2 + sio2_breite / 2) / M, (hoehe / 2) / M,
    (start + breite / 2 + sio2_breite/2) / M, (hoehe + sio2_breite/2) / M,
    start / M, (hoehe + sio2_breite / 2) / M);

// ausgabedatei schliessen
fclose(datei);

// information ausgeben
textbackground(2);
cprintf("\nscript fuer nummerierung erstellt.\r\n");
textbackground(0);

// 0 + enter zum fortfahren
```

```
    taste();
}


//------------------------------------------------------------------------
/*  eigentliches erstellen der skripte: lesen der daten fuer den X.
    streuruecken aus "st_gbX.dat" sowie "st_wbX.dat" und
    fuer den sammelruecken aus "sa_gbX.dat" sowie "sa_wbX.dat"
    (jeweils ein wert pro zeile). die anzahlen der werte in den beiden
    dateien muessen uebereinstimmen! */
void gitters(int nummer, int nr) {

    // rueckenlaenge
    int lang;

    // arrays fuer die halbe gitterbreite
    double gb_streu[GRABEN_MAX];
    double gb_sammel[GRABEN_MAX];

    // arrays fuer die halbe breite der korrekturzone
    double wb_streu[GRABEN_MAX];
    double wb_sammel[GRABEN_MAX];

    // allgemeine startwerte der gitter
    double y0;
    double x0_streu;
    double x0_sammel;

    // die sechs punkte eines grabens
    double gr[6][2];

    // erster graben wird gerade gezeichnet bzw. muss noch gezeichnet werden
    int first;

    // index des zuletzt gezeichneten grabens
    int last = 0;

    // streuruecken mit wellensumpf abschliessen?
    bool sumpf;

    // zeiger auf datei
    FILE *datei;
    FILE *datei2;

    // pointer auf dateinamen (lesen)
    char streu_gb[80];
    char streu_wb[80];
    char sammel_gb[80];
    char sammel_wb[80];

    /*  pointer auf dateinamen (schreiben)
        out  = skript fuer gitter
        out2 = skript fuer fuehrung und korrekturzone */
    char out[80];
    char out2[80];

    // hilfsvariablen
    int i;
    double delta, start;

    // minima und maxima fuers zoomen vorbelegen
    xmin = anzahl * breite;
    ymin = hoehe;
    xmax = 0;
    ymax = 0;

    // ----- dateinamen vorbereiten -----

    // input
    strcpy(streu_gb, datverz);
```

```
strcat(streu_gb, "/st");
strcpy(streu_wb, datverz);
strcat(streu_wb, "/st");
strcpy(sammel_gb, datverz);
strcat(sammel_gb, "/sa");
strcpy(sammel_wb, datverz);
strcat(sammel_wb, "/sa");
// output
strcpy(out, scrverz);
strcat(out, "/flt");
strcpy(out2, scrverz);
strcat(out2, "/fhr");

// nummer der struktur in scriptdateinamen bringen
switch(nummer) {
case 1:
    strcat(out, "1");
    strcat(out2, "1");
    break;
case 2:
    strcat(out, "2");
    strcat(out2, "2");
    break;
case 3:
    strcat(out, "3");
    strcat(out2, "3");
    break;
case 4:
    strcat(out, "4");
    strcat(out2, "4");
    break;
case 5:
    strcat(out, "5");
    strcat(out2, "5");
    break;
}

strcat(out, "-");
strcat(out2, "-");

// nummer des filters in namen eintragen
switch(nr) {
case 0:
    // input
    strcat(streu_gb, "1");
    strcat(streu_wb, "1");
    strcat(sammel_gb, "1");
    strcat(sammel_wb, "1");
    // output
    strcat(out, "1");
    strcat(out2, "1");
    break;
case 1:
    // input
    strcat(streu_gb, "2");
    strcat(streu_wb, "2");
    strcat(sammel_gb, "2");
    strcat(sammel_wb, "2");
    // output
    strcat(out, "2");
    strcat(out2, "2");
    break;
case 2:
    // input
    strcat(streu_gb, "3");
    strcat(streu_wb, "3");
    strcat(sammel_gb, "3");
    strcat(sammel_wb, "3");
    // output
    strcat(out, "3");
```

```
    strcat(out2, "3");
    break;
case 3:
    // input
    strcat(streu_gb, "4");
    strcat(streu_wb, "4");
    strcat(sammel_gb, "4");
    strcat(sammel_wb, "4");
    // output
    strcat(out, "4");
    strcat(out2, "4");
    break;
case 4:
    // input
    strcat(streu_gb, "5");
    strcat(streu_wb, "5");
    strcat(sammel_gb, "5");
    strcat(sammel_wb, "5");
    // output
    strcat(out, "5");
    strcat(out2, "5");
    break;
case 5:
    // input
    strcat(streu_gb, "6");
    strcat(streu_wb, "6");
    strcat(sammel_gb, "6");
    strcat(sammel_wb, "6");
    // output
    strcat(out, "6");
    strcat(out2, "6");
    break;
}

// input
strcat(streu_gb, "_gb.dat");
strcat(streu_wb, "_wb.dat");
strcat(sammel_gb, "_gb.dat");
strcat(sammel_wb, "_wb.dat");
// output
strcat(out, ".scr");
strcat(out2, ".scr");

// ----- einlesen der breiten fuer streu ruecken -----
printf("\n  filter nr. %i\n ~~~~~~~~~~~~~~\n", nr+1);

// -- gitterbreiten --
lang = liesArray(streu_gb, gb_streu, NULL, "");

// auf laengenueberschreitung pruefen
if(lang==-1) {
    zl_fehler();
    return;
}

// -- weiten der korrekturzone --
i = liesArray(streu_wb, wb_streu, gb_streu, "streu");

// auf laengenueberschreitung pruefen
if(i==-1) {
    zl_fehler();
    return;
}

// laengen vergleichen (pruefen)
if(i!=lang) {
    vl_fehler(streu_wb);
    return;
}
```

```
// ----- einlesen der breiten fuer sammel ruecken -----

// -- gitterbreiten --
i = liesArray(sammel_gb, gb_sammel, NULL, "");

// auf laengenueberschreitung pruefen
if(i==-1) {
    zl_fehler();
    return;
}

// laengen vergleichen (pruefen)
if(i!=lang) {
    vl_fehler(sammel_gb);
    return;
}

// -- weiten der korrekturzone --
i = liesArray(sammel_wb, wb_sammel, gb_sammel, "sammel");

// auf laengenueberschreitung pruefen
if(i==-1) {
    zl_fehler();
    return;
}

// laengen vergleichen (pruefen)
if(i!=lang) {
    vl_fehler(sammel_wb);
    return;
}

// ----- startwerte des gitters errechnen bzw. abfragen -----

// mittelkoordinate der beiden lwl (in nanometer)
x0_sammel = breite * (nr + 1) - lwl_abstand / 2;
x0_streu  = breite * (nr + 1) + lwl_abstand / 2;

// anzahl der zellen fuer die aktuelle struktur
printf("grabenanzahl: %i zellen\n", lang);

// y-startwert errechnen und aendern lassen, falls erwuenscht
start = ( hoehe - lambda * lang) / 2;
printf("y-startwert (mikrometer) [0->%f] ? ", start/1000);
scanf("%lf", &y0);
y0 = y0 * 1000;
if(y0==0) y0 = start;

// mit wellensumpf abschliessen?
printf("streuruecken mit wellensumpf abschliessen (0=nein, 1=ja) ? ");
scanf("%d", &i);
if(i==1)
    sumpf = true;
else
    sumpf = false;

// ----- berechnung und erstellung der script-dateien -----

// - gitter -
// ausgabedatei oeffnen
datei = fopen(out, "w");

// aktuellen layer setzen
fprintf(datei, "-layer SE gitter\n\n");

// farbe setzen (rot)
fprintf(datei, "farbe 1\n");

/*  am anfang grob zoomen
    1600nm, weil max. gitterbreite = 3 mikrometer */
```

```
fprintf(datei, "zoom F %f,%f %f,%f\n",
    (x0_sammel - 1600) / M, (start - lambda) / M,
    (x0_streu + 1600) / M, (start + (lang + 5) * lambda) / M);

// - fuehrung -
// ausgabedatei oeffnen
datei2 = fopen(out2, "w");

// aktuellen layer setzen
fprintf(datei2, "-layer SE fuehrung\n\n");

// farbe setzen (gruen)
fprintf(datei2, "farbe 3\n");

// -- streu --

// erster graben muss noch gezeichnet werden
first = 1;

// 1. hinweg
for(i=0; i<lang; i++) {
    // schauen, ob die grabenbreite verschieden null ist
    if(gb_streu[i]!=0) {
        // berechnung des startpunktes fuer diesen einzelnen graben
        start = y0 + i * lambda;

        // generierung der notwendigen punkte fuer den i. graben
        gr[0][0] = x0_streu + gb_streu[i];
        gr[0][1] = start - gb_streu[i];

        gr[1][0] = x0_streu + gb_streu[i];
        gr[1][1] = start - gb_streu[i] + lambda / 4;

        gr[2][0] = x0_streu + gb_streu[i];
        gr[2][1] = start - gb_streu[i] + lambda / 2;

        gr[3][0] = x0_streu - gb_streu[i];
        gr[3][1] = start + gb_streu[i] + lambda / 2;

        gr[5][0] = x0_streu - gb_streu[i];
        gr[5][1] = start + gb_streu[i];

        // - graben -
        fprintf(datei, "plinie");
        schreibePunkt(datei, gr[0][0], gr[0][1]);
        schreibePunkt(datei, gr[2][0], gr[2][1]);
        schreibePunkt(datei, gr[3][0], gr[3][1]);
        schreibePunkt(datei, gr[5][0], gr[5][1]);
        fprintf(datei, " s\n");

        // - korrekturzone -
        if(first==1) {
            // anfangsraum (links vom ersten gitter)
            fprintf(datei2, "plinie");
            schreibePunktNoY(datei2, x0_streu + welle / 2, -sio2_breite/2);
            schreibePunkt(datei2, x0_streu + welle / 2, gr[0][1]);
            schreibePunkt(datei2, x0_streu + wb_streu[i], gr[0][1]);

            // anfangsraum ist gezeichnet
            first = 0;
        }

        // korrekturzonenpunkt fuer diesen graben (rechts bzw. unten)
        schreibePunkt(datei2, x0_streu + wb_streu[i], gr[1][1]);

        // aktuellen index merken
        last = i;
    }
}
```

```
// korrekturzone:
if(gb_streu[last]>welle/2) {
    /*  letzter graben ist breiter als wellenfuehrung -> endraum
        (rechts vom letzten gitter), parallel zum letzten gitter */
    schreibePunkt(datei2, x0_streu + wb_streu[last], gr[2][1] + lambda / 2);
    schreibePunkt(datei2, x0_streu + gb_streu[last], gr[2][1] + lambda / 2);
    schreibePunkt(datei2, x0_streu + welle / 2,
        gr[2][1] + lambda / 2 + gb_streu[last] - welle / 2);
} else {
    // letzer graben ist schmaler, deshalb nur geradefuehren.
    schreibePunkt(datei2, x0_streu + welle / 2,
    gr[3][1] + lambda / 2 - gb_streu[last] + welle / 2);
}

if(sumpf) {
    // wir schliessen mit wellensumpf ab
    wellensumpf(datei2, x0_streu, gr[3][1] + lambda / 2 -
        gb_streu[last] + welle / 2, welle);
} else {
    // wellenleiterfuehrung bis zum rand
    schreibePunktNoY(datei2, x0_streu + welle / 2, hoehe + sio2_breite / 2);
    schreibePunktNoY(datei2, x0_streu - welle / 2, hoehe + sio2_breite / 2);
}

schreibePunkt(datei2, x0_streu - welle / 2,
    gr[3][1] + lambda / 2 - gb_streu[last] + welle / 2);

if(gb_streu[last]>welle/2) {
    /*  letzter graben ist breiter als wellenfuehrung -> endraum
        (rechts vom letzten gitter), parallel zum letzten gitter */
    schreibePunkt(datei2, x0_streu - gb_streu[last], gr[3][1] + lambda / 2);
    schreibePunkt(datei2, x0_streu - wb_streu[last], gr[3][1] + lambda / 2);
}

// 2. rueckweg (nur fuer korrekturzone)
for(i=lang-1; i>=0; i--) {
    // schauen, ob die grabenbreite verschieden null ist
    if(gb_streu[i]!=0) {
        // berechnung des startpunktes fuer diesen einzelnen graben
        start = y0 + i * lambda;

        // generierung der notwendigen punkte fuer den i. graben
        gr[0][0] = x0_streu + gb_streu[i];
        gr[0][1] = start - gb_streu[i];

        gr[4][0] = x0_streu - gb_streu[i];
        gr[4][1] = start + gb_streu[i] + lambda / 4;

        // korrekturzonenpunkt fuer diesen graben (links bzw. oben)
        schreibePunkt(datei2, x0_streu - wb_streu[i], gr[4][1]);

        // aktuellen index merken
        last = i;
    }
}

/*  korrekturzone:
    wieder anfangsraum (links vom ersten gitter) und abschluss */
schreibePunkt(datei2, x0_streu - wb_streu[last], gr[0][1]);
schreibePunkt(datei2, x0_streu - welle / 2, gr[0][1]);
schreibePunktNoY(datei2, x0_streu - welle / 2, - sio2_breite / 2);
fprintf(datei2, " s\n");

// -- sammel --

// erster graben muss noch gezeichnet werden
first = 1;

// 1. hinweg
for(i=0; i<lang; i++) {
```

```
        // schauen, ob die grabenbreite verschieden null ist
        if(gb_sammel[i]!=0) {
            /*  checkvariablen fuer verschiebung bei negativen werten
                (um lambda/2 nach links bzw. unten oder
                um lambda/2 nach rechts bzw. oben) */
            if(gb_sammel[i]<0) {
                // wert positiv machen
                gb_sammel[i] = - gb_sammel[i];

                if(i<lang/2) {
                    // verschiebung nach links/unten
                    delta = - lambda / 2;
                } else {
                    // verschiebung nach rechts/oben
                    delta = lambda / 2;
                }
            } else
                delta = 0;

            // berechnung des startpunktes fuer diesen einzelnen graben
            start = y0 + i * lambda + delta;

            // generierung der notwendigen punkte fuer den i. graben
            gr[0][0] = x0_sammel + gb_sammel[i];
            gr[0][1] = start + gb_sammel[i];

            gr[1][0] = x0_sammel + gb_sammel[i];
            gr[1][1] = start + gb_sammel[i] + lambda / 4;

            gr[2][0] = x0_sammel + gb_sammel[i];
            gr[2][1] = start + gb_sammel[i] + lambda / 2;

            gr[3][0] = x0_sammel - gb_sammel[i];
            gr[3][1] = start - gb_sammel[i] + lambda / 2;

            gr[5][0] = x0_sammel - gb_sammel[i];
            gr[5][1] = start - gb_sammel[i];

            if(delta!=0) {
                // gitterbreite wieder negativ machen (fuer rueckweg)
                gb_sammel[i] = - gb_sammel[i];
            }

            // - graben -
            fprintf(datei, "plinie");
            schreibePunkt(datei, gr[0][0], gr[0][1]);
            schreibePunkt(datei, gr[2][0], gr[2][1]);
            schreibePunkt(datei, gr[3][0], gr[3][1]);
            schreibePunkt(datei, gr[5][0], gr[5][1]);
            fprintf(datei, " s\n");

            // - korrekturzone -
            if(first==1) {
                // anfangsraum (links vom ersten gitter)
                fprintf(datei2, "plinie");
                schreibePunktNoY(datei2, x0_sammel + welle / 2, -sio2_breite/2);
                schreibePunkt(datei2, x0_sammel + welle / 2, gr[5][1]);
                schreibePunkt(datei2, x0_sammel + wb_sammel[i], gr[5][1]);

                // anfangsraum ist gezeichnet
                first = 0;
            }

            // korrekturzonenpunkt fuer diesen graben (rechts bzw. unten)
            schreibePunkt(datei2, x0_sammel + wb_sammel[i], gr[1][1]);

            // aktuellen index merken
            last = i;
        }
    }
```

```
    // korrekturzone: endraum (rechts vom letzten gitter)
    schreibePunkt(datei2, x0_sammel + wb_sammel[last], gr[2][1]);
    schreibePunkt(datei2, x0_sammel + welle / 2, gr[2][1]);
    schreibePunktNoY(datei2, x0_sammel + welle / 2, hoehe + sio2_breite / 2);
    schreibePunktNoY(datei2, x0_sammel - welle / 2, hoehe + sio2_breite / 2);
    schreibePunkt(datei2, x0_sammel - welle / 2, gr[2][1]);
    schreibePunkt(datei2, x0_sammel - wb_sammel[last], gr[2][1]);

    // 2. rueckweg (nur fuer korrekturzone)
    for(i=lang-1; i>=0; i--) {
        // schauen, ob die grabenbreite verschieden null ist
        if(gb_sammel[i]!=0) {
            /*  checkvariablen fuer verschiebung bei negativen werten
                (um lambda/2 nach links bzw. unten oder
                um lambda/2 nach rechts bzw. oben) */
            if(gb_sammel[i]<0) {
                // wert positiv machen
                gb_sammel[i] = - gb_sammel[i];

                if(i<lang/2) {
                    // verschiebung nach links/unten
                    delta = - lambda / 2;
                } else {
                    // verschiebung nach rechts/oben
                    delta = lambda / 2;
                }
            } else
                delta = 0;

            // berechnung des startpunktes fuer diesen einzelnen graben
            start = y0 + i * lambda + delta;

            // generierung der notwendigen punkte fuer den i. graben
            gr[4][0] = x0_sammel - gb_sammel[i];
            gr[4][1] = start - gb_sammel[i] + lambda / 4;

            gr[5][0] = x0_sammel - gb_sammel[i];
            gr[5][1] = start - gb_sammel[i];

            // korrekturzonenpunkt fuer diesen graben (links bzw. oben)
            schreibePunkt(datei2, x0_sammel - wb_sammel[i], gr[4][1]);

            // aktuellen index merken
            last = i;
        }
    }

    /*  korrekturzone:
        wieder anfangsraum (links vom ersten gitter) und abschluss */
    schreibePunkt(datei2, x0_sammel - wb_sammel[last], gr[5][1]);
    schreibePunkt(datei2, x0_sammel - welle / 2, gr[5][1]);
    schreibePunktNoY(datei2, x0_sammel - welle / 2, - sio2_breite / 2);
    fprintf(datei2, " s\n");

    // auf gezeichneten bereich zoomen
    fprintf(datei, "zoom F %f,%f %f,%f\n",
        xmin / M, ymin / M, xmax / M, ymax / M);

    // ausgabedateien schliessen
    fclose(datei);
    fclose(datei2);

    // info ausgeben
    textbackground(2);
    cprintf("script erfolgreich erstellt.\r\n");
    textbackground(0);
}
```

```
//------------------------------------------------------------------------
/*  erstellung der skripte fuer grundlegende einstellung und
    hilfsstruktur sowie fuer die pruefstruktur (bragg gitter) */
void grund() {

    // anzahl der zu zeichnenden vierecke
    int anz;

    // zeiger auf datei
    FILE *datei;

    // anzahl der zu zeichnenden linien
    int lin;

    // array fuer zu zeichnende linien
    double linie[100][2][2];

    // breite eines lwl
    double lwl_breite;

    // array fuer die zu zeichnenden vierecke
    double pkt[1000][4][2];

    // array fuer die laenge der pruef-bragg-gitter
    int pruef[50];

    // array fuer breite der pruefgitter
    double pruef_breite[50];

    // hilfsvariablen
    int i, j;
    double mitte[50];
    double start, breit;
    char name[80];

    // bildschirm leeren
    clrscr();

    // ----- laden der von acad_v.m gespeicherten anzahl der filter -----

    // dateinamen vorbereiten
    strcpy(name, scrverz);
    strcat(name, "/zeichng.dat");

    // datei oeffnen und anschliessend einlesen
    datei = fopen(name, "r");
    fscanf(datei, "%d", &anzahl);
    if(anzahl>ANZAHL_MAX) anzahl = ANZAHL_MAX;

    // datei schliessen
    fclose(datei);

    // ----- relevante eingaben -----

    printf("\nanzahl der filter fuer diese zeichnung :    ");
    printf("                       %d\n", anzahl);

    printf("breite eines filters (mikrometer) [0->4000] ");
    printf("?                      ");
    scanf("%lf", &breite);
    if(breite==0) breite = 4000;
    breite = breite * 1000;

    printf("hoehe der struktur (mikrometer) [0->4000] ? ");
    printf("                       ");
    scanf("%lf", &hoehe);
    if(hoehe==0) hoehe = 4000;
    hoehe = hoehe * 1000;

    printf("breite eines lwl (mikrometer) [0->125] ?    ");
```

```
printf("                      ");
scanf("%lf", &lwl_breite);
if(lwl_breite==0) lwl_breite = 125;
lwl_breite = lwl_breite * 1000;

printf("breite der seele eines lwl (mikrometer) [0->");
printf("12] ?                 ");
scanf("%lf", &lwl_seele);
if(lwl_seele==0) lwl_seele = 12;
lwl_seele = lwl_seele * 1000;

printf("abstand der lwl fuer ein- und auskopplung (m");
printf("ikrometer) [0->400] ? ");
scanf("%lf", &lwl_abstand);
if(lwl_abstand==0) lwl_abstand = 400;
lwl_abstand = lwl_abstand * 1000;

printf("wellenlaenge im material (nanometer) [0->387");
printf("] ?                   ");
scanf("%lf", &lambda);
if(lambda==0) lambda = 387;

printf("\n");
for(i=0; i<2*anzahl; i++) {
    printf("anzahl der bragg-gitter fuer %2d. (%2d-%d) lwl [0"
        "->kein gitter] ?    ", i+1, abs(i/2)+1, i%2+1);
    scanf("%d", &pruef[i]);

    if(pruef[i]>0) {
        printf("  bragg-gitter-breite in nm [0->lambda/2 1->l"
            "ambda/2/sqrt(2)] ?   ");
        scanf("%lf", &pruef_breite[i]);

        if(pruef_breite[i]==0) pruef_breite[i] = lambda / 2;
        if(pruef_breite[i]==1) pruef_breite[i] = lambda / 2 / sqrt(2);
    } else
        pruef_breite[i] = 0;
}

// ----- generierung der punkte fuers zeichnen -----

// laufende anzahl zuruecksetzen
anz = 0;

/*  -- vierecke fuer die linke und rechte
    kennzeichnung der gesamten struktur -- */
// links
pkt[anz][0][0] = 0;
pkt[anz][0][1] = 0;
pkt[anz][1][0] = breite / 2;
pkt[anz][1][1] = 0;
pkt[anz][2][0] = breite / 2;
pkt[anz][2][1] = hoehe;
pkt[anz][3][0] = 0;
pkt[anz][3][1] = hoehe;
anz++;
// rechts
pkt[anz][0][0] = (anzahl + .5) * breite;
pkt[anz][0][1] = 0;
pkt[anz][1][0] = (anzahl + 1) * breite;
pkt[anz][1][1] = 0;
pkt[anz][2][0] = (anzahl + 1) * breite;
pkt[anz][2][1] = hoehe;
pkt[anz][3][0] = (anzahl + .5) * breite;
pkt[anz][3][1] = hoehe;
anz++;

// -- vierecke fuer jede einzelne einheit / filter --
for(i=0; i<anzahl; i++) {
    // untere linke ecke
```

```
    pkt[anz][0][0] = breite * (i + .5);
    pkt[anz][0][1] = 0;

    // untere rechte ecke
    pkt[anz][1][0] = breite * (i + 1.5);
    pkt[anz][1][1] = 0;

    // obere rechte ecke
    pkt[anz][2][0] = breite * (i + 1.5);
    pkt[anz][2][1] = hoehe;

    // obere linke ecke
    pkt[anz][3][0] = breite * (i + .5);
    pkt[anz][3][1] = hoehe;

    anz++;
}

// -- vierecke fuer jeden einzelnen lwl --
for(i=0; i<anzahl; i++) {
    // - linker lwl -

    // mitte des lwl
    mitte[2*i] = breite * (i + 1) - lwl_abstand / 2;

    // untere linke ecke
    pkt[anz][0][0] = mitte[2*i] - lwl_breite / 2;
    pkt[anz][0][1] = - sio2_breite / 2;

    // untere rechte ecke
    pkt[anz][1][0] = mitte[2*i] + lwl_breite / 2;
    pkt[anz][1][1] = - sio2_breite / 2;

    // obere rechte ecke
    pkt[anz][2][0] = mitte[2*i] + lwl_breite / 2;
    pkt[anz][2][1] = hoehe + sio2_breite / 2;

    // obere linke ecke
    pkt[anz][3][0] = mitte[2*i] - lwl_breite / 2;
    pkt[anz][3][1] = hoehe + sio2_breite / 2;

    anz++;

    // - rechter lwl -

    // mitte des lwl
    mitte[2*i+1] = breite * (i + 1) + lwl_abstand / 2;

    // untere linke ecke
    pkt[anz][0][0] = mitte[2*i+1] - lwl_breite / 2;
    pkt[anz][0][1] = - sio2_breite / 2;

    // untere rechte ecke
    pkt[anz][1][0] = mitte[2*i+1] + lwl_breite / 2;
    pkt[anz][1][1] = - sio2_breite / 2;

    // obere rechte ecke
    pkt[anz][2][0] = mitte[2*i+1] + lwl_breite / 2;
    pkt[anz][2][1] = hoehe + sio2_breite / 2;

    // obere linke ecke
    pkt[anz][3][0] = mitte[2*i+1] - lwl_breite / 2;
    pkt[anz][3][1] = hoehe + sio2_breite / 2;

    anz++;
}

// -- vierecke fuer die seelen der einzelnen lwl --
for(i=0; i<anzahl; i++) {
    // - linker lwl -
```

```
    // untere linke ecke
    pkt[anz][0][0] = mitte[2*i] - lwl_seele / 2;
    pkt[anz][0][1] = - sio2_breite / 2;

    // untere rechte ecke
    pkt[anz][1][0] = mitte[2*i] + lwl_seele / 2;
    pkt[anz][1][1] = - sio2_breite / 2;

    // obere rechte ecke
    pkt[anz][2][0] = mitte[2*i] + lwl_seele / 2;
    pkt[anz][2][1] = hoehe + sio2_breite / 2;

    // obere linke ecke
    pkt[anz][3][0] = mitte[2*i] - lwl_seele / 2;
    pkt[anz][3][1] = hoehe + sio2_breite / 2;

    anz++;

    // - rechter lwl -

    // untere linke ecke
    pkt[anz][0][0] = mitte[2*i+1] - lwl_seele / 2;
    pkt[anz][0][1] = - sio2_breite / 2;

    // untere rechte ecke
    pkt[anz][1][0] = mitte[2*i+1] + lwl_seele / 2;
    pkt[anz][1][1] = - sio2_breite / 2;

    // obere rechte ecke
    pkt[anz][2][0] = mitte[2*i+1] + lwl_seele / 2;
    pkt[anz][2][1] = hoehe + sio2_breite / 2;

    // obere linke ecke
    pkt[anz][3][0] = mitte[2*i+1] - lwl_seele / 2;
    pkt[anz][3][1] = hoehe + sio2_breite / 2;

    anz++;
}

// zaehler fuer linien zuruecksetzen
lin = 0;

// -- linien fuer mitte der einzelnen lwl --
for(i=0; i<anzahl; i++) {
    // - linker lwl -
    linie[lin][0][0] = mitte[2*i];
    linie[lin][0][1] = - sio2_breite / 2;

    linie[lin][1][0] = mitte[2*i];
    linie[lin][1][1] = hoehe + sio2_breite / 2;

    lin++;

    // - rechter lwl -
    linie[lin][0][0] = mitte[2*i+1];
    linie[lin][0][1] = - sio2_breite / 2;

    linie[lin][1][0] = mitte[2*i+1];
    linie[lin][1][1] = hoehe + sio2_breite / 2;

    lin++;
}

// -- waagerechte hilfslinie in der mitte --
linie[lin][0][0] = 0;
linie[lin][0][1] = hoehe / 2;

linie[lin][1][0] = (anzahl + 1) * breite;
linie[lin][1][1] = hoehe / 2;
```

```
lin++;

// ----- erstellung der script-datei fuer die grundstruktur -----

// dateinamen vorbereiten
strcpy(name, scrverz);
strcat(name, "/grund.scr");

// ausgabedatei oeffnen
datei = fopen(name, "w");

// erstmal die einheiten setzen
fprintf(datei, "einheit 2 4 1 0 0 N\n");

// dann die limiten...
fprintf(datei, "limiten %f,%f %f,%f\n",
    (-sio2_breite / 2) / M, (-sio2_breite / 2) / M,
    ((anzahl + 1) * breite + sio2_breite / 2) / M,
    (hoehe + sio2_breite / 2) / M);

// alle erforderlichen layer erstellen
fprintf(datei, "-layer N fuehrung N gitter N sio2\n\n");

// -- justierungsmarken auf allen layern setzen --

// GITTER: aktuellen layer setzen
fprintf(datei, "-layer SE gitter\n\n");

// farbe setzen (rot)
fprintf(datei, "farbe 1\n");

// markierung setzen
markierung(datei);

// FUEHRUNG: aktuellen layer setzen
fprintf(datei, "-layer SE fuehrung\n\n");

// farbe setzen (gruen)
fprintf(datei, "farbe 3\n");

// markierung setzen
markierung(datei);

// 0: aktuellen layer setzen
fprintf(datei, "-layer SE 0\n\n");

// farbe setzen (hellgrau)
fprintf(datei, "farbe 9\n");

// markierung setzen
markierung(datei);

// schreiben der einzelnen daten fuer die zu zeichnenden vierecke
for(i=0; i<anz; i++)
    fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f s\n",
        pkt[i][0][0] / M, pkt[i][0][1] / M,
        pkt[i][1][0] / M, pkt[i][1][1] / M,
        pkt[i][2][0] / M, pkt[i][2][1] / M,
        pkt[i][3][0] / M, pkt[i][3][1] / M);

// schreiben der einzelnen daten fuer die zu zeichnenden linien
for(i=0; i<lin; i++)
    fprintf(datei, "plinie %f,%f %f,%f s\n",
        linie[i][0][0] / M, linie[i][0][1] / M,
        linie[i][1][0] / M, linie[i][1][1] / M);

// SIO2: aktuellen layer setzen
fprintf(datei, "-layer SE sio2\n\n");
```

```
// farbe setzen (weiss)
fprintf(datei, "farbe 7\n");

// markierung setzen
markierung(datei);

// linke kennzeichnung (dreieck) auf sio2 layer
fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f "
    "%f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f s\n",
    (-sio2_breite / 2) / M, (-sio2_breite / 2) / M,
    (breite / 2) / M, (-sio2_breite / 2) / M,
    (breite / 2) / M, (hoehe / 2) / M,
    (breite / 2 - sio2_breite / 2) / M, (hoehe / 2) / M,
    (breite / 2 - sio2_breite / 2) / M, (sio2_breite / 2) / M,
    (sio2_breite / 2) / M, (sio2_breite / 2) / M,
    (sio2_breite / 2) / M, (hoehe / 2) / M,
    (hoehe / 20) / M, (hoehe / 2) / M,
        (hoehe / 20) / M, (hoehe / 20) / M,
    (breite / 2 - hoehe / 20) / M, (hoehe / 2) / M,
    (hoehe / 20) / M, (hoehe - hoehe / 20) / M,
    (hoehe / 20) / M, (hoehe / 2) / M,
    (sio2_breite / 2) / M, (hoehe / 2) / M,
    (sio2_breite / 2) / M, (hoehe - sio2_breite / 2) / M,
    (breite / 2 - sio2_breite / 2) / M, (hoehe - sio2_breite / 2) / M,
    (breite / 2 - sio2_breite / 2) / M, (hoehe / 2) / M,
    (breite / 2) / M, (hoehe / 2) / M,
    (breite / 2) / M, (hoehe + sio2_breite / 2) / M,
    (-sio2_breite / 2) / M, (hoehe + sio2_breite / 2) / M);

// brechhilfe fuer die einzelnen filter und kennzeichnung der lwls
for(i=0; i<anzahl; i++) {
    start = breite * (i + .5);

    // brechstruktur
    fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f %f,%f",
        start / M, (-sio2_breite / 2) / M,
        (start + breite) / M, (-sio2_breite / 2) / M,
        (start + breite) / M, (hoehe / 2) / M,
        (start + breite - sio2_breite / 2) / M, (hoehe / 2) / M,
        (start + breite - sio2_breite / 2) / M, (sio2_breite / 2) / M);

    // kennzeichnung lwl rechts unten
    // rechts davon
    kennzeichnung(datei, mitte[2*i+1], 0, 1, 1);
    // links davon
    kennzeichnung(datei, mitte[2*i+1], 0, -1, 1);

    // kennzeichnung lwl links unten
    // rechts davon
    kennzeichnung(datei, mitte[2*i], 0, 1, 1);
    // links davon
    kennzeichnung(datei, mitte[2*i], 0, -1, 1);

    // brechstruktur
    fprintf(datei, " %f,%f %f,%f",
        (start + sio2_breite / 2) / M, (sio2_breite / 2) / M,
        (start + sio2_breite / 2) / M, (hoehe - sio2_breite / 2) / M);

    // kennzeichnung lwl links oben
    // links davon
    kennzeichnung(datei, mitte[2*i], hoehe, -1, -1);
    // rechts davon
    kennzeichnung(datei, mitte[2*i], hoehe, 1, -1);

    // kennzeichnung lwl rechts oben
    // links davon
    kennzeichnung(datei, mitte[2*i+1], hoehe, -1, -1);
    // rechts davon
    kennzeichnung(datei, mitte[2*i+1], hoehe, 1, -1);
```

```
    // brechstruktur
    fprintf(datei, " %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f s\n",
        (start + 3 * breite / 4) / M, (hoehe - sio2_breite / 2) / M,
        (start + breite - sio2_breite/2) / M, (3 * hoehe / 4) / M,
        (start + breite - sio2_breite / 2) / M, (hoehe / 2) / M,
        (start + breite) / M, (hoehe / 2) / M,
        (start + breite) / M, (hoehe + sio2_breite / 2) / M,
        start / M, (hoehe + sio2_breite / 2) / M);
}

// auf zeichnungsgrenzen zoomen
fprintf(datei, "_zoom G\n");

// ausgabedatei schliessen
fclose(datei);

// information ausgeben
textbackground(2);
cprintf("\nscript fuer grundstruktur (hilfsstruktur) erstellt.\r\n");
textbackground(0);

// ----- erstellung der script-datei fuer die pruefstruktur -----

// dateinamen vorbereiten
strcpy(name, scrverz);
strcat(name, "/pruef.scr");

// ausgabedatei oeffnen
datei = fopen(name, "w");

// durchgehen der einzelnen pruefstrukturen
for(i=0; i<2*anzahl; i++) {
    // - schreiben der einzelnen daten fuer die zu zeichnenden graeben -
    if(pruef[i]>0) {
        // y-startwert der graeben bestimmen
        start = (hoehe - pruef_breite[i] * (2 * pruef[i] - 1)) / 2;

        // GITTER: aktuellen layer setzen
        fprintf(datei, "-layer SE gitter\n\n");

        // farbe setzen (rot)
        fprintf(datei, "farbe 1\n");

        for(j=0; j<pruef[i]; j++)
            fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f s\n",
                (mitte[i] - pruef_laenge / 2) / M,
                (start + j * 2 * pruef_breite[i]) / M,
                (mitte[i] + pruef_laenge / 2) / M,
                (start + j * 2 * pruef_breite[i]) / M,
                (mitte[i] + pruef_laenge / 2) / M,
                (start + (j * 2 + 1) * pruef_breite[i]) / M,
                (mitte[i] - pruef_laenge / 2) / M,
                (start + (j * 2 + 1) * pruef_breite[i]) / M);
    }

    // - schreiben der fuehrungsstruktur -
    // FUEHRUNG: aktuellen layer setzen
    fprintf(datei, "-layer SE fuehrung\n\n");

    // farbe setzen (gruen)
    fprintf(datei, "farbe 3\n");

    // allgemeine punkte (unten links und unten rechts)
    fprintf(datei, "plinie %f,%f %f,%f",
        (mitte[i] - welle / 2) / M, (- sio2_breite / 2) / M,
        (mitte[i] + welle / 2) / M, (- sio2_breite / 2) / M);

    if(pruef[i]>0) {
        // fuehrung, da wo die gitter sind, aufweiten
        fprintf(datei, " %f,%f %f,%f %f,%f %f,%f",
```

```
            (mitte[i] + welle / 2) / M, (start - pruef_breite[i]) / M,
            (mitte[i] + pruef_laenge / 2) / M,
                (start - pruef_breite[i]) / M,
            (mitte[i] + pruef_laenge / 2) / M,
                (start + 2 * pruef[i] * pruef_breite[i]) / M,
            (mitte[i] + welle / 2) / M,
                (start + 2 * pruef[i] * pruef_breite[i]) / M);
    }

    // allgemeine punkte (oben links und oben rechts)
    fprintf(datei, " %f,%f %f,%f",
        (mitte[i] + welle / 2) / M, (hoehe + sio2_breite / 2) / M,
        (mitte[i] - welle / 2) / M, (hoehe + sio2_breite / 2) / M);

    if(pruef[i]>0) {
        // fuehrung, da wo die gitter sind, aufweiten
        fprintf(datei, " %f,%f %f,%f %f,%f %f,%f",
            (mitte[i] - welle / 2) / M,
                (start + 2 * pruef[i] * pruef_breite[i]) / M,
            (mitte[i] - pruef_laenge / 2) / M,
                (start + 2 * pruef[i] * pruef_breite[i]) / M,
            (mitte[i] - pruef_laenge / 2) / M,
                (start - pruef_breite[i]) / M,
            (mitte[i] - welle / 2) / M, (start - pruef_breite[i]) / M);
    }

    // und schliessen der poly-linie
    fprintf(datei, " s\n");
}

// -- kennzeichnung der pruefstruktur

// aktuellen layer setzen
fprintf(datei, "-layer SE sio2\n\n");

// farbe setzen (weiss)
fprintf(datei, "farbe 7\n");

// brechhilfe und kennzeichnung der struktur (rechts)
   start = breite * (anzahl + .5);
breit = breite / 6 - hoehe / 20;

fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f "
    "%f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f "
    "%f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f "
    "%f,%f %f,%f %f,%f %f,%f %f,%f s\n",
    // anfang brechstruktur
    start / M, (-sio2_breite / 2) / M,
    (start + breite / 2 + sio2_breite / 2) / M, (-sio2_breite / 2) / M,
    (start + breite / 2 + sio2_breite / 2) / M, (hoehe / 2) / M,
    (start + breite / 2 - sio2_breite / 2) / M, (hoehe / 2) / M,
    (start + breite / 2 - sio2_breite / 2) / M, (sio2_breite / 2) / M,
    (start + sio2_breite / 2) / M, (sio2_breite / 2) / M,
    (start + sio2_breite / 2) / M, (hoehe - sio2_breite / 2) / M,
    (start + breite / 4) / M, (hoehe - sio2_breite / 2) /M,
    // das p
    (start + breite / 4) / M, (hoehe - hoehe / 20) / M,
    (start + hoehe / 20) / M, (hoehe - hoehe / 20) / M,
    (start + hoehe / 20) / M, (hoehe / 20) / M,
    (start + breite / 6) / M, (hoehe / 20) / M,
    (start + breite / 6) / M, (hoehe / 2 - breit / 2) / M,
    (start + breite / 2 - hoehe / 20) / M, (hoehe / 2 - breit / 2) / M,
    (start + breite / 2 - hoehe / 20) / M, (hoehe / 2 + breit / 2) / M,
    (start + breite/2 - hoehe/20 - breit) / M, (hoehe/2 + breit/2) / M,
    (start + breite / 6) / M, (hoehe / 2 + breit / 2) / M,
    (start + breite / 6) / M, (hoehe - hoehe / 20 - breit) / M,
    (start + breite/2 - hoehe/20 - breit)/M, (hoehe - hoehe/20 - breit)/M,
    (start + breite/2 - hoehe/20 - breit) / M, (hoehe/2 + breit/2) / M,
    (start + breite / 2 - hoehe / 20) / M, (hoehe / 2 + breit / 2) / M,
    (start + breite / 2 - hoehe / 20) / M, (hoehe - hoehe / 20) / M,
```

```
        (start + breite / 4) / M, (hoehe - hoehe / 20) / M,
        // ende brechstruktur
        (start + breite / 4) / M, (hoehe - sio2_breite / 2) /M,
        (start + breite / 2 - sio2_breite/2) / M, (hoehe - sio2_breite/2) / M,
        (start + breite / 2 - sio2_breite / 2) / M, (hoehe / 2) / M,
        (start + breite / 2 + sio2_breite / 2) / M, (hoehe / 2) / M,
        (start + breite / 2 + sio2_breite/2) / M, (hoehe + sio2_breite/2) / M,
        start / M, (hoehe + sio2_breite / 2) / M);

    // ausgabedatei schliessen
    fclose(datei);

    // information ausgeben
    textbackground(2);
    cprintf("script fuer pruefstruktur erstellt.\r\n");
    textbackground(0);

    // ----- erstellung der daten-datei zur sicherung der relevanten werte -----

    // dateinamen vorbereiten
    strcpy(name, scrverz);
    strcat(name, "/zeichng.dat");

    // ausgabedatei oeffnen
    datei = fopen(name, "w");

    // daten schreiben
    fprintf(datei, "%d\n", anzahl);
    fprintf(datei, "%f\n", breite);
    fprintf(datei, "%f\n", hoehe);
    fprintf(datei, "%f\n", lwl_abstand);
    fprintf(datei, "%f\n", lambda);

    // ausgabedatei schliessen
    fclose(datei);

    // information ausgeben
    textbackground(2);
    cprintf("relevante zeichnungsdaten gesichert.\r\n");
    textbackground(0);

    // 0 + enter zum fortfahren
    taste();
}


//---------------------------------------------------------------------------
/*  schreiben der punkte fuer eine lwl kennzeichnung
    lire: links = -1, rechts = 1
    obun: oben = -1, unten = 1 */
void kennzeichnung(FILE *datei, double x0, double y0, int lire, int obun) {

    fprintf(datei, " %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f %f,%f",
        (x0 + lire * (3 * lwl_seele / 2 + kenn_breit / 2)) / M,
        (y0 + obun * sio2_breite / 2) / M,
        (x0 + lire * (3 * lwl_seele / 2 + kenn_breit / 2)) / M,
        (y0 + obun * (sio2_breite / 2 + kenn_breit)) / M,
        (x0 + lire * (3 * lwl_seele / 2 + kenn_breit)) / M,
        (y0 + obun * (sio2_breite / 2 + kenn_breit)) / M,
        (x0 + lire * (3 * lwl_seele / 2 + kenn_breit)) / M,
        (y0 + obun * (sio2_breite / 2 + kenn_breit + kenn_hoch)) / M,
        (x0 + lire * 3 * lwl_seele / 2) / M,
        (y0 + obun * (sio2_breite / 2 + kenn_breit + kenn_hoch)) / M,
        (x0 + lire * 3 * lwl_seele / 2) / M,
        (y0 + obun * (sio2_breite / 2 + kenn_breit)) / M,
        (x0 + lire * (3 * lwl_seele / 2 + kenn_breit / 2)) / M,
        (y0 + obun * (sio2_breite / 2 + kenn_breit)) / M,
        (x0 + lire * (3 * lwl_seele / 2 + kenn_breit / 2)) / M,
        (y0 + obun * sio2_breite / 2) / M);
}
```

```
//---------------------------------------------------------------------------
/*  einlesen aller datenwerte aus einer ascii-datei
    und ablegen dieser in einem array. falls verify
    verschieden null ist, wird geprueft, ob der i.
    wert des eingelesenen feldes (dest) groesser als der
    i. wert des verify feldes ist (wb[i]>gb[i]) */
int liesArray(char *source, double *dest, double *verify, char *ruecken) {

    // zaehler
    int count = 0;

    // checkvarialbel fuer pruefung
    bool ok = true;

    // hilfsvariablen
    double value;
    int c;

    // datei oeffnen und anschliessend lesen
    FILE *datei = fopen(source, "r");

    /*  pro zeile ein wert, wir muessen aber durch zwei teilen;
        komplette grabenbreite muss in der datei in nanometern vorliegen */
    while(((c=fscanf(datei, "%lf", &value))!=EOF) && c>0) {
        dest[count] = value / 2;

        // auf laengenueberschreitung pruefen
        if(count>GRABEN_MAX-1) {
            return -1;
        }

        /*  pruefen, ob dest[count]<verify[count] ist.
            wenn ja: checkvariable auf falsch setzen */
        if(verify!=NULL) {
            if(dest[count]<verify[count])
                ok = false;
        }

        // zaehlvariable erhoehen
        count++;
    }

    // eingabedatei schliessen
    fclose(datei);

    // warnung ausgeben, falls dest[i] irgendwann kleiner verify[i] war
    if(verify!=NULL) {
        if(!ok) {
            textbackground(1);
            cprintf("warnung: gitterbreite groesser als korrekturweite ");
            cprintf("(%s-ruecken) !\r\n", ruecken);
            textbackground(0);
        }
    }

    return count;
}


//---------------------------------------------------------------------------
// marken fuer justierung der einzelnen layer setzen
void markierung(FILE *datei) {

    // unten links
    marke(datei, -(mark_abstand + mark), -(mark_abstand + mark));

    // unten rechts
    marke(datei, (anzahl+1) * breite + mark_abstand, -(mark_abstand + mark));
```

```
    // oben rechts
    marke(datei, (anzahl+1) * breite + mark_abstand, hoehe + mark_abstand);

    // oben links
    marke(datei, -(mark_abstand + mark), hoehe + mark_abstand);
}


//------------------------------------------------------------------------
// punkte fuer eine justiermarke schreiben
void marke(FILE *datei, double x0, double y0) {

    fprintf(datei, "plinie %f,%f %f,%f %f,%f %f,%f s\n",
        x0 / M, y0 / M,
        (x0 + mark) / M, y0 / M,
        (x0 + mark) / M, (y0 + mark) / M,
        x0 / M, (y0 + mark) / M);
}


//------------------------------------------------------------------------
/*  einen punkt in eine datei schreiben
    dabei x- und y-koordinate auf maximum bzw.
    minimum preufen (fuers zoomen) */
void schreibePunkt(FILE *datei, double x, double y) {

    // minima pruefen
    if(x<xmin) xmin = x;
    if(y<ymin) ymin = y;

    // maxima pruefen
    if(x>xmax) xmax = x;
    if(y>ymax) ymax = y;

    fprintf(datei, " %f,%f", x / M, y / M);
}


//------------------------------------------------------------------------
/*  einen punkt in eine datei schreiben
    dabei x-koordinate auf maximum bzw.
    minimum preufen (fuers zoomen) */
void schreibePunktNoY(FILE *datei, double x, double y) {

    // minima pruefen
    if(x<xmin) xmin = x;

    // maxima pruefen
    if(x>xmax) xmax = x;

    fprintf(datei, " %f,%f", x / M, y / M);
}


//------------------------------------------------------------------------
/*  warten nach einer erstellung, damit der benutzer
    die information lesen kann - dafuer:
    0 und eingabetaste druecken, um fortzufahren = menue wieder anzeigen */
void taste() {

    // hilfsvariable
    int help;

    printf("\n0 und anschliessend die Eingabetaste druecken, ");
    printf("um ins Menue zu gelangen... ");
    scanf("%d", &help);
}
```

```
//---------------------------------------------------------------------------
// fehlermeldung wegen unterschiedlicher rueckenlaengen ausgeben
void vl_fehler(char *welche) {

    textbackground(1);
    cprintf("filterlaenge in \"%s\" stimmt nicht !\r\n", welche);
    textbackground(0);
}


//---------------------------------------------------------------------------
/*  schreiben der punkte fuer den wellensumpf am ende des streugitters.
    uebergen werden die datei, in welche geschrieben werden soll,
    die startkoordinate in y-richtung (y0) sowie die breite bei y0 (b0). */
void wellensumpf(FILE *datei, double x0, double y0, double b0) {

    /*  wir gehen bis maximal e^-expo_max und setzen somit expo_max punkte,
        und zwar bei e^-1, e^-2, ..., e^-expo_max. */
    int expo_max = 10;

    /*  die auflösung kann durch setzen der variable faktor veraendert werden.
        faktor = 1 entspricht expo_max, faktor = 2 entspricht 2 * expo_max
        punkten, usw. */
    int faktor = 10;

    /*  der abstand zwischen dem punkt bei e^0 und e^-1 wird mit der
        variablen delta gesetzt. er wird in nanometer angegeben.
        die länge des wellensumpfs ergibt sich zu
        expo_max * delta + delta / faktor.
        wir wollen, dass sie genau die hälfte des verbleibenden weges
        bis zum rand der struktur einnimmt. der eigentliche sumpf
        beginnt erst nach 10% dieser strecke. bis dahin bleiben die
        begrenzungslinien parallel. */
    double delta = .9 * (hoehe - y0) / (2 * (expo_max + 1 / faktor));

    // die y-koordinate fuer den beginn des sumpfes.
    double y0_sumpf = y0 + .1 * (hoehe - y0) / 2;

    // hilfsvariablen
    int i, j;
    double x, y, expo;

    // -- rechte seite --
    // parallele begrenzung
    schreibePunktNoY(datei, x0 + b0 / 2, y0_sumpf);
    // wellensumpf
    for(i=0; i<expo_max; i++)
        for(j=faktor-1; j>=0; j--) {
            expo = i + 1 - (double) j / faktor;
            x = x0 + b0 * exp(-expo) / 2;
            y = y0_sumpf + expo * delta;
            schreibePunktNoY(datei, x, y);
        }

    // mittlerer punkt bei x0
    schreibePunktNoY(datei, x0, y0_sumpf + expo_max * delta + delta / faktor);

    // -- linke seite --
    // wellensumpf
    for(i=expo_max-1; i>=0; i--)
        for(j=0; j<faktor; j++) {
            expo = i + 1 - (double) j / faktor;
            x = x0 - b0 * exp(-expo) / 2;
            y = y0_sumpf + expo * delta;
            schreibePunktNoY(datei, x, y);
        }
    // parallele begrenzung
    schreibePunktNoY(datei, x0 - b0 / 2, y0_sumpf);
}
```

```
//---------------------------------------------------------------------------
// fehlermeldung wegen zu hoher rueckenlaenge ausgeben
void zl_fehler() {

    textbackground(1);
    cprintf("streu- oder sammelrueckenlaenge ist zu gross !\r\n");
    textbackground(0);
}
```

D.2. ainr.cpp

```
/*  berechnung der reflexionsfaktoren fuer den sammelgraben aus
    den filterkoeffizienten (si-funktion).

    Michael Becker, 11.02.2000 - 06.03.2000 */

//---------------------------------------------------------------------------
#include <vcl\condefs.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include <d:\studium\diplom\cpp\include\sagrab.h>
#include <d:\studium\diplom\cpp\include\mytime.h>

#pragma hdrstop
//---------------------------------------------------------------------------
USERES("ainr.res");

const int M_MAX = 7000;
const int P_MAX = 8;
const complex i(0, 1);

//---------------------------------------------------------------------------
int main(int argc, char **argv) {

    // chararray fuer quelldatei
    char quelle[80];

    // chararray fuer zieldatei
    char ziel[80];

    // aktuelle zeit und aktuelles datum
    char zeit_str[79];
    char datum_str[79];

    // rueckenhoehe setzen
    int P = 8;

    // breite des rueckens
    int M = 0;

    // array fuer filterkoeffizienten
    double aE[M_MAX];

    // array fuer zu berechnende reflexionsfaktoren
    double rn[M_MAX];

    // maximaler, minimaler reflexionsfaktor
    int idxmax = -1, idxmin = -1;
    double rmax = 0.0, rmin = 0.0;

    // zeiger auf datei
    FILE *datei;
```

```
// array fuer alle vertikalen ausgaenge
complex aus_alle[P_MAX];

// array fuer alle transmittierten werte
complex trans_alle[P_MAX];

// array fuer relevante vertikale ausgaenge fuer berechnung
double aus[P_MAX];

// array fuer relevante transmittierte werte fuer berechnung
double faktor[P_MAX];

// untere eingaenge fuer den ruecken
complex E[M_MAX];

// vordere eingaenge fuer den ruecken
complex Evorn[P_MAX];

// index des aktuellen eingangs
int akt_eing;

// produkt der vorher liegenden transmissionsfaktoen
double tt;

// variablen zur loesung der gleichung
double pN, pZ, qp, qq, a, b, d, delta1, delta2;
double r1, r2, zaehler, nenner;

// linear(=false) oder quadratisch(=true)
bool quad;

// hilfsvariablen
double value;
int c;

// laufvariablen
int m, p, ii;

// -- pruefen, ob erforderliche parameter uebergeben wurden --
if(argc!=4 && argc!=6) {
    // falsche anzahl oder fehlende parameter
    printf("\n (1) ainr <inputdatei> <outputdatei> <l/q>");
    printf("\n (2) ainr ");
    printf("<rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <l/q>\n");
    return -1;
} else {
    if(argc==4) {
        strcpy(quelle, argv[1]);
        strcpy(ziel, argv[2]);

        // linear oder quadratisch ?
        if(strcmp(argv[3], "q")==0)
            quad = true;
        else
            quad = false;
    }

    if(argc==6) {
        strcpy(quelle, argv[2]);
        strcat(quelle, "pi\\");
        strcat(quelle, argv[1]);
        strcat(quelle, "\\");
        strcat(quelle, argv[3]);
        strcat(quelle, "\\a\\");
        strcat(quelle, argv[4]);

        strcpy(ziel, argv[2]);
        strcat(ziel, "pi\\");
        strcat(ziel, argv[1]);
        strcat(ziel, "\\");
```

```
            strcat(ziel, argv[3]);
            strcat(ziel, "\\sammel\\r\\");
            strcat(ziel, argv[4]);
            strcat(ziel, argv[5]);

            // linear oder quadratisch ?
            if(strcmp(argv[5], "q")==0)
                quad = true;
            else
                quad = false;
        }
    }
/*
printf("quelle: %s\n", quelle);
printf("ziel:   %s\n", ziel);
if(quad) printf("quadratisch\n"); else printf("linear\n");
return -1;
*/
    // -- begruessung --
    printf("\n\n  herzlich willkommen!\n");
    printf("  wir rechnen heute mal ein bisschen a in r um,\n");
    printf("  und das auch noch schraeg und mit loesungsformel...\n  == ");

    // quadratisch oder linear ?
    if(quad)
        printf("quadratische betrachtung");
    else
        printf("lineare betrachtung");

    printf(" ==\n ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

    // filter koeffizienten laden
    printf("-> lade filterkoeffizienten aus '%s'...", quelle);
    datei = fopen(quelle, "r");

    while(((c=fscanf(datei, "%lf", &value))!=EOF) && c>0) {
        if(M==M_MAX) {
            printf("\nainr: RUECKENBREITE ZU GROSS! (MAX=%d)", M_MAX);
            printf("\n      RESTLICHE WERTE WERDEN NICHT VERWENDET.\n");
            scanf("%d", &c);
            c = 1;
            break;
        }

        aE[M++] = value;
    }

    fclose(datei);
    if(c==-1) printf(" OK.\n\n");

    // -- berechnung --
    datum(datum_str);
    zeit(zeit_str);
    printf("beginn der berechnung am %s um %s uhr.\n", datum_str, zeit_str);

    for(m=0; m<M; m++) {
        /*  vertikale ausgaenge zuruecksetzen.
            erster ausgang ist null, weil da nur die
            reflexions am gesuchten r stattfindet */
        for(p=0; p<P; p++) {
            aus[p] = 0.0;
        }

        /*  faktoren fuer reflexion mit dem zu bestimmenden
            reflexionsfaktor zuruecksetzen.
            erster faktor ist eins, weil dort nur die reflexion
            mit dem zu bestimmenden reflexionsfaktor vorkommt. */
        for(p=0; p<P; p++) {
            faktor[p] = 0.0;
        }
```

```
        faktor[0] = 1.0;

        // und los
        for(p=1; p<P; p++) {
            if(m==0) {
                // erster reflexionsfaktor soll bestimmt werden
                faktor[p] = 1.0;
            } else {
                /*    untere eingaenge (ohne aktuelle spalte) einzeln setzen
                    und entsprechenden ausgang berechnen (ab m-1 bis m-P+1) */
                akt_eing = m - p;

                // eingangsvektoren vorbereiten
                for(ii=0; ii<M; ii++) {
                    E[ii] = 0.0;
                }

                for(ii=0; ii<P-1; ii++) {
                    Evorn[ii] = 0.0;
                }

                if(akt_eing>=0) {
                    // entsprechenden eingang setzen
                    E[akt_eing] = 1.0f / i;
                } else {
                    // entsprechenden vorderen eingang setzen
                    Evorn[akt_eing+P-1] = 1.0f / i;
                }

                // entsprechenden ruecken berechnen
                sagrab(rn, M, m-1, P, E, Evorn, -1, -1, aus_alle, trans_alle);
                aus[p] = real(aus_alle[p]);
/*
for(ii=0; ii<P; ii++) {
    printf("\naus%d=%g",ii+1,real(aus_alle[ii]));
}
*/
                faktor[p] = real(trans_alle[p-1] * i);
            }
        }

        /*  zu jedem ausgang muss noch die reflexion 1. ordnung am graben
            mit dem zu bestimmenden reflexionsfaktor hinzufuegt werden. */

        // produkt der links vorher liegenden transmissionsfaktoren bilden.
        tt = 1.0;
        for(ii=0; ii<m; ii++) {
            tt *= sqrt(1 - rn[ii] * rn[ii]);
        }

        if(quad) {
            // quadratische betrachtung
            pN = 0.0;
            pZ = 0.0;
            qq = 0.0;
            for(p=0; p<P; p++) {
                a = tt * faktor[p];
                b = aus[p];

//printf("\ntt=%g; faktor=%g; a=%g; b=%g;", tt, faktor[p], a, b);

                pN += a * b;
                pZ += a * a;

                qq += b * b;
            }

            qp = 2 * pN / pZ;
            qq = (qq - aE[m] * aE[m]) / pZ;
```

```
            // diskriminante bestimmen
            d = qp * qp / 4 - qq;

//printf("\np=%g; q=%g; d=%g; aE=%g;\n", qp, qq, d, aE[m]);

            // gleichung loesen
            if(d>=0) {
                // reelles ergebnis
                r1 = - qp / 2 - sqrt(d);
                r2 = - qp / 2 + sqrt(d);
            } else {
                // komplexes ergebnis!
                printf("\rreflexionsfaktor fuer spalte %d komplex."
                    "                                  \n", m+1);

                // entsprechenden reflexionsfaktor auf 0 setzen
                r1 = 0.0;
                r2 = 0.0;
            }
        } else {
            // lineare betrachtung
            zaehler = aE[m];
            for(p=0; p<P; p++) zaehler -= aus[p];

            nenner = faktor[0];
            for(p=1; p<P; p++) nenner += faktor[p];
            nenner *= tt;

            r1 = zaehler / nenner;
            if(fabs(r1)>1) {
                printf("\rbetrag des reflexionsfaktors fuer spalte %d "
                    "groesser eins.                \n", m+1);

                /*  wir brechen den ganzen spass hier ab,
                    ausgabedatei erstellen und beenden. */
                datei = fopen(ziel, "w");
                fprintf(datei, "%d\n", m+1);
                fclose(datei);
                return -1;
            }

            // nur eine loesung
            r2 = 100.0;
        }

        // abstaende zum filterkoeffizienten bestimmen
        delta1 = fabs(r1 - aE[m]);
        delta2 = fabs(r2 - aE[m]);

        /*  wir nehmen das ergebnis fuer r, welches am naechsten bei dem
            zugehoerigen aE liegt -> kleinstes delta suchen */
        if(delta1<delta2)
            rn[m] = r1;
        else
            rn[m] = r2;

        // auf maximalwert pruefen
        if(rn[m]>rmax) {
            rmax = rn[m];
            idxmax = m;
        }

        // auf minimalwert pruefen
        if(rn[m]<rmin) {
            rmin = rn[m];
            idxmin = m;
        }

        // ausgeben wie weit wir sind
```

```
        printf("\r%4d von %4d [r=%6.4f%%, rmin(%d)=%6.4f%%, "
            "rmax(%d)=%6.4f%%] fertig. ", m+1, M, 100*rn[m],
            idxmin+1, 100*rmin, idxmax+1, 100*rmax);

//if(m==9) break;
    }

    // ausgabe der zeit und der extremwerte
    datum(datum_str);
    zeit(zeit_str);
    printf("\rende der berechnung am %s um %s uhr.                          \n",
        datum_str, zeit_str);
    printf("rmin = %f%% bei index %d, rmax = %f%% bei index %d\n",
        100*rmin, idxmin+1, 100*rmax, idxmax+1);

    // speichern
    printf("\n-> speichere reflexionsfaktoren in '%s'...", ziel);
    datei = fopen(ziel, "w");

    for(m=0; m<M; m++) fprintf(datei, "%g\n", rn[m]);

    fclose(datei);
    printf(" OK.\n\n");

    return 0;
}
```

D.3. ainr2.cpp

```
/*  berechnung der reflexionsfaktoren fuer den streu- und sammelgraben aus
    den filterkoeffizienten (si-funktion).

    Michael Becker, 11.02.2000 - 27.04.2000 */

//---------------------------------------------------------------------------
#include <vcl\condefs.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include <d:\studium\diplom\cpp\include\stgrab.h>
#include <d:\studium\diplom\cpp\include\sagrab.h>
#include <d:\studium\diplom\cpp\include\mytime.h>

#pragma hdrstop
//---------------------------------------------------------------------------
USERES("ainr2.res");

const int M_MAX = 7000;
const int P_MAX = 8;
//const complex i(0, 1);

//---------------------------------------------------------------------------
int main(int argc, char **argv) {

    // chararray fuer quelldatei
    char quelle[80];

    // chararray fuer zieldatei (streu)
    char ziel1[80];

    // chararray fuer zieldatei (sammel)
    char ziel2[80];

    // aktuelle zeit und aktuelles datum
    char zeit_str[79];
```

```
char datum_str[79];

// rueckenhoehe setzen
int P = 8;

// breite des rueckens
int M = 0;

// array fuer filterkoeffizienten
double aE[M_MAX];

// array fuer zu berechnende reflexionsfaktoren
double rn[M_MAX];

// maximaler, minimaler reflexionsfaktor
int idxmax = -1, idxmin = -1;
double rmax = 0.0, rmin = 0.0;

// zeiger auf datei
FILE *datei;

// array fuer alle horizontalen ausgaenge des streuruecken
complex ausHori[M_MAX+P_MAX-1];

// array fuer alle horizontalen eingaenge des sammelrueckens
complex einHori[M_MAX+P_MAX-1];

// array fuer alle vertikalen ausgaenge des sammelrueckens
complex ausVert[P_MAX];

// array fuer relevante vertikale ausgaenge fuer berechnung
double aus[P_MAX];

// array fuer relevante transmittissionsfaktoren fuer berechnung
double faktor[P_MAX];

// vertikale eingaenge fuer den streuruecken
complex ein[M_MAX];

// produkt der vorher liegenden transmissionsfaktoen
double tt;

// variablen zur loesung der quadratischen gleichung
double pN, pZ, qp, qq, a, b, d, zaehler, nenner;
double r1, r2, rf[4], delta, delta_min;
int index;

// linear(=false) oder quadratisch(=true)
bool quad;

// hilfsvariablen
double value;
int c, ende;

// laufvariablen
int m, p, ii;

// -- pruefen, ob erforderliche parameter uebergeben wurden --
if(argc!=5 && argc!=6) {
    // falsche anzahl oder fehlende parameter
    printf("\n (1) ainr2 <inputdatei1> <outputdatei1> <outputdatei2> ");
    printf("<l/q>\n (2) ainr2 ");
    printf("<rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <l/q>\n");
    return -1;
} else {
    if(argc==5) {
        strcpy(quelle, argv[1]);
        strcpy(ziel1, argv[2]);
        strcpy(ziel2, argv[3]);
```

```
            // linear oder quadratisch ?
            if(strcmp(argv[4], "q")==0)
                quad = true;
            else
                quad = false;
        }

        if(argc==6) {
            // quelldatei mit filterkoeffizienten
            strcpy(quelle, argv[2]);
            strcat(quelle, "pi\\");
            strcat(quelle, argv[1]);
            strcat(quelle, "\\");
            strcat(quelle, argv[3]);
            strcat(quelle, "\\a\\");
            strcat(quelle, argv[4]);

            // zieldatei streu
            strcpy(ziel1, argv[2]);
            strcat(ziel1, "pi\\");
            strcat(ziel1, argv[1]);
            strcat(ziel1, "\\");
            strcat(ziel1, argv[3]);
            strcat(ziel1, "\\streu\\r2\\");
            strcat(ziel1, argv[4]);
            strcat(ziel1, argv[5]);

            // zieldatei sammel
            strcpy(ziel2, argv[2]);
            strcat(ziel2, "pi\\");
            strcat(ziel2, argv[1]);
            strcat(ziel2, "\\");
            strcat(ziel2, argv[3]);
            strcat(ziel2, "\\sammel\\r2\\");
            strcat(ziel2, argv[4]);
            strcat(ziel2, argv[5]);

            // linear oder quadratisch ?
            if(strcmp(argv[5], "q")==0)
                quad = true;
            else
                quad = false;
        }
    }
/*
printf("quelle: %s\n", quelle);
printf("ziel1:  %s\n", ziel1);
printf("ziel2:  %s\n", ziel2);
if(quad) printf("quadratisch\n"); else printf("linear\n");
return -1;
*/
    // -- begruessung --
    printf("\n\n  herzlich willkommen!\n");
    printf("  wir rechnen heute mal ein bisschen a in r um,\n");
    printf("  und das auch noch schraeg\n");
    printf("  und mit loesungsformel\n");
    printf("  und auch noch fuer beide ruecken...\n  == ");

    // quadratisch oder linear ?
    if(quad)
        printf("quadratische betrachtung");
    else
        printf("lineare betrachtung");

    printf(" ==\n ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

    // filter koeffizienten laden
    printf("-> lade filterkoeffizienten aus '%s'...", quelle);
    datei = fopen(quelle, "r");
```

```
while(((c=fscanf(datei, "%lf", &value))!=EOF) && c>0) {
    if(M==M_MAX) {
        printf("\nainr: RUECKENBREITE ZU GROSS! (MAX=%d)", M_MAX);
        printf("\n      RESTLICHE WERTE WERDEN NICHT VERWENDET.\n");
        scanf("%d", &c);
        c = 1;
        break;
    }

    aE[M++] = value;
}

fclose(datei);
if(c==-1) printf(" OK.\n\n");

// -- berechnung --
datum(datum_str);
zeit(zeit_str);
printf("beginn der berechnung am %s um %s uhr.\n", datum_str, zeit_str);

for(m=0; m<M; m++) {
    /*  vertikale ausgaenge des sammelgrabens zuruecksetzen sowie
        faktoren fuer reflexion am graben mit dem zu bestimmenden
        reflexionsfaktor vorerst auf eins setzen. */
    for(p=0; p<P; p++) {
        aus[p] = 0.0;
        faktor[p] = 1.0;
    }

    /*  letzter faktor ist eins und letztes aus ist null, weil dort
        nur die reflexion mit dem zu bestimmenden reflexionsfaktor
        vorkommt. diese werte sind schon gesetzt.
        die restlichen werden jetzt berechnet. */
    for(p=0; p<P-1; p++)
        if(m==0) {
            /*  erster reflexionsfaktor soll bestimmt werden,
                faktor ist eins und aus ist null, weil links keine
                graeben vorhanden sind, alles schon so gesetzt. */
        } else {
            /*  vertikale eingaenge einzeln setzen, struktur berechnen
                und entsprechenden ausgang herausfischen. */

            // eingangsvektor vorbereiten
            for(ii=0; ii<P; ii++) ein[ii] = complex(0.0f, 0.0f);

            // aktuellen eingang auf eins setzen
            ein[p] = complex(1.0f, 0.0f);

            // streuruecken berechnen
            stgrab(rn, M, m-1, P, ein, -1, -1, ausHori);

            // alle irrelevanten ausgaenge unterdruecken
            for(ii=0; ii<M+P-1; ii++)
                if(ii==m+p)
                    einHori[ii] = ausHori[ii];
                else
                    einHori[ii] = complex(0.0f, 0.0f);

            // sammelreucken berechnen
            sagrab(rn, M, m-1, P, einHori, -1, -1, ausVert);

            // relevanten ausgang fuer dieses p in aus setzen
            aus[p] = real(ausVert[P-p-1]);

            /*  produkt der relevanten transmissionsfaktoren fuer vertikale
                betrachtung bilden, unterschiedlich fuer jedes p! */

            // endwert setzen
            ende = m-P+p+1;
            if(ende<0) ende = 0;
```

```
                for(ii=m-1; ii>=ende; ii--)
                    faktor[p] *= sqrt(1 - rn[ii] * rn[ii]);
            }

        /*  zu jedem ausgang muss noch die reflexion 1. ordnung am graben
            mit dem zu bestimmenden reflexionsfaktor hinzufuegt werden. */

        // produkt der relevanten transmissionsfaktoren bilden.
        // hier: horizontale betrachtung (fuer alle p gleich)
        tt = 1.0;
        for(ii=0; ii<m; ii++)
            tt *= sqrt(1 - rn[ii] * rn[ii]);

        if(quad) {
            // quadratische betrachtung der ausgaenge

            /*  es ergbit sich bei quadratischer betrachtung fuer jede spalte
                folgende biquadratische gleichung, die es zu loesen gilt.

                (tt^2 * r^2 * faktor[0]^2 * i^2 + aus[0])^2 +
                (tt^2 * r^2 * faktor[1]^2 * i^2 + aus[1])^2 + ... = aE[m]^2

                mit a[x] = - tt^2 * faktor[x]^2 und b[x] = aus[x] ergibt sich

                (r^2 * a[x] + b[x])^2 + ... = aE[m]^2

                r^4 * a[x]^2 + 2 * r^2 * a[x] * b[x] + b[x]^2 + ...
                    - aE[m]^2 = 0

                somit ergibt sich fuer p und q in der loesungsformel fuer
                quadratische gleichungen nach substitution mit z = r^2
                folgendes:

                          pZ          a[0]b[0] + a[1]b[1] + ...
                p = 2 * ---- =  2 * -------------------------- = qp
                          pN            a[0]^2 + a[1]^2 + ...

                     b[0]^2 + b[1]^2 + ... -aE[m]^2
                q = ------------------------------- = qq
                                  pN

                nach dieser formel loesen wir nun die gleichung.

                z1/2 = -p / 2 +- sqrt(p^2 / 4 - q) = r1/2

                fuer r ergibt sich dann

                r1/2 = +- sqrt(z1) = rf[0]/rf[1] und
                r3/4 = +- sqrt(z2) = rf[2]/rf[3]
            */

            // biquadratische gleichung mit formel loesen
            pN = 0.0;
            pZ = 0.0;
            qq = 0.0;
            for(p=0; p<P; p++) {
                a = - tt * tt * faktor[p] * faktor[p];
                b = aus[p];

//printf("\ntt=%g; faktor=%g; a=%g; b=%g;", tt, faktor[p], a, b);

                pZ += a * b;
                pN += a * a;

                qq += b * b;
            }

            qp = 2 * pZ / pN;
            qq = (qq - aE[m] * aE[m]) / pN;
```

```
            // diskriminante bestimmen
            d = qp * qp / 4 - qq;

//printf("\np=%g; q=%g; d=%g; aE=%g;\n", qp, qq, d, aE[m]);

            // gleichung loesen
            if(d>=0.0) {
                // reelles ergebnis in substitutionsgleichung
                r1 = - qp / 2 - sqrt(d);
                r2 = - qp / 2 + sqrt(d);
            } else {
                // komplexes ergebnis in substitutionsgleichung!
                printf("\rreflexionsfaktor fuer spalte %d komplex."
                    "                                  \n", m+1);

                // entsprechende ergebnisse auf 0 setzen
                r1 = 0.0;
                r2 = 0.0;
            }

            // alle ergebnisse auf null setzen
            for(ii=0; ii<4; ii++) rf[ii] = 0.0;

            if(r1>0) {
                rf[0] = sqrt(r1);
                rf[1] = - sqrt(r1);
            } else {
                rf[0] = sqrt(-r1);
                rf[1] = - sqrt(-r1);
            }

            if(r2>0) {
                rf[2] = sqrt(r2);
                rf[3] = - sqrt(r2);
            } else {
                rf[2] = sqrt(-r2);
                rf[3] = - sqrt(-r2);
            }
        } else {
            // lineare betrachtung der ausgaenge
            zaehler = fabs(aE[m]);
            for(p=0; p<P; p++) zaehler -= aus[p];

            nenner = faktor[0] * faktor[0];
            for(p=1; p<P; p++) nenner += faktor[p] * faktor[p];
            nenner *= tt * tt;

            // diskriminante (unter der wurzel)
            d = zaehler / nenner;

            // auf zu grosses r pruefen
            if(fabs(d)>1) {
                printf("\rbetrag des reflexionsfaktors fuer spalte %d "
                    "groesser eins.                  \n", m+1);

                /*  wir brechen den ganzen spass hier ab,
                    ausgabedateien erstellen und beenden. */
                datei = fopen(ziel1, "w");
                fprintf(datei, "%d\n", m+1);
                fclose(datei);
                datei = fopen(ziel2, "w");
                fprintf(datei, "%d\n", m+1);
                fclose(datei);
                return -1;
            }

            if(d<0) {
                // komplexes ergebnis!
                printf("\rreflexionsfaktor fuer spalte %d komplex."
```

```
                    "                                   \n", m+1);

                // r wird null gesetzt
                rf[0] = 0.0;
                rf[1] = 0.0;
            } else {
                // reell!
                rf[0] = sqrt(d);
                rf[1] = - sqrt(d);
            }

            // nicht verwendete ergebnisse fuer r im vgl. zur quadr. betr.
            rf[2] = 100.0;
            rf[3] = 100.0;
        }

        /*  wir nehmen das ergebnis fuer r, welches am naechsten bei dem
            zugehoerigen aE liegt -> kleinstes delta suchen */
        delta_min = 100;
        index = -1;
        for(ii=0; ii<4; ii++) {
            delta = fabs(rf[ii] - aE[m]);

            if(delta<delta_min) {
                delta_min = delta;
                index = ii;
            }
        }

        // ergebnis zuweisen
           rn[m] = rf[index];

        // auf maximalwert pruefen
        if(rn[m]>rmax) {
            rmax = rn[m];
            idxmax = m;
        }

        // auf minimalwert pruefen
        if(rn[m]<rmin) {
            rmin = rn[m];
            idxmin = m;
        }

        // ausgeben wie weit wir sind
        printf("\r%4d von %4d [r=%6.4f%%, rmin(%d)=%6.4f%%, "
            "rmax(%d)=%6.4f%%] fertig. ", m+1, M, 100*rn[m],
            idxmin+1, 100*rmin, idxmax+1, 100*rmax);

//if(m==9) break;
    }

    // ausgabe der zeit und der extremwerte
    datum(datum_str);
    zeit(zeit_str);
    printf("\rende der berechnung am %s um %s uhr.                          \n",
        datum_str, zeit_str);
    printf("rmin = %f%% bei index %d, rmax = %f%% bei index %d\n",
        100*rmin, idxmin+1, 100*rmax, idxmax+1);

    // speichern (streu)
    printf("\n-> speichere reflexionsfaktoren (streu) in '%s'...", ziel1);
    datei = fopen(ziel1, "w");

    // hier muessen alle r positiv sein
    for(m=0; m<M; m++) fprintf(datei, "%g\n", fabs(rn[m]));

    fclose(datei);
    printf(" OK.");
```

```
    // speichern (sammel)
    printf("\n-> speichere reflexionsfaktoren (sammel) in '%s'...", ziel2);
    datei = fopen(ziel2, "w");

    // hier mit vorzeichen
    for(m=0; m<M; m++) fprintf(datei, "%g\n", rn[m]);

    fclose(datei);
    printf(" OK.\n\n");

    return 0;
}
```

D.4. aus.cpp

```
/*  berechnung der vertikalen ausgaenge des sammelgrabens nach
    verschiedenen methoden (parameter).

    Michael Becker, 11.02.2000 - 27.04.2000 */

//---------------------------------------------------------------------------
#include <vcl\condefs.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include <d:\studium\diplom\cpp\include\stgrab.h>
#include <d:\studium\diplom\cpp\include\sagrab.h>
#include <d:\studium\diplom\cpp\include\mytime.h>

#pragma hdrstop
USERES("aus.res");

//---------------------------------------------------------------------------
const int M_MAX = 7000;
const int P_MAX = 8;
const int SCHRITTE_MAX = 401;
const complex i(0, 1);

//---------------------------------------------------------------------------
// testfunktionen
void sammel();
void streu();

//---------------------------------------------------------------------------
int main(int argc, char **argv) {

    // chararray fuer quelldatei1 (reflexionsfaktoren streuruecken)
    char quelle1[80];

    // chararray fuer quelldatei2 (reflexionsfaktoren sammelruecken)
    char quelle2[80];

    // chararray fuer zieldatei
    char ziel[80];

    // chararray fuer horizontale ausgaenge des senderueckens
    char ziel_aushori[80];

    // aktuelle zeit und aktuelles datum
    char zeit_str[79];
    char datum_str[79];

    // true, wenn beide ruecken berechnet werden, false = nur sammel
    bool alles = true;
```

```
// -- konstanten --

// mittenwellenlaenge
double l0 = 1.3e-6;

// von wellenlänge, bis wellenlänge, schritte
double start;
double schritte = SCHRITTE_MAX;
double ende;

// gitterhoehe
int P = 8;

// gitterbreite
int M = 0, M1 = 0;

// array fuer reflexionsfaktoren streu
double rstreu[M_MAX];

// array fuer reflexionsfaktoren sammel
double rsammel[M_MAX];

// zeilenvektor fuer wellenlaengen
double lambdas[SCHRITTE_MAX];

// ergebnismatrix fuer erst quadrieren, dann summieren
double eqds[3][SCHRITTE_MAX];

// ergebnismatrix fuer erst summieren, dann quadrieren
double esdq[3][SCHRITTE_MAX];

// vertikale eingaenge fuer streugraben
complex einVert[P_MAX];

// eingaenge fuer sammelgraben
complex ausEinHori[M_MAX+P_MAX-1];

// ausgaenge des sammelgrabens
complex aus[P_MAX];
complex aus_eqds;
complex aus_esdq;

// zeiger auf datei
FILE *datei;

// hilfsvariablen
double value, delta, l, max_eqds, max_esdq;
int c, count;

// -- testen? --
if(argc==2) {
    if(strcmp(argv[1], "sammel")==0) sammel();
    if(strcmp(argv[1], "streu")==0) streu();
    return 1;
}

// -- pruefen, ob erforderliche parameter uebergeben wurden --
if(argc!=3 && argc!=5 && argc!=7) {
    // falsche anzahl oder fehlende parameter
    printf("\n (1) aus <inputdatei> <outputdatei>");
    printf("\n (2) aus <inputdatei1> <inputdatei2> <outputdatei> ");
    printf("<output_hori>\n (3) aus <rueckenbreite> ");
    printf("<rstart> <anz_pi> <kaiser_beta> <si_max> <l/q>\n");
    printf("  <rstart=-1>: nur sammelruecken, ");
    printf("sonst streu- und sammelruecken berechnen\n");
    printf("  <rstart=-2>: streu- und sammelruecken fuer ainr2 ");
    printf("berechnen\n");
    return -1;
} else {
    if(argc==3) {
```

```
    // nur sammel
    alles = false;

    strcpy(quelle2, argv[1]);
    strcpy(ziel, argv[2]);
}

if(argc==5) {
    strcpy(quelle1, argv[1]);
    strcpy(quelle2, argv[2]);
    strcpy(ziel, argv[3]);
    strcpy(ziel_aushori, argv[4]);
}

if(argc==7) {
    // zieldatei
    strcpy(ziel, argv[3]);
    strcat(ziel, "pi\\");
    strcat(ziel, argv[1]);
    strcat(ziel, "\\");
    strcat(ziel, argv[4]);

    // zieldatei fuer horizontale ausgaenge
    strcpy(ziel_aushori, ziel);
    strcat(ziel_aushori, "\\e\\");

    // nochmal (haut-)zieldatei
    strcat(ziel, "\\p\\");

    if(strcmp(argv[2], "-1")!=0 && strcmp(argv[2], "-2")!=0) {
        // streu und sammel mit unterschiedlicher wichtung

        // datei mit reflexionsfaktoren streu
        strcpy(quelle1, argv[3]);
        strcat(quelle1, "pi\\");
        strcat(quelle1, argv[1]);
        strcat(quelle1, "\\streu\\r\\");
        strcat(quelle1, argv[2]);

        // datei mit reflexionsfaktoren sammel
        strcpy(quelle2, argv[3]);
        strcat(quelle2, "pi\\");
        strcat(quelle2, argv[1]);
        strcat(quelle2, "\\");
        strcat(quelle2, argv[4]);
        strcat(quelle2, "\\sammel\\r\\");
        strcat(quelle2, argv[5]);
        strcat(quelle2, argv[6]);

        // richtige erweiterung der zieldateien
        strcat(ziel, argv[5]);
        strcat(ziel, argv[6]);
        strcat(ziel, "-");
        strcat(ziel, argv[2]);

        strcat(ziel_aushori, argv[5]);
        strcat(ziel_aushori, argv[6]);
        strcat(ziel_aushori, "-");
        strcat(ziel_aushori, argv[2]);
    } else {
        if(strcmp(argv[2], "-1")==0) {
            // nur sammelruecken
            alles = false;

            // datei mit reflexionsfaktoren sammel
            strcpy(quelle2, argv[3]);
            strcat(quelle2, "pi\\");
            strcat(quelle2, argv[1]);
            strcat(quelle2, "\\");
            strcat(quelle2, argv[4]);
```

```
                    strcat(quelle2, "\\sammel\\r\\");
                    strcat(quelle2, argv[5]);
                    strcat(quelle2, argv[6]);

                    // richtige erweiterung der zieldatei
                    strcat(ziel, argv[5]);
                    strcat(ziel, argv[6]);
                }

                if(strcmp(argv[2], "-2")==0) {
                    // ausgaenge fuer ainr2 berechnen
                    // gemeinsame angaben quell- und zieldateien am anfang:

                    // datei mit reflexionsfaktoren streu
                    strcpy(quelle1, argv[3]);
                    strcat(quelle1, "pi\\");
                    strcat(quelle1, argv[1]);
                    strcat(quelle1, "\\");
                    strcat(quelle1, argv[4]);
                    strcat(quelle1, "\\streu\\r2\\");
                    strcat(quelle1, argv[5]);
                    strcat(quelle1, argv[6]);

                    // datei mit reflexionsfaktoren sammel
                    strcpy(quelle2, argv[3]);
                    strcat(quelle2, "pi\\");
                    strcat(quelle2, argv[1]);
                    strcat(quelle2, "\\");
                    strcat(quelle2, argv[4]);
                    strcat(quelle2, "\\sammel\\r2\\");
                    strcat(quelle2, argv[5]);
                    strcat(quelle2, argv[6]);

                    // zieldatei
                    strcpy(ziel, argv[3]);
                    strcat(ziel, "pi\\");
                    strcat(ziel, argv[1]);
                    strcat(ziel, "\\");
                    strcat(ziel, argv[4]);

                    // zieldatei fuer horizontale ausgaenge
                    strcpy(ziel_aushori, ziel);
                    strcat(ziel_aushori, "\\e2\\");
                    strcat(ziel_aushori, argv[5]);
                    strcat(ziel_aushori, argv[6]);

                    // nochmal (haupt-)zieldatei
                    strcat(ziel, "\\p2\\");
                    strcat(ziel, argv[5]);
                    strcat(ziel, argv[6]);
                }
            }
        }
    }
/*
printf("quelle1:      %s\n", quelle1);
printf("quelle2:      %s\n", quelle2);
printf("ziel:         %s\n", ziel);
printf("ziel_aushori: %s\n", ziel);
if(alles) printf("alles\n"); else printf("nur sammel\n");
return -1;
*/
    // -- begruessung --
    printf("\n\n    Darstellung der Ausgangsleistung des Sammelgrabens in\n");
    printf("  Abhaengigkeit von der Wellenlaenge des einfallenden Lichts\n");
    printf(" ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

    // -- laden der erforderlichen daten --
    // reflexionsfaktoren fuer sammelgraben
    printf("-> lade reflexionsfaktoren (sammel) aus '%s'...", quelle2);
```

```
datei = fopen(quelle2, "r");

while(((c=fscanf(datei, "%lf", &value))!=EOF) && c>0) {
    if(M==M_MAX) {
        printf("\naus: RUECKENBREITE ZU GROSS! (MAX=%d)", M_MAX);
        printf("\n     RESTLICHE WERTE WERDEN NICHT VERWENDET.\n");
        scanf("%d", &c);
        c = 1;
        break;
    }

    rsammel[M++] = value;
}

fclose(datei);
if(c==-1) printf(" OK.\n");

if(alles) {
    // reflexionsfaktoren fuer streugraben
    printf("-> lade reflexionsfaktoren (streu) aus '%s'...", quelle1);
    datei = fopen(quelle1, "r");

    while(((c=fscanf(datei, "%lf", &value))!=EOF) && c>0) {
        if(M1==M_MAX) {
            printf("\naus: RUECKENBREITE ZU GROSS! (MAX=%d)", M_MAX);
            printf("\n     RESTLICHE WERTE WERDEN NICHT VERWENDET.\n");
            scanf("%d", &c);
            c = 1;
            break;
        }

        rstreu[M1++] = value;
    }

    fclose(datei);

    if(M1!=M) {
        printf("\nRUECKENBREITE SAMMEL VERSCHIEDEN RUECKENBREITE STREU.\n");
        return -1;
    }

    if(c==-1) printf(" OK.\n");
}

// -- auf mindestlaenge pruefen --
if(M<P) {
    /*  zu kurzer ruecken, wir brechen den ganzen spass
        hier ab, ausgabedatei erstellen und beenden. */
    datei = fopen(ziel, "w");
    fprintf(datei, "zu wenig werte\n");
    fclose(datei);
    return -1;
}

// -- rechnen --
datum(datum_str);
zeit(zeit_str);
printf("\nbeginn der berechnung am %s um %s uhr.\n", datum_str, zeit_str);

// startwert, endwert und eingeange setzen
if(alles) {
    start = 1295;
    ende = 1305;

    /*  -- horizontale ausgaenge des streugrabens abspeichern --
        hier: ausgabedatei oeffnen */
    datei = fopen(ziel_aushori, "w");
} else {
    start = 1290;
    ende = 1310;
```

```
}

// differenzwellenlaenge bestimmen
delta = (ende - start) / (schritte - 1);

for(count=0; count<schritte; count++) {
    /*  aktuelle wellenlänge bestimmen und
        in ergebnistabelle eintragen */
    l = start + delta * count;
    lambdas[count] = l;
    l = l * 1e-9;

    // -- berechnung fuer erst quadrieren, dann summieren eqds --
    if(alles) {
        // eingaenge fuer streugraben vorbelegen
        for(c=0; c<P; c++)
            einVert[c] = complex(1.0f / sqrt(P), 0.0f);

        // streugraben berechnen
        stgrab(rstreu, M, M-1, P, einVert, l0, l, ausEinHori);

        // -- horizontale ausgaenge des streugrabens abspeichern --
        for(c=0; c<M+P-1; c++) {
            // fuer jeden ausgang zuerst den real-, dann den imaginärteil
            fprintf(datei, "%g\n%g\n", real(ausEinHori[c]),
                imag(ausEinHori[c]));
        }
    } else {
        // eingaenge fuer sammelgraben vorbelegen
        for(c=0; c<M+P-1; c++)
            ausEinHori[c] = complex(1.0f / sqrt(M), 0.0f);
    }

    // sammelgraben berechnen
    sagrab(rsammel, M, M-1, P, ausEinHori, l0, l, aus);

    // ausgangsleistung bestimmen
    aus_eqds = complex(0.0f, 0.0f);
    for(c=0; c<P; c++) aus_eqds += aus[c] * aus[c];

    // ergebnisse zuweisen -> betrag der leistung
    eqds[0][count] = abs(aus_eqds);

    // ergebnisse zuweisen -> phase der leistung
    eqds[2][count] = 180 * arg(aus_eqds) / M_PI;

    // -- berechnung fuer erst summieren, dann quadrieren esdq --
    if(alles) {
        // eingaenge fuer streugraben vorbelegen
        for(c=0; c<P; c++)
            einVert[c] = complex(1.0f / P, 0.0f);

        // streugraben berechnen
        stgrab(rstreu, M, M-1, P, einVert, l0, l, ausEinHori);
    } else {
        // eingaenge fuer sammelgraben vorbelegen
        for(c=0; c<M+P-1; c++)
            ausEinHori[c] = complex(1.0f / M, 0.0f);
    }

    // sammelgraben berechnen
    sagrab(rsammel, M, M-1, P, ausEinHori, l0, l, aus);

    // ausgangsleistung bestimmen
    aus_esdq = complex(0.0f, 0.0f);
    for(c=0; c<P; c++) aus_esdq += aus[c];

    // ergebnisse zuweisen -> betrag der leistung
    esdq[0][count] = abs(aus_esdq * aus_esdq);
```

```
    // ergebnisse zuweisen -> phase der leistung
    esdq[2][count] = 180 * arg(aus_esdq * aus_esdq) / M_PI;

    // -- ausgeben, wie weit wir sind --
    printf("\r%4d von %4.0f [%7.2fnm, eqds=%7.5f%%, esdq=%7.5f%%] fertig.  ",
        count+1, schritte, l*1e9, eqds[0][count]*100, esdq[0][count]*100);
}

// maxima der leistungen finden
max_eqds = -10;
max_esdq = -10;
for(count=0; count<schritte; count++) {
    if(eqds[0][count]>max_eqds) {
        max_eqds = eqds[0][count];
    }

    if(esdq[0][count]>max_esdq) {
        max_esdq = esdq[0][count];
    }
}

for(count=0; count<schritte; count++) {
    // relative leistung absolut
    eqds[1][count] = eqds[0][count] / max_eqds;
    esdq[1][count] = esdq[0][count] / max_esdq;

    // relative leistung logarithmisch
    eqds[1][count] = 10 * log10(fabs(eqds[1][count]));
    esdq[1][count] = 10 * log10(fabs(esdq[1][count]));
}

// ausgabe der zeit
datum(datum_str);
zeit(zeit_str);
printf("\rende der berechnung am %s um %s uhr.                        \n",
    datum_str, zeit_str);

if(alles) {
    /*  -- horizontale ausgaenge des streugrabens abspeichern --
        abschließend die wellenlaengen sichern und P (gitterhoehe)
        [ die ist wichtig fuer die bestimmung der schrittanzahl beim
        plotten mit matlab zur visualisierung der daten ] */

    // wellenlaengen
    for(count=0; count<schritte; count++)
        fprintf(datei, "%g\n", lambdas[count]);

    // gitterhoehe P
    fprintf(datei, "%i.0\n", P);

    fclose(datei);

    // ausgabe der information ueber speicherung
    printf("\n-> horizontale streu-ausgaenge nach '%s' gespeichert.",
        ziel_aushori);
}

// ergebnisse speichern
printf("\n-> speichere ergebnisse in '%s'...", ziel);
datei = fopen(ziel, "w");

// wellenlaengen
for(count=0; count<schritte; count++)
    fprintf(datei, "%g\n", lambdas[count]);

for(c=0; c<3; c++) {
    /*  c=0:    absolute leistung
        c=1:    relative leistung logarithmisch
        c=2:    phase der leistung */
    for(count=0; count<schritte; count++)
```

```
            fprintf(datei, "%g\n", eqds[c][count]);

        for(count=0; count<schritte; count++)
            fprintf(datei, "%g\n", esdq[c][count]);
    }

    fclose(datei);
    printf(" OK.\n\n");

    return 0;
}


//------------------------------------------------------------------------
// sammelruecken testen
void sammel() {

    int c;

    double rn[M_MAX];
    int M = 5;
    int P = 3;
    complex E[M_MAX];
    complex Evorn[P_MAX-1];
    complex aus[P_MAX];
    complex trans[P_MAX];

    for(c=0; c<M; c++) {
        rn[c] = .1;
        E[c] = 1 / i;

        if(c<P-1) Evorn[c] = 1 / i;
    }

    sagrab(rn, M, M-1, P, E, Evorn, -1, -1, aus, trans);

    for(c=0; c<P; c++)
        printf("aus[%i]   = (%g, %g)\n", c, real(aus[c]), imag(aus[c]));

    for(c=0; c<P; c++)
        printf("trans[%i] = (%g, %g)\n", c, real(trans[c]), imag(trans[c]));

    scanf("%d", &c);
}


//------------------------------------------------------------------------
// streuruecken testen
void streu() {

    int c;

    double rn[M_MAX];
    int M = 5;
    int P = 3;
    complex ein[P_MAX];
    complex E[M_MAX];
    complex Evorn[P_MAX-1];

    for(c=0; c<M; c++) {
        rn[c] = .1;

        if(c<P) ein[c] = 1 / sqrt(P);
    }

    stgrab(rn, M, P, ein, -1, -1, E, Evorn);

    for(c=0; c<P-1; c++)
        printf("Evorn[%i] = (%g, %g);\n", c, real(Evorn[c]), imag(Evorn[c]));
```

```
    for(c=0; c<M; c++)
        printf("E[%i]     = (%g, %g);\n", c, real(E[c]), imag(E[c]));

    scanf("%d", &c);
}
```

D.5. mytime.h

```
/*  Michael Becker, 12.02.2000 - 12.02.2000 */

//-----------------------------------------------------------------------------
#include <time.h>

//-----------------------------------------------------------------------------
/*  gibt im character array jetzt der laenge 79 das aktuelle
    datum in der form "TT:MM:JJJJ" zurueck. */
void datum(char *jetzt) {

    struct tm *time_now;
    time_t secs_now;

    time(&secs_now);
    time_now = localtime(&secs_now);
    strftime(jetzt, 79,  "%d.%m.%Y", time_now);
}


//-----------------------------------------------------------------------------
/*  gibt im character array jetzt der laenge 79 das aktuelle
    datum und die uhrzeit in der form "TT:MM:JJJJ HH:MM:SS" zurueck. */
void datum_und_zeit(char *jetzt) {

    struct tm *time_now;
    time_t secs_now;

    time(&secs_now);
    time_now = localtime(&secs_now);
    strftime(jetzt, 79,  "%d.%m.%Y %H:%M:%S", time_now);
}


//-----------------------------------------------------------------------------
/*  gibt im character array jetzt der laenge 79 die aktuelle
    uhrzeit in der form "HH:MM:SS" zurueck. */
void zeit(char *jetzt) {

    struct tm *time_now;
    time_t secs_now;

    time(&secs_now);
    time_now = localtime(&secs_now);
    strftime(jetzt, 79,  "%H:%M:%S", time_now);
}
```

D.6. sagrab.h

```
/*  Michael Becker, 11.02.2000 - 28.02.2000 */

#include <complex.h>
#include <d:\studium\diplom\cpp\include\sazelle.h>
```

```
const SAGRAB_MMAX = 7000;
const SAGRAB_PMAX = 8;

//-------------------------------------------------------------------------
// berechnung der ausgaenge des sammelrueckens bei schraegen graeben
void sagrab(
    // zu uebergebende parameter
        double *rn,              // reflexionsfaktoren der spalten
        int M,                   // breite des rueckens
        int Mmax,                // bis zu welchem index (spalte, 0<=Mmax<M)
        int P,                   // gitterhoehe
        complex *E,              // eingaenge der untersten zeile
        complex *Evorn,          // eingaenge der linken vorderen zellen
        double L,                // laserwellenlaenge
        double l,                // betrachtete wellenlaenge
    // zeiger auf die ergebnisfelder
        complex *ausVert,        // zeiger auf vertikale linke ausgaenge
        complex *trans) {        // zeiger auf transmissionen (letzter graben)

    // laufvariablen
    int m, p;

    // -- definieren der 4 relevanten arrays fuer eine zeile --

    // eingaenge
    complex A2[SAGRAB_MMAX+SAGRAB_PMAX-1];
    complex A3[SAGRAB_MMAX+SAGRAB_PMAX-1];

    // ausgaenge
    complex B1[SAGRAB_MMAX+SAGRAB_PMAX-1];
    complex B4[SAGRAB_MMAX+SAGRAB_PMAX-1];

    // ergebnisse fuer eine zellenberechnung
    complex B1B4[2];

    // -- parameter pruefen --
    if(M>SAGRAB_MMAX || P>SAGRAB_PMAX || Mmax>=M) {
        printf("\nsagrab: FEHLER IN PARAMETERUEBERGABE!");
        scanf("%d", &m);
        return;
    }

    // -- eingaenge der untersten zeile vorbelegen --
    // grabenfreie zellen
    for(m=0; m<=P-2; m++) A2[m] = Evorn[m];

    // normale zellen
    for(m=0; m<M; m++) A2[m+P-1] = E[m];

    // -- berechnung von unten nach oben und jeweils von rechts nach links --
    for(p=0; p<P; p++) {
        // rechten eingang der ersten verwendeten zelle auf null setzen
        A3[Mmax+P-p-1] = complex(0.0f, 0.0f);

        // normale rueckenzellen in dieser zeile
        for(m=Mmax+P-p-1; m>=P-p-1; m--) {
            sazelle(A2[m], A3[m], rn[m-P+p+1], L, l, B1B4);
            B1[m] = B1B4[0];
            B4[m] = B1B4[1];

            /*  eingaengen der benachbarten zellen die ausgaenge
                der aktuellen zelle zuweisen */

            // spalte links daneben
            if(m>0)    A3[m-1] = B1[m];

            // darueberliegende zeile
            A2[m] = B4[m];
        }
```

```
        // drehung in den grabenfreien zellen (vorn links) nachtraeglich
        // berechnen, weil diese oben nicht beruecksichtigt wurden
        for(m=P-p-2; m>=0; m--) {
            sazelle(A2[m], A3[m], 0, L, l, B1B4);
            B1[m] = B1B4[0];
            B4[m] = B1B4[1];

            /*  eingaengen der benachbarten zellen die ausgaenge
                der aktuellen zelle zuweisen */

            // spalte links daneben
            if(m>0)    A3[m-1] = B1[m];

            // darueberliegende zeile
            A2[m] = B4[m];
        }

        // -- ergebnisse zuweisen --
        // vordere ausgaenge
        ausVert[p] = B1[0];

        // nach oben transmittierte werte in der letzten spalte
        trans[p] = B4[Mmax+P-p-1];
    }
}


//------------------------------------------------------------------------
// berechnung der ausgaenge des sammelrueckens bei schraegen graeben
void sagrab(
    // zu uebergebende parameter
        double *rn,              // reflexionsfaktoren der spalten
        int M,                   // breite des rueckens
        int Mmax,                // bis zu welchem index (spalte, 0<=Mmax<M)
        int P,                   // gitterhoehe
        complex *einHori,        // alle eingaenge der untersten zeile
        double L,                // laserwellenlaenge
        double l,                // betrachtete wellenlaenge
    // zeiger auf die ergebnisfelder
        complex *aus) {          // zeiger auf vertikale linke ausgaenge

    // ungenutzte ausgaenge definieren
    complex trans[SAGRAB_PMAX];

    // neue eingabefelder definieren
    complex E[SAGRAB_MMAX];
    complex Evorn[SAGRAB_PMAX-1];

    // laufvariable
    int m;

    // -- eingaenge aufteilen --
    // grabenfreie zellen
    for(m=0; m<=P-2; m++) Evorn[m] = einHori[m];

    // normale zellen
    for(m=P-1; m<M+P-1; m++) E[m-P+1] = einHori[m];

    // eigentliche funktion aufrufen
    sagrab(rn, M, Mmax, P, E, Evorn, L, l, aus, trans);
}
```

D.7. sazelle.h

```
/*  Michael Becker, 11.02.2000 - 12.02.2000 */
```

```
#include <complex.h>
#include <math.h>

//---------------------------------------------------------------------------
// berechnung der ausgangsgroessen fuer eine zelle des empfangsrueckens
void sazelle(
    // zu uebergebende parameter
        complex A2,               // eingang unten
        complex A3,               // eingang rechts
        double rd,                // reflexionsfaktor
        double L,                 // laserwellenlaenge
        double l,                 // betrachtete wellenlaenge
    // zeiger auf ergebnisfeld
        complex *B1B4) {          // zeiger auf ergebnis-array

    // faktor fuer frequenzbetrachtung
    complex faktor(1.0f, 0.0f);

    // transmissionsfaktor bestimmen
    double t = sqrt(1 - fabs(rd) * fabs(rd));
    // reflexionsfaktor um 90 grad drehen
    complex r = rd * complex(0.0f, 1.0f);

    // frequenz- bzw. wellenlaengenbetrachtung?
    if(L!=-1 && l!=-1) {
        // ja (mit drehung)
        faktor = polar(1, -2 * M_PI * L / l);
    }

    // berechnung
    B1B4[0] = faktor * (r * A2 + t * A3);
    B1B4[1] = faktor * (t * A2 + r * A3);
}
```

D.8. stgrab.h

```
/*  Michael Becker, 13.02.2000 - 28.02.2000 */

#include <complex.h>
#include <d:\studium\diplom\cpp\include\stzelle.h>

const STGRAB_MMAX = 7000;
const STGRAB_PMAX = 8;

//---------------------------------------------------------------------------
// berechnung der ausgaenge des sammelrueckens bei schraegen graeben
void stgrab(
    // zu uebergebende parameter
        double *rn,               // reflexionsfaktoren der spalten
        int M,                    // breite des rueckens
        int Mmax,                 // bis zu welchem index (spalte, 0<=Mmax<M)
        int P,                    // gitterhoehe
        complex *ein,             // vertikale eingaenge (links)
        double L,                 // laserwellenlaenge
        double l,                 // betrachtete wellenlaenge
    // zeiger auf die ergebnisfelder
        complex *E,               // obere ausgaenge der "normalen" zellen
        complex *Evorn,           // obere vordere ausgaenge
        complex *ausHori,         // obere ausgaenge aller zellen
        complex *ausVert) {       // rechte ausgaenge

    // laufvariablen
    int m, p;

    // -- definieren der 4 relevanten arrays fuer eine spalte --
```

```
    // eingaenge
    complex A1[STGRAB_PMAX];
    complex A2[STGRAB_PMAX];

    // ausgaenge
    complex B3[STGRAB_PMAX];
    complex B4[STGRAB_PMAX];

    // aktuell verwendeter refelxionsfaktor
    double r;

    // ergebnisse fuer eine zellenberechnung
    complex B3B4[2];

    // -- parameter pruefen --
    if(P>STGRAB_PMAX || Mmax>=M) {
        printf("\nstgrab: FEHLER IN PARAMETERUEBERGABE!");
        scanf("%d", &m);
        return;
    }

    // -- eingaenge vorbelegen --

    // vertikalte eingaenge der ersten spalte vorbelegen
    for(p=0; p<P; p++) A1[p] = ein[p];

    // unteren eingang auf null setzen
    A2[0] = complex(0.0f, 0.0f);

    // -- berechnung von links nach rechts und jeweils von unten nach oben --
    for(m=0; m<Mmax+P; m++) {
        // rueckenzellen in dieser spalte
        for(p=0; p<P; p++) {
            // aktuellen reflexionsfaktor setzen
            if(m-p<0 || m-p>Mmax) {
                // wir sind in einer linken/rechten grabenfreien zelle
                r = 0;
            } else
                r = rn[m-p];

            stzelle(A1[p], A2[p], r, L, l, B3B4);
            B3[p] = B3B4[0];
            B4[p] = B3B4[1];

            /*  eingaengen der benachbarten zellen die ausgaenge
                der aktuellen zelle zuweisen */

            // spalte rechts daneben
            A1[p] = B3[p];

            // darueberliegende zeile
            if(p<P-1) A2[p+1] = B4[p];
        }

        // -- ergebnisse zuweisen --

        // getrennte rueckgabe vordere und normale zellen
        if(m<P-1)
            Evorn[m] = B4[P-1];
        else
            E[m-P+1] = B4[P-1];

        // rueckgabe alle horizontalen ausgaenge
        ausHori[m] = B4[P-1];
    }

    // vertikale ausgaenge der letzten spalte
    for(p=0; p<P; p++) ausVert[p] = B3[p];
}
```

```
//------------------------------------------------------------------------
// berechnung der ausgaenge des sammelrueckens bei schraegen graeben
// alternativaufruf zur obigen funktion, berechnung des gesamten rueckens
void stgrab(
    // zu uebergebende parameter
        double *rn,              // reflexionsfaktoren der spalten
        int M,                   // breite des rueckens
        int P,                   // gitterhoehe
        complex *ein,            // vertikale eingaenge (links)
        double L,                // laserwellenlaenge
        double l,                // betrachtete wellenlaenge
    // zeiger auf die ergebnisfelder
        complex *E,              // obere ausgaenge der "normalen" zellen
        complex *Evorn) {        // obere vordere ausgaenge

    // nicht verwendete ergebnisfelder definieren
    complex ausHori[STGRAB_MMAX+STGRAB_PMAX-1];
    complex ausVert[STGRAB_PMAX];

    // eigentliche funktion aufrufen
    stgrab(rn, M, M-1, P, ein, L, l, E, Evorn, ausHori, ausVert);
}


//------------------------------------------------------------------------
// berechnung der ausgaenge des sammelrueckens bei schraegen graeben
// alternativaufruf zur obigen funktion, fuer ainr2
void stgrab(
    // zu uebergebende parameter
        double *rn,              // reflexionsfaktoren der spalten
        int M,                   // breite des rueckens
        int Mmax,                // bis zu welchem index (spalte, 0<=Mmax<M)
        int P,                   // gitterhoehe
        complex *ein,            // vertikale eingaenge (links)
        double L,                // laserwellenlaenge
        double l,                // betrachtete wellenlaenge
    // zeiger auf die ergebnisfelder
        complex *ausHori) {       // obere ausgaenge aller zellen

    // nicht verwendete ergebnisfelder definieren
    complex E[STGRAB_MMAX];
    complex Evorn[STGRAB_PMAX-1];
    complex ausVert[STGRAB_PMAX];

    // eigentliche funktion aufrufen
    stgrab(rn, M, Mmax, P, ein, L, l, E, Evorn, ausHori, ausVert);
}
```

D.9. stzelle.h

```
/*  Michael Becker, 13.02.2000 - 13.02.2000 */

#include <complex.h>
#include <math.h>

//------------------------------------------------------------------------
// berechnung der ausgangsgroessen fuer eine zelle des senderueckens
void stzelle(
    // zu uebergebende parameter
        complex A1,              // eingang links
        complex A2,              // eingang unten
        double rd,               // reflexionsfaktor
        double L,                // laserwellenlaenge
        double l,                // betrachtete wellenlaenge
    // zeiger auf ergebnisfeld
```

```
        complex *B3B4) {          // zeiger auf ergebnis-array

    // faktor fuer frequenzbetrachtung
    complex faktor(1.0f, 0.0f);

    // transmissionsfaktor bestimmen
    double t = sqrt(1 - fabs(rd) * fabs(rd));

    // reflexionsfaktor um 90 grad drehen
    complex r = rd * complex(0.0f, 1.0f);

    // frequenz- bzw. wellenlaengenbetrachtung?
    if(L!=-1 && l!=-1) {
        // ja (mit drehung)
        faktor = polar(1, -2 * M_PI * L / l);
    }

    // berechnung
    B3B4[0] = faktor * (t * A1 + r * A2);
    B3B4[1] = faktor * (r * A1 + t * A2);
}
```

E. Inhalte der Batch-Dateien

E.1. ainrfind.bat

```
@echo off
echo "ainrfind <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <delta_start>"
echo.

java -cp mibec.jar mibec.diplom.AinrFinder 137.193.213.95 %2pi\%1\%3\a %4 %5
%2pi\%1\%3\sammel\r %2pi\%1\%3\p
```

E.2. ainr2find.bat

```
@echo off
echo "ainrfind <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <delta_start>"
echo.

java -cp mibec.jar mibec.diplom.Ainr2Finder 137.193.213.95 %2pi\%1\%3\a %4 %5
%2pi\%1\%3\streu\r2 %2pi\%1\%3\sammel\r2 %2pi\%1\%3\p2
```

E.3. sjainr.bat

```
@echo off
echo "sjainr <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <l/q>"
echo.

java -cp mibec.jar mibec.share.Job ainr.exe 1 1 1 %2pi\%1\%3\a\%4
%2pi\%1\%3\sammel\r\%4%5 %5
```

E.4. sjainr2.bat

```
@echo off
echo "sjainr2 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <l/q>"
echo.

java -cp mibec.jar mibec.share.Job ainr2.exe 1 2 1 %2pi\%1\%3\a\%4
%2pi\%1\%3\streu\r2\%4%5 %2pi\%1\%3\sammel\r2\%4%5 %5
```

E.5. sjaus.bat

```
@echo off
echo "sjaus <rueckenbreite> <rstart> <anz_pi> <kaiser_beta> <si_max> <l/q>"
echo.

java -cp mibec.jar mibec.share.Job aus.exe 2 2 0 %3pi\%1\streu\r\%2
%3pi\%1\%4\sammel\r\%5%6 %3pi\%1\%4\p\%5%6-%2 %3pi\%1\%4\e\%5%6-%2
```

E.6. sjaus-1.bat

```
@echo off
echo "sjaus-1 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <l/q>"
echo.

java -cp mibec.jar mibec.share.Job aus.exe 1 1 0
%2pi\%1\%3\sammel\r\%4%5 %2pi\%1\%3\p\%4%5
```

E.7. sjaus-2.bat

```
@echo off
echo "sjaus-2 <rueckenbreite> <anz_pi> <kaiser_beta> <si_max> <l/q>"
echo.

java -cp mibec.jar mibec.share.Job aus.exe 2 2 0 %2pi\%1\%3\streu\r2\%4%5
%2pi\%1\%3\sammel\r2\%4%5 %2pi\%1\%3\p2\%4%5 %2pi\%1\%3\e2\%4%5
```

Erklärung

Ich versichere, daß ich die vorliegende Diplomarbeit mit dem Thema „Dimensionierung und Optimierung eines optischen Wellenfilters“ selbständig angefertigt und alle mir zuteil gewordenen Hilfen sowie die verwendete Literatur am Ende der Arbeit angegeben habe.

Neubiberg, 27. April 2000

Michael Becker

Danksagung

Zu guter letzt möchte ich mich bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

Herrn Prof. Dr.-Ing. Udo Barabas danke ich für die freundliche Überlassung des Themas und die Möglichkeit, Einblick in die Forschungsarbeit an seinem Institut nehmen zu können.

Besonders danke ich Herrn Dipl.-Ing. Peter Crassen Hruschka für die fürsorgliche Betreuung und die wichtigen Hinweise sowie die oft stundenlangen Gespräche, die das Gedeihen dieser Arbeit sehr unterstützt haben.

Maxime Ahoyo, Seongjae Boo, Thomas Chladek, Peter Hruschka, Rainer Schmeil, Markus Schnöll und Michael Teubert gilt mein Dank für die Bereitstellung ihrer Computer als Clients für das verteilte Rechnen.