TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Diplomarbeit

zum Erlangen des akademischen Grades

Diplom-Ingenieur

(Dipl.-Ing.)

Untersuchung und Implementierung numerischer Optimierungsalgorithmen zur Lösung inverser elektromagnetischer Streuprobleme

eingereicht von Christian Stüllein
geboren am 27.11.1978 in Meiningen

Matrikel-Nummer	3186873
Seminargruppe	IT02
1. Hochschulbetreuer	Dr.-Ing. Dirk Plettemeier
2. Hochschulbetreuer	Dipl.-Ing. Diemo Landmann

Datum der Themenausgabe	14.02.2007
Datum der Abgabe	14.08.2007

Technische Universität Dresden

FAKULTÄT ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Aufgabenbeschreibung für die Diplomarbeit von Herrn Christian Stüllein

Thema: Untersuchung und Implementierung numerischer Optimierungsmethoden zur Lösung inverser elektromagnetischer Streuprobleme

Aufgabenstellung:

Im Rahmen des Raumfahrtexperimentes CONSERT (Comet Nucleus Sounding Experiment by Radiowave Transmission) der Kometenmission Rosetta werden quasi tomographische Untersuchungen des Kometenkerns durchgeführt. Hierbei ist es das Ziel, durch die Lösung des inversen Streuproblems Rückschlüsse auf die dielektrischen Eigenschaften der inneren Kernstruktur ziehen zu können. Neben Methoden der direkten Inversion kann das inverse Streuproblem auch auf ein Optimierungsproblem zurückgeführt werden, zu dessen Lösung lediglich die Vorwärtsrechnung erforderlich ist. Im Rahmen dieser Diplomarbeit soll die Leistungsfähigkeit verschiedener Optimierungsalgorithmen zur Materialparameterrekonstruktion untersucht werden.
Die Aufgabenstellung umfasst folgende Punkte:

- Einarbeitung in das Finite-Difference-Time-Domain Verfahren zur Vorwärtsrechnung
- Erstellung eines einfachen dielektrischen Modells für einen Kometenkern
- Implementierung der FDTD-Methode in die Vorwärtsrechnung für das dielektrische Modell des Kometenkerns
- Implementierung numerischer Optimierungsalgorithmen zur Materialrekonstruktion
- Untersuchung der Rekonstruktionsperformance der Optimierungsverfahren
- Visualisierung der Ergebnisse, z.B. in Form von Schnittbildern
- Dokumentation der Programme und Niederschrift der Arbeit

Ausgehändigt am:	**14.02.2007**	**Betreuer:**	**Dipl.-Ing. Diemo Landmann**
Einzureichen bis:	**14.08.2007**		**Dr.-Ing. Dirk Plettemeier**

Selbständigkeitserklärung

Ich versichere, dass ich diese Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Heidelberg, 09.08.2007 Christian Stüllein

Abstract

Die Lösung des inversen Streuproblems zielt auf die Rekonstruktion der dielektrischen Parameter eines unbekannten Streuobjektes ab. Im Gegensatz zum *einfachen* Streuproblem, bei dem die elektromagnetische Feldverteilung in Abhängigkeit von bekannten Quellen und Streukörpern berechnet wird (Vorwärtsrechnung), wird hier ein unbekanntes Streuobjekt ausgehend von gemessenen Feldern näherungsweise bestimmt. Ein Ansatz hierfür besteht in der Anwendung nichtlinearer Optimierungsalgorithmen. In einem iterativen Verfahren wird eine bestimmte Verteilung der dielektrischen Eigenschaften innerhalb des Berechnungsraumes angenommen, die resultierenden Felder mittels Vorwärtsrechnung bestimmt und die Parameterverteilung so angepasst, damit die Differenz von berechnetem und gemessenem Feld minimiert wird. In dieser Arbeit werden für ausgewählte Optimierungsverfahren und für die benötigte Vorwärtsrechnung die theoretischen Grundlagen behandelt, eine mögliche Implementierung vorgestellt und in diversen Versuchen die Funktionstüchtigkeit der Vorwärtsrechnung und der Rekonstruktion belegt.

The inverse scattering approach is dealing with the reconstruction of dielectric parameters of an unknown scattering object. In contrast to the direct scattering problem, which calculates the resulting electromagnetic field distribution according to known sources and known scatterers (forward calculation), an unknown scattering object will be approximately determined based on a set of measured field values. One possible approach consits of applying non-linear deterministic optimization methods. In an iterative procedure a distribution of dielectric properties is assumed, the resulting field is determined by forward calculation and the distribution of parameters is updated in a manner, that the difference between calculated and measured field will decrease and converge to an minimum. In this thesis, the theoretical fundamentals for a selected subset of optimization algorithms and for the required forward solution are investigated. Furthermore a possible implementation is introduced and in a number of experiments the operational reliability of the forward calculation and the reconstruction algorithms are proven.

Inhaltsverzeichnis

Verzeichnis der verwendeten Abkürzungen und Symbole

Abkürzungen

ABC	Absorbing Boundary Conditions
FDTD	Finite Difference Time Domain
FEM	Finite Elements Method
GN	Gauss Newton
GPML	Generalized Perfectly Matched Layer
IDE	Integrated Development Environment
LM	Levenberg Marquardt
MoM	Method of Moments
NME	Normalized Misfit Error
PML	Perfectly Matched Layer
SD	Steppest Descent

Symbole

A	Amplitude einer Welle
$\vec{B}$	magnetische Flussdichte
c	Lichtgeschwindigkeit im Medium
c_0	Lichtgeschwindigkeit im Vakuum
$\mathbf{\Delta p}$	Änderung des Parametervektors
Δs	Kantenlänge der kubischen Yee-Zelle
Δt	Zeitschrittlänge
Δx	Kantenlänge der Yee-Zelle in x-Richtung
Δy	Kantenlänge der Yee-Zelle in y-Richtung
Δz	Kantenlänge der Yee-Zelle in z-Richtung
δ^i	für den Iterationsschritt i gewählte Schrittweite
δ^j	Folge von Schrittweiten
$\vec{D}$	elektrische Flussdichte
ϵ	Permittivität

ϵ_0	elektrische Feldkonstante (Permittivität des Vakuums)
ϵ_r	relative Permittivität bzw. Permittivitätszahl
$\vec{E}$	elektrischer Feldvektor
$\vec{E}^{ein}$	einfallendes elektrisches Feld $\left(\vec{E}^{i}\right)$
$\vec{E}^{streu}$	gestreutes elektrisches Feld $\left(\vec{E}^{s}\right)$
$\vec{E}^{total}$	gesamtes elektrisches Feld $\left(\vec{E}^{t}\right)$
$\vec{e}_i$	Einheitsvektor in Richtung i
E	transportierte Energie einer Welle
E^i	Feldwerte an den Messstellen zur i'ten Iteration
E^{meas}	Messwerte
$E_m^n(i,j,k)$	m-Komponente ($m = x, y, z$) des elektrischen Feldes an dem diskreten Ort (i, j, k) zum diskreten Zeitpunkt n
$E_m^{s,n}(i,j,k)$	Wert der m-Komponente ($m = x, y, z$) des gestreuten elektrischen Feldes an dem diskreten Ort (i, j, k) zum diskreten Zeitpunkt n
E_x	x-Komponente des elektrischen Feldes
E_y	y-Komponente des elektrischen Feldes
E_z	z-Komponente des elektrischen Feldes
$f(s,t)$	allgemeine Funktion des Ortes s und der Zeit t
$FDTD()$	Operator der Vorwärtsrechnung
$FL_{dp,CPU}$	Anzahl doppelt genauer Rechenoperationen, die eine CPU pro Sekunde ausführen kann
Γ	Wellenwiderstand
Γ_F	Wellenwiderstand des freien Raumes
$\mathbf{G}$	Gauß-Hesse-Matrix
$\vec{H}$	magnetischer Feldvektor
$\vec{H}^{ein}$	einfallendes magnetisches Feld $\left(\vec{H}^{i}\right)$
$\vec{H}^{streu}$	gestreutes magnetisches Feld $\left(\vec{H}^{s}\right)$
$\vec{H}^{total}$	gesamtes magnetisches Feld $\left(\vec{H}^{t}\right)$
$H_K(\mathbf{p^i})$	Hessematrix der Kostenfunktion K und ihrer Parameter $\mathbf{p}^i$

H_x	x-Komponente des magnetischen Feldes
H_y	y-Komponente des magnetischen Feldes
H_z	z-Komponente des magnetischen Feldes
i	diskrete Ortskoordinate in x-Richtung
$\mathbf{J}^i\left(\mathbf{r}^i, \mathbf{p}^\mathbf{i}\right)$	Jacobimatrix; Sensitivität des aktuellen Residuums auf Änderung des Parametervektors
$\vec{J}$	Stromdichtevektor
j	diskrete Ortskoordinate in y-Richtung
K	allgemeine Kostenfunktion, die es zu minimieren gilt
k	diskrete Ortskoordinate in z-Richtung
λ	Dämpfungsfaktor des Levenberg-Marquardt Verfahrens
λ_{min}	kleinste, aufzulösende Wellenlänge
μ	Permeabilität
μ_0	magnetische Feldkonstante (Permeabilität des Vakuums)
μ_r	relative Permeabilität bzw. Permeabilitätszahl
N	Anzahl der Messstellen
n	diskrete Zeit oder Brechzahl
N_M	Anzahl der verschiedenen Materialien
N_O	Anzahl der Beobachtungspunkte
N_R	Anzahl der Empfänger
N_S	Anzahl der Quellen
N_s	Ausdehnung des Problemraumes in alle Richtungen (kubischer Raum)
N_T	Anzahl der Zeitschritte
N_x	Ausdehnung des Problemraumes in x-Richtung
N_y	Ausdehnung des Problemraumes in y-Richtung
N_z	Ausdehnung des Problemraumes in z-Richtung
∇	Nabla-Operator
∇A	Gradient des Skalarfeldes A
$\nabla \vec{A}$	Divergenz des Vektorfeldes $\vec{A}$

$\nabla \times \vec{A}$	Rotation des Vektorfeldes $\vec{A}$
∇K	Gradient der Kostenfunktion K
OP_{CPU}	Anzahl der auszuführenden Rechenoperationen
$\mathbf{p}$	Parametervektor
$\mathbf{p}^i$	aktueller Parametervektor
Φ	magnetischer Fluss
Ψ	allgemeine Feldkomponente
Ψ_i^n	die i-Komponente ($i = x, y, z$) zum diskreten Zeitpunkt n der allgemeinen Feldkomponente
P	Anzahl der unbekannten, zu bestimmenden Parameter
$\mathbf{r}^i$	aktueller Residuenvektor (enthält Abweichung der berechneten von den wahren Feldwerten)
R	Reflexionsfaktor
r	Abstand von der Quelle
rot	Rotationsoperator
σ	elektrische Leitfähigkeit
σ^*	magnetische Leitfähigkeit
s	Ort
s_x	inverse Geschwindigkeit in x-Richtung
s_y	inverse Geschwindigkeit in y-Richtung
s_z	inverse Geschwindigkeit in z-Richtung
T	Transmissionsfaktor
t	Zeit
t_n	aktueller, diskreter Zeitpunkt
T_{CPU}	Zeit, welche die CPU benötigt um eine Vorwärtsrechnung durchzuführen
v_n	minimale numerische Geschwindigkeit der Informationsausbreitung
v_y	Geschwindigkeit in y-Richtung
v_z	Geschwindigkeit in z-Richtung
X	Kantenlänge eines aus Yee-Zellen aufgebauten Quaders in x-Richtung

x	Ortskoordinate in x-Richtung
x_0	untere Grenze von x
x_m	obere Grenze von x
Y	Kantenlänge eines aus Yee-Zellen aufgebauten Quaders in y-Richtung
y	Ortskoordinate in y-Richtung
y_m	obere Grenze von y
Z	Kantenlänge eines aus Yee-Zellen aufgebauten Quaders in z-Richtung
z	Ortskoordinate in z-Richtung
z_m	obere Grenze von z

1. Einleitung

Bei dem sogenannten Microwave Imaging handelt es sich um eine Methode, welche es ermöglicht die unbekannte Parameterverteilung eines Streukörpers zu rekonstruieren. Zu diesem Zweck werden elektromagnetische Wellen im Mikrowellenbereich[1] ausgesendet und die Feldvektoren im eingeschwungenen Zustand an diskreten Stellen um das Objekt herum aufgenommen. Durch die Verteilung der elektrischen und magnetischen Eigenschaften innerhalb des Objektes kommt es zu Reflexionen und Streuungen, welche die resultierenden Felder beeinflussen. Die aufgenommenen Feldkomponenten beinhalten damit Informationen über den Aufbau des Streukörpers. Ziel des Mikrowave Imaging ist die Rekonstruktion des Objektes auf Basis dieser Felder.
Neben der direkten Inversion ist der Ansatz das Streuproblem mittels mathematischer Optimierungsverfahren zu lösen sehr interessant. Hierbei wird, von einer geschätzten Parameterverteilung innerhalb des Streukörpers ausgehend, das von einer bekannten Quelle verursachte Feld mittels spezieller Methoden an den definierten Messorten berechnet und mit den gemessenen Werten verglichen. Die Optimierungsalgorithmen sorgen für eine schrittweise Anpassung der Parameterverteilung, um die Differenz von gemessenem und berechnetem Feld zu minimieren. Ist diese Abweichung klein genug, so kann davon ausgegangen werden, dass die zugehörige Parameterverteilung der des realen Objektes entspricht. Somit ist es möglich die elektrischen Eigenschaften eines Körpers allein aus der bekannten Quelle und der Feldvektoren an den Messstellen zu ermitteln.

Das beschriebene Verfahren erfordert es, ausgehend von einer Quelle die Felder an beliebigen Raumpunkten berechnen zu können. Dabei müssen die Einflüsse des Streukörpers berücksichtigt werden. In der vorliegenden Arbeit wurde die FDTD (Finite Differences/Time Domain) für diese Aufgabe eingesetzt. Die FDTD wurde in ihrer heutigen Form erstmals 1966 von Yee [Yee66] beschrieben. Da der benötigte Rechenaufwand sehr hoch ist, gewann sie erst mit der Verfügbarkeit schneller Rechner in den letzten 20 Jahren an Bedeutung. Doch auch heute übersteigen die Anforderungen mitunter die verfügbare Rechenleistung, weshalb der zu betrachtende Problemraum immer möglichst klein gewählt werden sollte. Da es an den Raumgrenzen zu diversen Problemen

[1] $300MHz$ bis $300GHz$ ($\lambda = 1mm..1m$)

kommt, müssen zusätzlich sogenannte absorbierende Randbedingungen zum Einsatz kommen. Ziel dieser Arbeit ist es, eine geeignete Implementierung der FDTD vorzunehmen und anschließend verschiedene deterministische Optimierungsalgorithmen so einzubauen, dass die Rekonstruktion vorgegebener Streuobjekte ermöglicht wird. Anhand dieser Werkzeuge sollen verschiedene Tests durchgeführt und die Performance der Optimierungsverfahren eingeschätzt werden. Die Arbeit gliedert sich dazu in drei Teile. Im ersten Abschnitt werden theoretische Betrachtungen bezüglich der FDTD und der Optimierungsverfahren dargestellt. Darauf folgt die Beschreibung der Implementierung dieser Tools. Abschließend werden Versuche durchgeführt, um die Funktionsweise der FDTD und der Optimierung darzustellen und um zu zeigen, dass es möglich ist, die Parameter eines Streuobjektes auf diese Weise zu rekonstruieren.

Motivation Eine der zukünftigen Hauptanwendungen dieser Rekonstruktionsverfahren wird die Erkennung von Tumoren sein. Viele wissenschaftliche Arbeiten beschäftigen sich mit der Anpassung und Optimierung des Verfahrens zur Detektion von Brustkrebs. Hierbei ist von bedeutendem Vorteil, dass keine Röntgenstrahlen eingesetzt werden müssen, sondern ungefährliche Mikrowellen zum Einsatz kommen. Anlass für die vorliegende Arbeit war allerdings eine gänzlich andere Anwendung. Im Jahr 2014 wird die Raumsonde Rosetta eine Umlaufbahn um den Kometen *Tschurjumow-Gerasimenko* einnehmen und den Lander *Philae* zur Landung auf den Kometen vorbereiten. Mithilfe dieser Konstellation werden elektromagnetische Wellen ausgesendet und von der Raumsonde an verschiedenen Punkten der Umlaufbahn aufgenommen. Anhand der gesammelten Daten sollen Rückschlüsse auf die materielle Zusammensetzung des Komentenkerns gezogen werden. Im Rahmen dieses Projektes laufen verschiedene wissenschaftliche Untersuchungen an der Technischen Universität Dresden. Parallel beschäftigen sich weitere Arbeiten mit der direkten Inversion[2] und der globalen Optimierung mittels stochastischer Verfahren.

[2]Doktorarbeit von Dipl.-Ing. Diemo Landmann

2. Theoretische Grundlagen

2.1. Die FDTD Methode

2.1.1. Einleitung

Bei der FDTD-Methode (Finite Difference Time Domain) handelt es sich um ein seit langem bekanntes Verfahren zur numerischen Berechnung von Differentialgleichungen. Zur Lösung der Maxwellschen Gleichungen [Gon04] wurde sie in ihrer jetzigen Form erstmals 1966 von Yee [Yee66] eingesetzt und ermöglicht die Berechnung der Ausbreitung elektromagnetischer Felder. Ein Vorteil der Feldberechnung mittels FDTD gegenüber anderen Verfahren wie zum Beispiel FEM (Finite Elements Method) oder MOM (Method of Moments) ist die Tatsache, dass die Feldkomponenten für jeden Zeitschritt ermittelt werden und somit auch die Propagierung von Pulsen und anderen nichtharmonischen Quellen simuliert werden kann. Ausgehend von den Maxwellschen Gleichungen, welche die Relationen zwischen magnetischen und elektrischen Feld beschreiben, führt die Diskretisierung der Variablen und ihrer örtlichen und zeitlichen partiellen Ableitungen zu einem Satz von Differenzengleichungen. Diese ermöglichen es ausgehend von Werten der elektromagnetischen Feldkomponenten zu einem Zeitpunkt (t_n) die Werte zum Folgezeitpunkt ($t_{n+1} = t_n + \Delta t$) zu berechnen. Die Differenzengleichungen werden deshalb auch Updategleichungen oder Aktualisierungsgleichungen genannt. Damit ist es möglich den räumlichen Verlauf des elektromagnetischen Feldes zu jedem diskreten Zeitpunkt zu bestimmen.

2.1.2. Formulierung der Methode

Die Maxwellschen Gleichungen Den Ausgangspunkt zur Herleitung der Aktualisierungsgleichungen des elektrischen und des magnetischen Feldes stellen die erste und dritte Maxwellsche Gleichung dar.

$$\oint \vec{H} \cdot d\vec{s} = \int_A \vec{J} \cdot d\vec{A} + \int_A \frac{\partial \vec{D}}{\partial t} \cdot d\vec{A} \tag{2.1}$$

$$\oint \vec{E} \cdot d\vec{s} = -\frac{\partial \Phi}{\partial t} \tag{2.2}$$

Unter Anwendung des Stokeschen Integralsatzes [Gon04]

$$\int_A rot\,\vec{a} \cdot d\vec{A} = \oint \vec{a} \cdot d\vec{s} \tag{2.3}$$

auf die linken Seiten von Gln. (2.1) und Gln. (2.2) und unter Beachtung von

$$\Phi = \int_A \vec{B}\, d\vec{A} \tag{2.4}$$

ergeben sich direkt:

$$\int_A rot\,\vec{H} \cdot d\vec{A} = \int_A \vec{J} \cdot d\vec{A} + \int_A \frac{\partial \vec{D}}{\partial t} \cdot d\vec{A} \tag{2.5}$$

$$\int_A rot\,\vec{E} \cdot d\vec{A} = -\int_A \frac{\partial \vec{B}}{\partial t} \cdot d\vec{A} \tag{2.6}$$

Durch Differenzierung nach $d\vec{A}$ entstehen die beiden Maxwellschen Gleichungen in ihrer differenziellen Form.

$$rot\,\vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \tag{2.7}$$

$$rot\,\vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{2.8}$$

Eine ausführliche Darstellung der beiden partiellen Differentialgleichungen erhält man, wenn die Rotation mittels des Nabla-Operators ausgedrückt ($rot\vec{A} = \nabla \times \vec{A}$) und für die folgende Definition des Nabla-Operators das Vektorprodukt bestimmt wird.

$$\nabla = \vec{e}_x \frac{\partial}{\partial x} + \vec{e}_y \frac{\partial}{\partial y} + \vec{e}_z \frac{\partial}{\partial z} \tag{2.9}$$

Für die Rotation gilt somit:

$$rot\,\vec{A} = \nabla \times \vec{A} = \left(\frac{\partial a_z}{\partial y} - \frac{\partial a_y}{\partial z}\right)\vec{e}_x + \left(\frac{\partial a_x}{\partial z} - \frac{\partial a_z}{\partial x}\right)\vec{e}_y + \left(\frac{\partial a_y}{\partial x} - \frac{\partial a_x}{\partial y}\right)\vec{e}_z \tag{2.10}$$

Angewendet auf Gln. (2.7) und (2.8) entsteht nach Vertauschen der Seiten und unter Einsetzen

von

$$\begin{aligned} \vec{D} &= \epsilon \vec{E} \\ \vec{J} &= \sigma \vec{E} \\ \vec{B} &= \mu \vec{H} \end{aligned} \tag{2.11}$$

folgendes System von Gleichungen in Komponentenschreibweise:

$$\begin{aligned} \sigma E_x + \epsilon \frac{\partial E_x}{\partial t} &= \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \sigma E_y + \epsilon \frac{\partial E_y}{\partial t} &= \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \sigma E_z + \epsilon \frac{\partial E_z}{\partial t} &= \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{aligned} \tag{2.12}$$

$$\begin{aligned} \mu \frac{\partial H_x}{\partial t} &= \frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} \\ \mu \frac{\partial H_y}{\partial t} &= \frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} \\ \mu \frac{\partial H_z}{\partial t} &= \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \end{aligned} \tag{2.13}$$

Diskretisierung Bei der FDTD-Methode folgt nun die direkte Diskretisierung der Gleichungen. Die Raum- und Zeitkoordinaten werden hierbei wie folgt ersetzt:

$$\begin{aligned} t &= n \cdot \Delta t \\ x &= i \cdot \Delta x \\ y &= j \cdot \Delta y \\ z &= k \cdot \Delta z \end{aligned} \tag{2.14}$$

Der gesamte betrachtete Ortsraum wird nun in uniforme Zellen aufgeteilt, für welche die einzelnen Feldkomponenten berechnet werden. Die Lokalisierung dieser Komponenten innerhalb einer Zelle wurde von Yee [Yee66] festgelegt und ist in Abbildung (1) dargestellt. Die Zellen werden auch Yee-Zellen genannt.

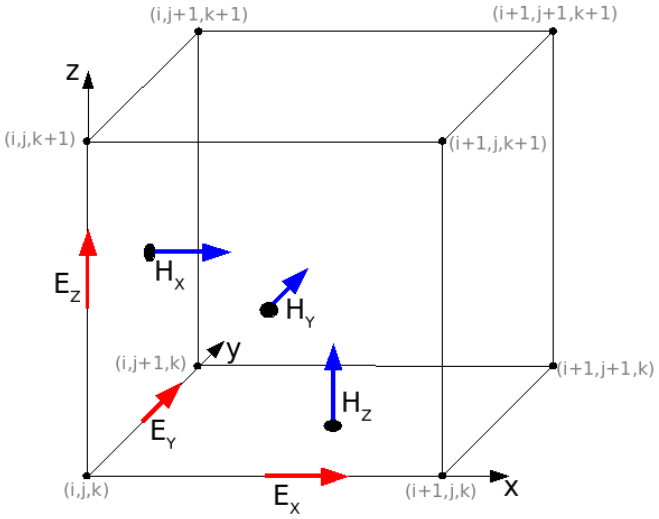

Abbildung 1: **Anordnung der Feldkomponenten innerhalb der Yee-Zelle:** 1966 legte Yee [Yee66] den Grundstein zur FDTD Methode, indem er die einzelnen Feldkomponenten nicht mehr wie bis dahin üblich im Zentrum der FDTD Zellen anordnete, sondern räumlich versetzt. Die Komponenten des Magnetfeldes werden in der Mitte der Flächen und die des elektrischen Feldes in der Mitte der Kanten einer sog. Yee-Zelle platziert. Dadurch ist es möglich durch Anwendung der zentralen Differenzen die örtlichen Ableitungen der Felder zu bestimmen

Hierbei ist ersichtlich, dass die E_x Komponente die relative Position $\left(i+\frac{1}{2},\, j,\, k\right)$ besitzt. Sie liegt damit an der absoluten örtlichen Position:

$$\left(\begin{array}{lcl} x & = & i\cdot\Delta x+\frac{1}{2} \\ y & = & j\cdot\Delta y \\ z & = & k\cdot\Delta z \end{array}\right) \tag{2.15}$$

Für die anderen Komponenten gelten entsprechend andere Offsets. In Tabelle (1) und Gln. (2.16)

Tabelle 1: **Position der Feldkomponenten**

Feldkomponenten	Offset innerhalb der Zelle	absolute Position
E_x	$\left(i+\frac{1}{2},\, j,\, k\right)$	$\left(i\cdot\Delta x+\frac{1}{2},\, j\cdot\Delta y,\, k\cdot\Delta z\right)$
E_y	$\left(i,\, j+\frac{1}{2},\, k\right)$	$\left(i\cdot\Delta x,\, j\cdot,\Delta y+\frac{1}{2},\, k\cdot\Delta z\right)$
E_z	$\left(i,\, j, k+\frac{1}{2}\right)$	$\left(i\cdot\Delta x, j\cdot\Delta y, k\cdot\Delta z+\frac{1}{2}\right)$
H_x	$\left(i,\, j+\frac{1}{2},\, k+\frac{1}{2}\right)$	$\left(i\cdot\Delta x,\, j\cdot\Delta y+\frac{1}{2},\, k\cdot\Delta z+\frac{1}{2}\right)$
H_y	$\left(i+\frac{1}{2},\, j,\, k+\frac{1}{2}\right)$	$\left(i\cdot\Delta x+\frac{1}{2},\, j\cdot\Delta y,\, k\cdot\Delta z+\frac{1}{2}\right)$
H_z	$\left(i+\frac{1}{2},\, j+\frac{1}{2},\, k\right)$	$\left(i\cdot\Delta x+\frac{1}{2},\, j\cdot\Delta y+\frac{1}{2},\, k\cdot\Delta z\right)$

sind die Positionen der Feldkomponenten einer Zelle (i,j,k) noch einmal dargelegt.

$$
\begin{aligned}
E_x(i,j,k) &\rightarrow \begin{pmatrix} i\cdot\Delta x+\frac{1}{2} \\ j\cdot\Delta y \\ k\cdot\Delta z \end{pmatrix} \quad H_x(i,j,k) \rightarrow H_x \begin{pmatrix} i\cdot\Delta x \\ j\cdot\Delta y+\frac{1}{2} \\ k\cdot\Delta z+\frac{1}{2} \end{pmatrix} \\
E_y(i,j,k) &\rightarrow \begin{pmatrix} i\cdot\Delta x \\ j\cdot\Delta y+\frac{1}{2} \\ k\cdot\Delta z \end{pmatrix} \quad H_y(i,j,k) \rightarrow H_y \begin{pmatrix} i\cdot\Delta x+\frac{1}{2} \\ j\cdot\Delta y \\ k\cdot\Delta z+\frac{1}{2} \end{pmatrix} \\
E_z(i,j,k) &\rightarrow \begin{pmatrix} i\cdot\Delta x \\ j\cdot\Delta y \\ k\cdot\Delta z+\frac{1}{2} \end{pmatrix} \quad H_z(i,j,k) \rightarrow H_z \begin{pmatrix} i\cdot\Delta x+\frac{1}{2} \\ j\cdot\Delta y+\frac{1}{2} \\ k\cdot\Delta z \end{pmatrix}
\end{aligned}
\tag{2.16}
$$

Um von den Differentialgleichungen aus Gln. (2.12) und (2.13) die Differenzengleichungen zu erhalten, werden die Ableitungen diskretisiert. Dazu wird das zentrale Differenzierungsschema verwendet. Hier gelten:

$$
\begin{aligned}
\frac{\partial f(s,t)}{\partial t} &\approx \frac{f(s,t_2)-f(s,t_1)}{\Delta t} && \text{für die Zeitableitung} \\
\frac{\partial f(s,t)}{\partial s} &\approx \frac{f(s_2,t)-f(s_1,t)}{\Delta s} && \text{für die Ortsableitung in Richtung s}
\end{aligned}
\tag{2.17}
$$

Die Aktualisierung der Felder geht in zwei Halbschritten vonstatten. Im ersten Halbschritt wird das E-Feld für den Zeitpunkt $t = n + \frac{1}{2}$ aktualisiert und diese Werte dann für die Berechnung des H-Feldes zum Zeitpunkt $t = n + 1$ verwendet. Da die Gleichungen in (2.12) auch die E-Komponenten ohne Ableitungen enthalten, könnten auch diese für den Zeitpunkt $t = n + \frac{1}{2}$ folgendermaßen approximiert werden [Tup97]:

$$
\sigma E\left(n+\frac{1}{2}\right) \approx \sigma\frac{E(n+1)-E(n)}{2} \tag{2.18}
$$

Da dies aber in [KL93] keine Beachtung fand, wurde auf diese Ersetzung auch in der vorliegenden

Arbeit verzichtet, um die Vergleichbarkeit mit dem Standardcode von Kunz und Luebbers zu gewährleisten. Für den Wert $E_i^{n+\frac{1}{2}}$ wird hier der entsprechende Wert zum Zeitpunkt $t = n+1$ verwendet.

Wird Gln. (2.12) mittels der Näherungen aus Gln. (2.17) zum Zeitpunkt $t = n+\frac{1}{2}$ diskretisiert, so ergeben sich:

$$\begin{aligned}
\sigma E_x^{n+1}\left(i+\frac{1}{2},j,k\right) &+ \epsilon\frac{E_x^{n+1}\left(i+\frac{1}{2},j,k\right)-E_x^{n}\left(i+\frac{1}{2},j,k\right)}{\Delta t} \\
&= \frac{H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)-H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j-\frac{1}{2},k\right)}{\Delta y} \\
&\quad -\frac{H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k-\frac{1}{2}\right)}{\Delta z} \\
\sigma E_y^{n+1}\left(i,j+\frac{1}{2},k\right) &+ \epsilon\frac{E_y^{n+1}\left(i,j+\frac{1}{2},k\right)-E_y^{n}\left(i,j+\frac{1}{2},k\right)}{\Delta t} \\
&= \frac{H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)-H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k-\frac{1}{2}\right)}{\Delta z} \\
&\quad -\frac{H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)-H_z^{n+\frac{1}{2}}\left(i-\frac{1}{2},j+\frac{1}{2},k\right)}{\Delta x} \\
\sigma E_z^{n+1}\left(i,j,k+\frac{1}{2}\right) &+ \epsilon\frac{E_z^{n+1}\left(i,j,k+\frac{1}{2}\right)-E_z^{n}\left(i,j,k+\frac{1}{2}\right)}{\Delta t} \\
&= \frac{H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-H_y^{n+\frac{1}{2}}\left(i-\frac{1}{2},j,k+\frac{1}{2}\right)}{\Delta x} \\
&\quad -\frac{H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)-H_x^{n+\frac{1}{2}}\left(i,j-\frac{1}{2},k+\frac{1}{2}\right)}{\Delta y}
\end{aligned} \tag{2.19}$$

Zur Vereinfachung wird die Schreibweise folgendermaßen abgewandelt. Die halben Zeitschritte treten im Programm nicht in Erscheinung, weshalb statt $n+\frac{1}{2}$ nur n geschrieben wird. Auch werden die Offsets der Feldkomponenten innerhalb der Yee-Zelle weggelassen und lediglich die zugehörige Zelle mit ihren Koordinaten (i,j,k) adressiert. Mit diesen Vereinbarungen und nach Auflösen der Gleichungen aus (2.19) nach den E-Feld Komponenten zum Zeitpunkt $t = n+1$

folgt:

$$\begin{aligned}
E_x^{n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_x^n(i,j,k) &+ \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_z^n(i,j,k) - H_z^n(i,j-1,k)\right)\\
&- \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_y^n(i,j,k) - H_y^n(i,j,k-1)\right)\\
E_y^{n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_y^n(i,j,k) &+ \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_x^n(i,j,k) - H_x^n(i,j,k-1)\right)\\
&- \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_z^n(i,j,k) - H_z^n(i-1,j,k)\right)\\
E_z^{n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_z^n(i,j,k) &+ \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_y^n(i,j,k) - H_y^n(i-1,j,k)\right)\\
&- \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_x^n(i,j,k) - H_x^n(i,j-1,k)\right)
\end{aligned} \tag{2.20}$$

Wenn die Diskretisierung analog für Gln. (2.13) zum Zeitpunkt $n+1$ vollzogen wird, gelten zunächst:

$$\begin{aligned}
\mu\frac{H_x^{n+\frac{3}{2}}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right) - H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)}{\Delta t} &= \frac{E_y^{n+1}\left(i,j+\frac{1}{2},k+1\right) - E_y^{n+1}\left(i,j+\frac{1}{2},k\right)}{\Delta z}\\
&- \frac{E_z^{n+1}\left(i,j+1,k+\frac{1}{2}\right) - E_z^{n+1}\left(i,j,k+\frac{1}{2}\right)}{\Delta y}\\
\mu\frac{H_y^{n+\frac{3}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right) - H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)}{\Delta t} &= \frac{E_z^{n+1}\left(i+1,j,k+\frac{1}{2}\right) - E_z^{n+1}\left(i,j,k+\frac{1}{2}\right)}{\Delta x}\\
&- \frac{E_x^{n+1}\left(i+\frac{1}{2},j,k+1\right) - E_x^{n+1}\left(i+\frac{1}{2},j,k\right)}{\Delta z}\\
\mu\frac{H_z^{n+\frac{3}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right) - H_z^{n+\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)}{\Delta t} &= \frac{E_x^{n+1}\left(i+\frac{1}{2},j+1,k\right) - E_x^{n+1}\left(i+\frac{1}{2},j,k\right)}{\Delta y}\\
&- \frac{E_y^{n+1}\left(i+1,j+\frac{1}{2},k\right) - E_y^{n+1}\left(i,j+\frac{1}{2},k\right)}{\Delta x}
\end{aligned} \tag{2.21}$$

Wie oben beschrieben werden nur ganze Zeitschritte n zugelassen und die Komponentenoffsets innerhalb der Yee-Zelle vernachlässigt. Die Auflösung nach H_i^n ergibt dann die Aktualisierungs-

gleichungen für die Magnetfeldkomponenten.

$$\begin{aligned}
H_x^{n+1}(i,j,k) = H_x^n(i,j,k) &+ \frac{\Delta t}{\mu}\frac{1}{\Delta z}\left(E_y^{n+1}(i,j,k+1) - E_y^{n+1}(i,j,k)\right) \\
&- \frac{\Delta t}{\mu}\frac{1}{\Delta y}\left(E_z^{n+1}(i,j+1,k) - E_z^{n+1}(i,j,k)\right) \\
H_y^{n+1}(i,j,k) = H_y^n(i,j,k) &+ \frac{\Delta t}{\mu}\frac{1}{\Delta x}\left(E_z^{n+1}(i+1,j,k) - E_z^{n+1}(i,j,k)\right) \\
&- \frac{\Delta t}{\mu}\frac{1}{\Delta z}\left(E_x^{n+1}(i,j,k+1) - E_x^{n+1}(i,j,k)\right) \\
H_z^{n+1}(i,j,k) = H_z^n(i,j,k) &+ \frac{\Delta t}{\mu}\frac{1}{\Delta y}\left(E_x^{n+1}(i,j+1,k) - E_x^{n+1}(i,j,k)\right) \\
&- \frac{\Delta t}{\mu}\frac{1}{\Delta x}\left(E_y^{n+1}(i+1,j,k) - E_y^{n+1}(i,j,k)\right)
\end{aligned} \tag{2.22}$$

Wie aus den beiden Sätzen an Aktualisierungsgleichungen (2.20) und (2.22) ersichtlich ist, werden für die Berechnung der zeitlichen Nachfolgewerte für die E-Feld Komponenten die aktuellen H-Feld Komponenten benötigt $\left(E^{n+1} = f\left(H^n\right)\right)$. Für die Aktualisierung des H-Feldes hingegen werden schon die neuen Werte des E-Feldes verwendet $\left(H^{n+1} = f\left(E^{n+1}\right)\right)$. Eigentlich handelt es sich aber nicht um die Komponenten zum selben Zeitpunkt, sondern um einen Halbschritt verschoben. Da Programmiersprachen aber keine halben Indizes für Felder unterstützen, wird der genannten Tatsache dadurch Rechnung getragen, dass erst das E-Feld und dann das H-Feld aktualisiert wird.

Mit Gln. (2.20) und (2.22) liegen nun die benötigten Gleichungssysteme vor, um einen Rechner, ausgehend von einer gegebenen örtlichen Feldverteilung, die Verteilung zum nächsten Zeitschritt bestimmen lassen zu können.

Implementierung von Quellen Quellen werden dadurch realisiert, dass in bestimmten Zellen, welche den Ort der Quelle darstellen, entsprechende Komponenten fest vorgegeben werden. So können beliebige Quellen[3] im Berechnungsraum platziert werden. Die Propagierung der sich ausbreitenden Welle erfolgt automatisch über die Aktualisierungsgleichungen.

[3] Wie beispielsweise Gauß-Puls oder harmonische Quellen

Stabilität der Methode Damit die Methode brauchbare Ergebnisse liefert, sind einige Kriterien bezüglich der Raum- und Zeitdiskretisierung zu beachten. Als erstes müssen die Zellen klein genug gewählt werden, um örtliche Feldschwankungen ausreichend genau darstellen zu können. Hierfür ist die kleinste aufzulösende Wellenlänge maßgebend. Als Faustformel gilt allgemein [Tup97]:

$$\Delta s \approx \frac{\lambda_{min}}{10} \dots \frac{\lambda_{min}}{20} \tag{2.23}$$

Hier steht Δs stellvertretend für die Kantenlängen der Zellen in jede der drei Raumrichtungen. Die andere kritische Diskretisierungsgröße ist Δt. Sie muss so gewählt werden, dass sich die Information in der numerischen Lösung mindestens genauso schnell ausbreiten kann, wie das wirkliche Signal, dessen Ausbreitungsgeschwindigkeit die Lichtgeschwindigkeit ist. Für den allgemeinen Fall einer nichtkubischen Zelle wird die numerische Ausbreitungsgeschwindigkeit der Information bestimmt, indem ein Quader mit den Kantenlängen X, Y und Z betrachtet wird. Die Länge seiner Diagonale beträgt $\sqrt{X^2 + Y^2 + Z^2}$. Die Zeit, um diese Strecke numerisch zurückzulegen, ergibt sich aus der Anzahl der Ortsschritte in jede Richtung multipliziert mit der Zeitschrittlänge. Für die numerische Ausbreitungsgeschwindigkeit gilt somit:

$$v_n = \frac{\sqrt{X^2 + Y^2 + Z^2}}{\left(\frac{X}{\Delta x} + \frac{Y}{\Delta y} + \frac{Z}{\Delta z}\right) \Delta t} \tag{2.24}$$

Die numerische Geschwindigkeit v_n wird minimal, wenn sich die Kantenlängen des Quaders indirekt proportional zu den Zell-Kantenlängen verhalten.

$$\begin{aligned} X &= \frac{1}{\Delta x} \\ Y &= \frac{1}{\Delta y} \\ Z &= \frac{1}{\Delta z} \end{aligned} \tag{2.25}$$

Wenn zum Beispiel Δx sehr klein ist, dafür die Kantenlänge X des Quaders sehr groß, dann müssen viele Schritte in X-Richtung gemacht werden und die Geschwindigkeit sinkt. Mit diesen

Überlegungen ergibt sich die Stabilitätsbedingung zu:

$$\Delta t \leq \frac{1}{\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}\, c} \tag{2.26}$$

Im Fall kubischer Zellen ergibt sich mit $\Delta x = \Delta y = \Delta z = \Delta s$:

$$\Delta t \leq \frac{\Delta s}{\sqrt{3}\, c} \tag{2.27}$$

Für beide Parameter (Δs und Δt) gilt, je kleiner sie gewählt werden, desto genauer ist die Rechnung. Allerdings hat ihre Wahl direkten Einfluss auf den Rechenaufwand. Für jede Aufgabe sollte also zwischen geforderter Auflösung und benötigter Rechenzeit abgewogen werden.

2.1.3. Leap-Frog Schema

Das Zeitschrittverfahren der FDTD Methode wird üblicherweise als Leap-Frog Schema bezeichnet. Um verständlich zu machen, was damit gemeint ist, betrachten wir den einfachen Fall der eindimensionalen Feldausbreitung. Das E-Feld soll sich in z-Richtung ausbreiten und in der x-z Ebene oszillieren. Da die Ausbreitungsrichtung einer elektromagnetischen Welle immer orthogonal auf den Schwingungsrichtungen von E-Feld und H-Feld steht, ergibt sich für unseren Fall die Schwingung des E-Feldes in x-Richtung und die des H-Feldes in y-Richtung. Die oben hergeleiteten Aktualisierungsgleichungen (2.20) und (2.22) reduzieren sich somit zu:

$$\begin{aligned} E_x^{n+1}(i,j,k) &= \frac{\epsilon}{\epsilon + \sigma\,\Delta t} E_x^n(i,j,k) - \frac{\Delta t}{\epsilon + \sigma\,\Delta t\,\Delta z}\left(H_y^n(i,j,k) - H_y^n(i,j,k-1)\right) \\ H_y^{n+1}(i,j,k) &= H_y^n(i,j,k) - \frac{\Delta t}{\mu\,\Delta z}\left(E_x^{n+1}(i,j,k+1) - E_x^{n+1}(i,j,k)\right) \end{aligned} \tag{2.28}$$

Die gegenseitigen Abhängigkeiten bei der Berechnung der Felder zum nächsten Zeitschritt können graphisch dargestellt werden. Auf Abbildung (2) ist gut zu erkennen, dass zunächst im ersten Halbschritt das E-Feld aktualisiert wird. Hierbei zeigt sich auch, wie der Ausdruck *Leap-Frog* zustande kommt. Um von der E-Feld Komponente 1 an der Stelle $k + \frac{1}{2}$ zum Wert 2 im nächsten Zeitschritt zu kommen, müssen die örtlich benachbarten und zeitlich dazwischen liegenden

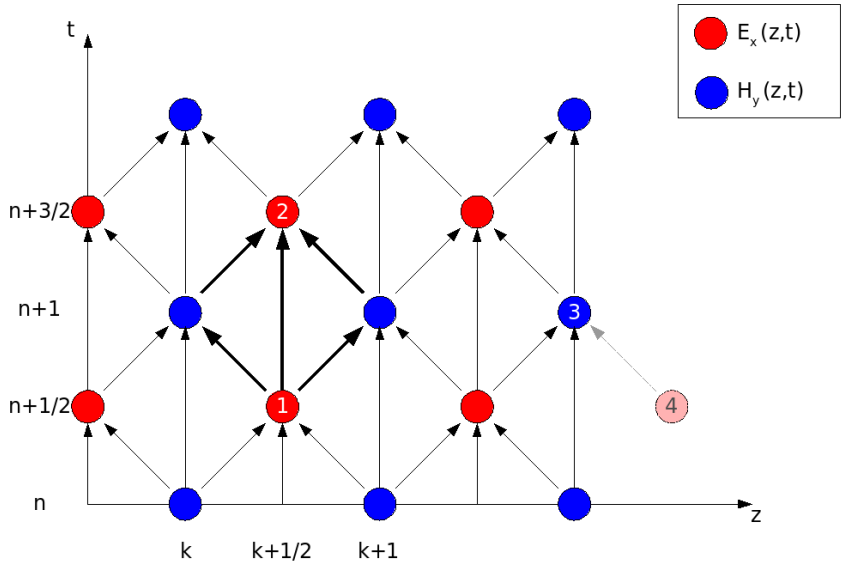

Abbildung 2: **Leap-Frog Schema:** Dieses Bild zeigt den Ablauf der Berechnung der einzelnen Feldkomponenten. Die roten Kreise stellen die Orte und Zeitpunkte dar, an denen die elektrischen Feldkomponenten bestimmt werden. Die blauen Kreise stehen für die Magnetfeldkomponenten. Am Beispiel der fett markierten Pfeile wird gezeigt, wie sich ausgehend von der E-Feldkomponente 1 in zwei Halbschritten über die Berechnung der benachbarten Magnetfeldkomponenten die E-Feldkomponente 2 am gleichen Ort zum nächsten Zeitschritt ergibt. Die Anordnung der Pfeile erinnert an die Bewegung eines springenden Laubfrosches, der zum Vorwärtsspringen seine Beine spreizt.

H-Feld Komponenten bekannt sein. Im Bild ähnelt das einem Frosch, der seine Beine spreizt, um vorwärts zu springen.

Aber auch ein Problem ist hier zu erkennen. Da zur Berechnung des E-Feldes immer auch beide benachbarten H-Feld Werte benötigt werden, und zu deren Berechnung ebenfalls die beiden E-Feld Nachbarn bekannt sein müssen, sind die normalen Gleichungen für die Randzellen nicht anwendbar. Am Beispiel der H_y Komponente 3 in Bild (2) wird ersichtlich, dass die erforderliche E-Feld Komponente 4 nicht verfügbar ist. Ohne Beachtung dieses Umstandes verhalten sich die Grenzen wie perfekt leitende Flächen, was zur totalen Reflexion der Felder führt. Sollen mittels FDTD abgeschlossene Räume mit entsprechenden Wänden simuliert werden, so ist das damit ohne Weiteres möglich. Sind hingegen die Felder in Teilräumen ohne solche Randeigenschaften von Interesse, so kann einerseits der Berechnungsraum vergrößert werden, um die ungewollten Reflexionen weit entfernt von Ort des Interesses zu haben[4] oder es kommen alternativ spezielle absorbierende Randbedingungen zum Einsatz, wie sie im Folgenden vorgestellt werden.

2.1.4. Absorbing Boundary Conditions

Bei einer absorbierenden Randbedingung (ABC - Absorbing Boundary Condition) handelt es sich im Idealfall um eine Begrenzung des zu berechnenden Raumes, die alle einfallenden Wellen vollständig absorbiert, um einen unendlich ausgedehnten Raum zu simulieren. Die real verfügbaren Randbedingungen führen allerdings zu mehr oder weniger ausgeprägten Reflexionen, die als Fehler in das Ergebnis eingehen. Die wichtigsten Randbedingungen sind die Mur'sche ABC erster und zweiter Ordnung sowie die (G)PML-Methode ((Generalized) Perfectly Matched Layer). Letztgenannte ist eine relativ neue Methode, die sehr gute Absorptionseigenschaften aufweist. Im Rahmen dieser Arbeit wurden aber nur die beiden Methoden von Mur betrachtet, da sich diese relativ einfach implementieren lassen und für die gegebene Aufgabenstellung vorerst völlig ausreichend sind. Für künftige weiterführende Arbeiten ist die Einführung der (G)PML aber sehr zu empfehlen, da sich durch ihre überragende Absorptionsperformance die Größe des zu berechnenden Raumes stark reduzieren lässt. Dieser Umstand hebt auch den Nachteil des erhöhten Rechenaufwands für die Randzellen mehr als auf.

[4] Auf Kosten von Rechenzeit und Arbeitsspeicher

Mur'sche Randbedingungen Bei den Mur'schen Randbedingungen wird davon ausgegangen, dass sich die Felder an den Raumgrenzen nach außen ausbreiten [Tup97]. Die Tangentialkomponenten sollen also idealerweise verschwinden. Mur geht von ebenen Wellen aus, für die gilt:

$$s_x^2 + s_y^2 + s_z^2 = \frac{1}{c_0^2} \tag{2.29}$$

Hierbei stellt s_i die sogenannte inverse Geschwindigkeit in i-Richtung dar.
Folgende Überlegungen müssen für jede Wand separat angestellt werden: Je nach Position der Wand muss die entsprechende inverse Geschwindigkeit vorzeichenrichtig eingesetzt werden. Hier soll die Betrachtung beispielhaft für die „linke" Wand[5] durchgeführt werden. Das Ebenwellenverhältnis zwischen der Zeit- und Ortsableitung lautet hier:

$$\left(\frac{\partial \psi}{\partial x} - s_x \frac{\partial \psi}{\partial t}\right)_{x=x_0} = 0 \tag{2.30}$$

Aus Gln. (2.29) kann s_x wie folgt bestimmt werden:

$$s_x = \frac{1}{c_0}\sqrt{1 - (c_0\, s_y)^2 - (c_0\, s_z)^2} \tag{2.31}$$

Mur'sche Randbedingung erster Ordnung In erster Näherung wird bei der Mur'schen ABC erster Ordnung der Wurzelausdruck aus Gln. (2.31) komplett vernachlässigt.

$$s_x \approx \frac{1}{c_0} \tag{2.32}$$

Es wird also davon ausgegangen, dass die Wellenausbreitung orthogonal zur Wand geschieht und damit der Vektor der Lichtgeschwindigkeit in die negative x-Richtung zeigt. Wird Gln. (2.32) in Gln. (2.30) eingesetzt, so gilt:

[5] $x = x_0$, das Raumvolumen liegt im Gebiet $x \geq x_0$

$$\left(\frac{\partial\psi}{\partial x} - \frac{1}{c_0}\frac{\partial\psi}{\partial t}\right)_{x=x_0} = 0 \tag{2.33}$$

Die Gleichung (2.33) muss nun diskretisiert werden. Damit die räumliche und zeitliche Ableitung zur gleichen Zeit am gleichen Ort vorliegen, wird dazu noch eine Interpolation der räumlichen Ableitung zum Zeitpunkt $n+\frac{1}{2}$ und der zeitlichen Ableitung am Ort $i+\frac{1}{2}$ durchgeführt.

$$\begin{aligned}
\frac{1}{2}\left(\frac{\psi_{i+1}^{n}-\psi_{i}^{n}}{\Delta x}+\frac{\psi_{i+1}^{n+1}-\psi_{i}^{n+1}}{\Delta x}\right)-\frac{1}{2}\left(\frac{\psi_{i}^{n+1}-\psi_{i}^{n}}{c_0\,\Delta t}+\frac{\psi_{i+1}^{n+1}-\psi_{i+1}^{n}}{c_0\,\Delta t}\right) &= 0 \\
\frac{\psi_{i+1}^{n}-\psi_{i}^{n}+\psi_{i+1}^{n+1}-\psi_{i}^{n+1}}{\Delta x}-\frac{\psi_{i}^{n+1}-\psi_{i}^{n}+\psi_{i+1}^{n+1}-\psi_{i+1}^{n}}{c_0\,\Delta t} &= 0
\end{aligned} \tag{2.34}$$

Umgestellt nach ψ_i^{n+1} ergeben sich die Aktualisierungsgleichung für die Randelemente ($i = x_0$).

$$\psi_i^{n+1} = \psi_{i+1}^{n} + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(\psi_{i+1}^{n+1} - \psi_i^{n}\right) \tag{2.35}$$

Wie die Annahme unter Gln . (2.32) nahelegt, arbeitet die Mur'sche Randbedingung erster Ordnung perfekt für senkrecht einfallende Wellen. Da diese Annahme aber spätestens in den Ecken nicht mehr für alle Wände zutreffen kann, ist der Fehler dort besonders groß.

Aus Gln. (2.35) geht auch hervor, dass die Werte der jeweiligen Randzelle und der Nachbarzelle zum Vorgängerzeitpunkt bekannt sein müssen. Da nur ein Zeitschritt in die Vergangenheit geschaut wird, werden diese Randbedingungen Mur'sche ABC *erster* Ordnung genannt.

Die Randbedingungen werden üblicherweise nur auf das elektrische Feld angewendet. Nachdem also im ersten Zeit-Halbschritt die E-Komponenten berechnet wurden, werden die Randelemente mittels der ABC-Aktualisierungsgleichungen korrigiert. Die im Folgeschritt berechneten Komponenten des Magnetfeldes beziehen sich dann bereits auf das am Rand absorbierte E-Feld.

Im Folgenden werden die Aktualisierungsgleichungen für die „linke“ und „rechte“ Wand dar-

gestellt. Zu beachten ist, dass durch die Annahme ebener Wellenausbreitung in die jeweilige Richtung i die Komponente E_i nicht existiert (EM-Felder schwingen immer orthogonal zur Ausbreitungsrichtung). Hier werden dementsprechend nur E_y und E_z bearbeitet.

$$\begin{aligned} E_{y/z}^{n+1}(1,j,k) &= E_{y/z}^{n}(2,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_{y/z}^{n+1}(2,j,k) - E_{y/z}^{n}(1,j,k)\right) \\ E_{y/z}^{n+1}(x_m,j,k) &= E_{y/z}^{n}(x_m-1,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_{y/z}^{n+1}(x_m-1,j,k) - E_{y/z}^{n}(x_m,j,k)\right) \end{aligned} \tag{2.36}$$

Mur'sche Randbedingung zweiter Ordnung Eine genauere Näherung des Wurzelausdruckes in Gln. (2.31) lautet nach Mur:

$$\sqrt{1-(c_0\,s_y)^2-(c_0\,s_z)^2} \approx 1 - \frac{1}{2}(c_0\,s_y)^2 - \frac{1}{2}(c_0\,s_z)^2 \tag{2.37}$$

Damit wird aus Gln. (2.31):

$$s_x \approx \frac{1}{c_0}\left(1 - \frac{1}{2}(c_0\,s_y)^2 - \frac{1}{2}(c_0\,s_z)^2\right) \tag{2.38}$$

Eingesetzt in Gln. (2.30) ergibt sich:

$$\left(\frac{\partial\psi}{\partial x} - \frac{1}{c_0}\frac{\partial\psi}{\partial t} - \frac{c_0}{2}s_y^2\frac{\partial\psi}{\partial t} - \frac{c_0}{2}s_z^2\frac{\partial\psi}{\partial t}\right)_{x=x_0} = 0 \tag{2.39}$$

Die zeitliche Ableitung ergibt dann:

$$\left(\frac{\partial^2\psi}{\partial x\partial t} - \frac{1}{c_0}\frac{\partial^2\psi}{\partial t^2} - \frac{c_0}{2}s_y^2\frac{\partial^2\psi}{\partial t^2} - \frac{c_0}{2}s_z^2\frac{\partial^2\psi}{\partial t^2}\right)_{x=x_0} = 0 \tag{2.40}$$

Da für die inversen Geschwindigkeiten

$$\begin{aligned} s_y &= \frac{1}{v_y} = \frac{t}{y} \\ s_z &= \frac{1}{v_z} = \frac{t}{z} \end{aligned} \tag{2.41}$$

gilt, kann durch diese Ersetzung Gln. (2.40) zu

$$\left(\frac{\partial^2\psi}{\partial x\partial t} - \frac{1}{c_0}\frac{\partial^2\psi}{\partial t^2} - \frac{c_0}{2}\left(\frac{\partial^2\psi}{\partial y^2} - \frac{\partial^2\psi}{\partial z^2}\right)\right)_{x=x_0} = 0 \tag{2.42}$$

umgeformt werden. Für die Diskretisierung der partiellen Ableitungen gilt hier:

$$\begin{aligned} \frac{\partial^2 f(s,t)}{\partial s^2} &\approx \frac{f(s_2,t) - 2f(s_1,t) + f(s_0,t)}{\Delta s^2} \\ \frac{\partial^2 f(s,t)}{\partial t^2} &\approx \frac{f(s,t_2) - 2f(s,t_1) + f(s,t_0)}{\Delta t^2} \\ \frac{\partial^2 f(s,t)}{\partial t\partial s} &\approx \frac{f(s_1,t_2) - f(s_1,t_0) - f(s_0,t_2) + f(s_0,t_0)}{2\,\Delta t\,\Delta s} \end{aligned} \tag{2.43}$$

Die Diskretisierung wird für die Zeitkoordinate n und für die Ortskoordinate $i+1$ durchgeführt. Die Diskretisierung des ersten Terms von Gln. (2.42) impliziert bereits die gewünschte Ortskoordinate. Für die anderen Terme wird wie bei der Mur'schen Randbedingung erster Ordnung eine Interpolation durchgeführt. Unter diesen Voraussetzungen erhält man:

$$
\begin{aligned}
0 \;=\; & \frac{\psi_{i+1}^{n+1} - \psi_{i+1}^{n-1} - \psi_{i}^{n+1} + \psi_{i}^{n-1}}{2\,\Delta x\,\Delta t} \\
& - \frac{1}{c_0}\frac{1}{2}\left(\frac{\psi_{i+1}^{n+1} - 2\psi_{i+1}^{n} + \psi_{i+1}^{n-1}}{\Delta t^2} + \frac{\psi_{i}^{n+1} - 2\psi_{i}^{n} + \psi_{i}^{n-1}}{\Delta t^2} \right) \\
& + \frac{c_0}{2}\frac{1}{2}\left(\frac{\psi_{i+1}^{j+1} - 2\psi_{i+1}^{j} + \psi_{i+1}^{j-1}}{c_0\,\Delta y^2} + \frac{\psi_{i}^{j+1} - 2\psi_{i}^{j} + \psi_{i}^{j-1}}{c_0\,\Delta y^2} \right) \\
& + \frac{c_0}{2}\frac{1}{2}\left(\frac{\psi_{i+1}^{k+1} - 2\psi_{i+1}^{k} + \psi_{i+1}^{k-1}}{c_0\,\Delta z^2} + \frac{\psi_{i}^{k+1} - 2\psi_{i}^{k} + \psi_{i}^{k-1}}{c_0\,\Delta z^2} \right)
\end{aligned}
\tag{2.44}
$$

Die Auflösung von Gln. (2.44) nach ψ_i^{n+1} ergibt wiederum die allgemeine Aktualisierungsgleichung für die „linke“ Wand.

$$
\begin{aligned}
\psi_i^{n+1} \;=\; & -\psi_{i+1}^{n-1} + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(\psi_{i+1}^{n+1} + \psi_i^{n-1}\right) + \frac{2\,\Delta x}{c_0\,\Delta t + \Delta x}\left(\psi_{i+1}^{n} + \psi_i^{n}\right) \\
& + \frac{\Delta x (c_0\,\Delta t)^2}{2\,\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(\psi_{i+1}^{j+1} - 2\psi_{i+1}^{j} + \psi_{i+1}^{j-1} + \psi_i^{j+1} - 2\psi_i^{j} + \psi_i^{j-1}\right) \\
& + \frac{\Delta x (c_0\,\Delta t)^2}{2\,\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(\psi_{i+1}^{k+1} - 2\psi_{i+1}^{k} + \psi_{i+1}^{k-1} + \psi_i^{k+1} - 2\psi_i^{k} + \psi_i^{k-1}\right)
\end{aligned}
\tag{2.45}
$$

Die konkreten Aktualisierungsgleichungen ergeben sich wie für Gln. (2.35) gezeigt. Hierbei ist zu beachten, dass die Werte der Zelle von zwei Vorgängerzeitschritten gespeichert werden müssen. Auch werden mehr Nachbarwerte zur Berechnung der Randzellen benötigt. Das führt dazu, dass die Kantenzellen nicht nach obiger Aktualisierungsgleichung berechnet werden können, da einige der benötigten Nachbarzellen nicht existieren. Für diese muss die Mur'sche ABC erster Ordnung verwendet werden.

2.1.5. Separate-Field Ansatz

In [KL93] wird eine alternative Methode zur Implementierung der FDTD vorgeschlagen. Die Autoren trennen strikt zwischen einfallendem und gestreutem Feld. Ersteres kann für den gesamten

Problemraum analytisch vorgegeben bzw. berechnet werden. Es ist definiert als das Feld, welches ohne einen Streukörper im Raum existieren würde. Das Streufeld hingegen entsteht durch die Interaktion des einfallenden Feldes mit einem Streukörper und muss numerisch berechnet werden. Der Vorteil dieses Ansatzes besteht darin, dass die Amplitude des Streufeldes meist viel kleiner ist, als die des einfallenden Feldes. Da die absorbierenden Randbedingungen hier nur das kleinere Streufeld behandeln müssen, ist die Absorptionsperformance bezogen auf das Gesamtfeld besser. Trotz dieses Vorteils muss beim Einsatz auf diverse Eigenheiten geachtet werden. Beispielsweise versagt dieser Ansatz in manchen Fällen. Man stelle sich z.B. eine Quelle außerhalb einer Hohlkugel vor, deren Oberfläche perfekt leitend ist. Im Inneren soll sich ein weiterer Streukörper befinden. Durch die Abschirmung sollte innerhalb der Kugel keinerlei Feld zu beobachten sein. Da der Separate-Field Ansatz nun aber von einem einfallenden Feld ausgeht, das jeden Körper ignoriert, wird der Streukörper innerhalb der Kugel ein Streufeld erzeugen, welches durch die korrekte FDTD-Ausbreitung die Kugel allerdings nicht verlassen wird. Ein weiterer Nachteil ist die konstante Ausbreitungsgeschwindigkeit der einfallenden Welle. Da Streukörper von dieser ignoriert werden, kann der Einfluss der Materialkonstanten auf die Ausbreitungsgeschwindigkeit nicht berücksichtigt werden. Somit wird die Anwesenheit einer einfallenden Welle an einem Ort bereits angenommen, obwohl diese in Wirklichkeit aufgrund der langsameren Ausbreitung innerhalb eines Körpers auf dem Weg noch nicht angekommen ist.

Trotzdem soll die Herleitung des Ansatzes hier kurz aufgezeigt werden. Da die Entwicklung ähnlich der normalen FDTD verläuft, wird diese hier nur grob besprochen. Es wird sich herausstellen, dass es mit den resultierenden Aktualisierungsgleichungen problemlos möglich ist, das Gesamtfeld auf herkömmliche Weise zu berechnen. Hierzu ist es lediglich nötig, dass einfallende Feld zu vernachlässigen und die Quellen wie bereits besprochen direkt einzusetzen.

Ausgangspunkt der Überlegungen ist die Aufteilung der Felder in deren Bestandteile.

$$\vec{E}^{total} = \vec{E}^{t} \equiv \vec{E}^{ein} + \vec{E}^{streu} = \vec{E}^{i} + \vec{E}^{s} \tag{2.46}$$

$$\vec{H}^{total} = \vec{H}^{t} \equiv \vec{H}^{ein} + \vec{H}^{streu} = \vec{H}^{i} + \vec{H}^{s} \tag{2.47}$$

Aus Gln. (2.7), (2.8) ergibt sich unter Einbeziehung von Gln. (2.11) und unter Beachtung von

Gln. (2.47):

$$rot\left(\vec{H}^i + \vec{H}^s\right) = \epsilon\frac{\partial\left(\vec{E}^i + \vec{E}^s\right)}{\partial t} + \sigma\left(\vec{E}^i + \vec{E}^s\right) \tag{2.48}$$

$$rot\left(\vec{E}^i + \vec{E}^s\right) = -\mu\frac{\partial\left(\vec{H}^i + \vec{H}^s\right)}{\partial t} - \sigma^*\left(\vec{H}^i + \vec{H}^s\right) \tag{2.49}$$

Für das ungestörte einfallende Feld gelten Freiraumbedingungen ($\mu = \mu_0$, $\epsilon = \epsilon_0$, $\sigma^* = 0$ und $\sigma = 0$):

$$rot\vec{H}^i = \epsilon_0\frac{\partial\vec{E}^i}{\partial t} \tag{2.50}$$

$$rot\vec{E}^i = -\mu_0\frac{\partial\vec{H}^i}{\partial t} \tag{2.51}$$

Werden die Gln. (2.51) und (2.51) von den Gln. (2.49) und (2.49) subtrahiert, so ergeben sich:

$$rot\vec{H}^s = \epsilon\frac{\partial\vec{E}^s}{\partial t} + \sigma\vec{E}^s + (\epsilon - \epsilon_0)\frac{\partial\vec{E}^i}{\partial t} + \sigma\vec{E}^i \tag{2.52}$$

$$rot\vec{E}^s = -\mu\frac{\partial\vec{H}^s}{\partial t} - \sigma^*\vec{H}^s - (\mu - \mu_0)\frac{\partial\vec{H}^i}{\partial t} - \sigma^*\vec{H}^i \tag{2.53}$$

Umgestellt nach den Streufeldern und unter Vernachlässigen der magnetischen Leitfähigkeit σ^* erhält man:

$$\sigma\vec{E}^s + \epsilon\frac{\partial\vec{E}^s}{\partial t} = -(\epsilon - \epsilon_0)\frac{\partial E^i}{\partial t} + \left(\nabla\times\vec{H}^s\right) - \sigma\vec{E}^i \tag{2.54}$$

$$\mu\frac{\partial\vec{H}^s}{\partial t} = -(\mu - \mu_0)\frac{\partial\vec{H}^i}{\partial t} - \left(\nabla\times\vec{E}^s\right) \tag{2.55}$$

Mittels des bekannten Differenzierungsschema aus Gln. (2.17) ergibt sich analog zu Gln. (2.19):

$$
\begin{aligned}
E_x^{s,n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_x^{s,n}(i,j,k) \quad &+ \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_z^{s,n}(i,j,k) - H_z^{s,n}(i,j-1,k)\right) \\
&- \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_y^{s,n}(i,j,k) - H_y^{s,n}(i,j,k-1)\right) \\
&- \quad \frac{\sigma\,\Delta t}{\epsilon+\sigma\,\Delta t}E_x^{i,n+1}(i,j,k) \\
&- \quad \frac{(\epsilon-\epsilon_0)\Delta t}{\epsilon+\sigma\,\Delta t}dE_x^{i,n+1}(i,j,k) \\
\\
E_y^{s,n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_y^{s,n}(i,j,k) \quad &+ \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_x^{s,n}(i,j,k) - H_x^{s,n}(i,j,k-1)\right) \\
&- \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_z^{s,n}(i,j,k) - H_z^{s,n}(i-1,j,k)\right) \\
&- \quad \frac{\sigma\,\Delta t}{\epsilon+\sigma\,\Delta t}E_y^{i,n+1}(i,j,k) \\
&- \quad \frac{(\epsilon-\epsilon_0)\Delta t}{\epsilon+\sigma\,\Delta t}dE_y^{i,n+1}(i,j,k) \\
\\
E_z^{s,n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_z^{s,n}(i,j,k) \quad &+ \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_y^{s,n}(i,j,k) - H_y^{s,n}(i-1,j,k)\right) \\
&- \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_x^{s,n}(i,j,k) - H_x^{s,n}(i,j-1,k)\right) \\
&- \quad \frac{\sigma\,\Delta t}{\epsilon+\sigma\,\Delta t}E_z^{i,n+1}(i,j,k) \\
&- \quad \frac{(\epsilon-\epsilon_0)\Delta t}{\epsilon+\sigma\,\Delta t}dE_z^{i,n+1}(i,j,k)
\end{aligned}
\tag{2.56}
$$

Die Terme bezüglich des einfallenden Feldes (E^i und dE^i) müssen von entsprechenden Funktionen analytisch zur Verfügung gestellt werden. Für die Aktualisierung des Magnetfeldes kann analog vorgegangen werden. Auf die Darstellung wird an dieser Stelle verzichtet. Im Anhang (A) können alle Aktualisierungsgleichungen nochmals nachgeschlagen werden.

Aus dem Vergleich von Gln. (2.20) und (2.56) ist leicht zu ersehen, dass beide, bis auf die Terme bezüglich der einfallenden Welle, identisch sind. Um Gln. (2.56) als normale FDTD verwenden zu können, muss das einfallende Feld lediglich zu Null gesetzt werden.

2.2. Optimierungsverfahren

2.2.1. Einleitung

Der in dieser Arbeit untersuchte Ansatz zur Lösung des inversen Streuproblemes beruht auf der Schätzung der Verteilung der dielektrischen Eigenschaften eines Streukörpers. Ausgehend von den bekannten Quellen wird die Feldverteilung mittels der Vorwärtsrechnung (siehe Kapitel 2.1-FDTD) unter Einbeziehung der angenommenen Parameter bestimmt. Die resultierenden Feldkomponenten werden an den Orten extrahiert, an denen sich auch die realen Messpunkte befinden. Die Differenz, die sich aus den gemessenen und berechneten Feldern ergibt, ist Ausgangspunkt für die zu minimierende Kostenfunktion, die damit abhängig von den gewählten Parametern ist.

$$\min_x K(\mathbf{p}) \tag{2.57}$$

Um diese Minimierung durchzuführen gibt es verschiedene Optimierungsverfahren, die in einem iterativen Zyklus einen Satz an neuen Parametern bestimmen, um die Lösung zu verbessern und das berechnete Feld an das gemessene Feld anzugleichen.
Neben den rein deterministischen Verfahren zur nichtlinearen Optimierung gibt es noch die große Gruppe der heuristischen Optimierungsmethoden. Diese sind für die gegebene Aufgabe sehr interessant, da sie die Möglichkeit zur globalen Optimierung bieten. Die Aufgabe der vorliegenden Arbeit bestand aber in der Untersuchung der erstgenannten Methoden. Da diese Verfahren eine lokale Optimierung durchführen, die Zielfunktion mitunter aber multimodale Eigenschaften aufweist, ist davon auszugehen, dass das Ergebnis stark vom gewählten Startpunkt (sprich der geschätzten Parameterverteilung) abhängig ist.

2.2.2. Gradientenabstiegsverfahren

Dieses Verfahren bestimmt den Gradienten, das heißt, die Richtung des steilsten Abstieges der Kostenfunktion. Nachdem die Richtung bekannt ist, wird eine Schrittweite gewählt. Diese kann

beispielsweise über eine eindimensionale Liniensuche ermittelt werden. Hierbei wird eine Folge von Schrittweiten δ^j gewählt und die Kostenfunktion an den Stellen $K(\mathbf{p}-\delta^j\nabla K(\mathbf{p}))$ ausgewertet. Hat sich die Kostenfunktion um einen definierten Wert vermindert oder nimmt der Fehler wieder zu, so wird die Suche abgebrochen und die jeweilige Schrittweite verwendet. Eine einfache Möglichkeit besteht in der *Methode der Halbierung der Schrittweite* mit $\delta^j = 2^{-j}$. Die Aktualisierungsgleichung des Gradientenverfahrens lautet:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \delta^i \nabla K\left(\mathbf{p}^i\right) \tag{2.58}$$

Dieses Verfahren konvergiert vor allem in der Nähe des lokalen Optimum sehr langsam und neigt zum Oszillieren.

2.2.3. Newton Verfahren

Das Newton Verfahren wird üblicherweise benutzt, um Nullstellen von ein- oder mehrdimensionalen Funktionen zu bestimmen. Es kann aber auch so abgewandelt werden, um lokale Extremstellen zu finden. Dies ist leicht einzusehen, wenn man bedenkt, dass ein solcher Extremwert auch eine Nullstelle der Ableitung der Funktion darstellt. Wird folgende Taylorentwicklung[6] betrachtet

$$K(\mathbf{p}+\Delta\mathbf{p}) \approx K(\mathbf{p}^i) + K^{'}(\mathbf{p}^i)\Delta\mathbf{p} + \frac{1}{2}K^{''}(\mathbf{p}^i)\Delta\mathbf{p}^2 \tag{2.59}$$

ergeben sich durch Ableitung nach $\Delta\mathbf{p}$:

$$K^{'}(\mathbf{p}+\Delta\mathbf{p}) \approx K^{'}(\mathbf{p}^i) + K^{''}(\mathbf{p}^i)\Delta\mathbf{p} \tag{2.60}$$

Um $K(\mathbf{p}+\Delta\mathbf{p})$ zu maximieren gilt für den nächsten Schritt:

[6] http://www.stat.washington.edu/quinn/classes/560/Newton.pdf]

$$K^{'}(\mathbf{p}^i) + K^{''}(\mathbf{p}^i)\Delta\hat{\mathbf{p}^i} = 0 \tag{2.61}$$

Nach $\Delta\hat{\mathbf{p}^i}$ umgestellt und nach Addition von $\mathbf{p}^i$ auf beiden Seiten erhält man:

$$\mathbf{p}^i + \Delta\hat{\mathbf{p}^i} = \mathbf{p}^i - \frac{K^{'}(\mathbf{p}^i)}{K^{''}(\mathbf{p}^i)} \tag{2.62}$$

Wird die linke Seite nun als neue Parameterschätzung interpretiert und die erste Ableitung der Kostenfunktion durch den Gradienten sowie die zweite Ableitung durch die Hessematrix ersetzt, so ergibt sich die Aktualisierungsgleichung:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - H_K(\mathbf{p}^i)^{-1}\nabla K(\mathbf{p}^i) \tag{2.63}$$

Wird davon $\mathbf{p}^i$ subtrahiert und die Hessematrix auf die linke Seite gebracht, folgt daraus die Normalform mit $\Delta\mathbf{p} = \mathbf{p}^{i+1} - \mathbf{p}^i$:

$$H_K(\mathbf{p}^i)\Delta\mathbf{p}^i = -\nabla K(\mathbf{p}^i) \tag{2.64}$$

Im Vergleich mit dem Gradientenverfahren weist dieses eine bessere Konvergenz auf. Diese Eigenschaft wird aber durch die geringere Robustheit gegenüber schlechten Anfangswerten erkauft. Desweiteren ist die genaue Bestimmung der Hessematrix ein nicht zu unterschätzendes Problem. Im Quasi-Newton Verfahren wird sie deshalb nur angenähert. Dieses Verfahren wird hier aber nicht genauer betrachtet, statt dessen folgt an dieser Stelle die Beschreibung des Verfahrens, welches am häufigsten zum Einsatz kommt, wenn es um nichtlineare Optimierung geht.

2.2.4. Gauß-Newton Verfahren

Beim Gauß-Newton Verfahren entsteht durch Anwendung der Methode der kleinsten Fehlerquadrate ein nichtlineares Ausgleichsproblem, welches zur Optimierung herangezogen wird.

$$\sum_{k=1}^{N} \left(FDTD(x_k, y_k, z_k : p1, p2..) - E^{meas}(x_k, y_k, z_k)\right)^2 \longrightarrow min \tag{2.65}$$

Das Verfahren geht davon aus, dass ein Satz von N Messwerten zur Verfügung steht. Ebenfalls existiere eine parametrierbare Modellfunktion, die für beliebige Variablen ein eindeutiges Ergebnis liefert. Hierbei ist zu beachten, dass die Abbildung der Variablen auf das Ergebnis von P Parametern abhängig ist. Mit dem Gauß-Newton Verfahren ist es nun möglich, in einem iterativen Verfahren die Parameter so zu bestimmen, dass die Ergebnisse der Modellfunktion möglichst genau die Messwerte approximieren.
Im betrachteten Fall existieren die Werte des EM-Feldes an bestimmten Messpunkten. Die FDTD stellt die Modellfunktion dar. Die Verteilung der elektrischen und magnetischen Eigenschaften innerhalb des Berechnungsraumes gehen dabei als Parameter ein. Wird nun die FDTD ausgeführt und so die Feldverteilung an den Messpunkten rechnerisch ermittelt, ergeben sich Abweichungen von den gemessenen Werten, die das Verfahren zu minimieren versucht (Minimierung der Fehlerquadrate).
Zur Anwendung der Methode wird als erstes einen Vektor mit den letzten in der Modellfunktion verwendeten Parametern $\mathbf{p}^i$ benötigt. Weiterhin ist ein Residuenvektor $\mathbf{r}^i(\mathbf{p}^i)$ oder kurz $\mathbf{r}^i$ erforderlich. Dieser enthält die Differenz der mittels Modellfunktion berechneten Werte von den realen Messdaten. In den vorher genannten Verfahren war das ganz allgemein die Kostenfunktion $K(\mathbf{p})$. Als letztes muss noch eine Matrix bestimmt werden, welche die ersten partiellen Ableitungen der Residuen nach den Parametern enthält. Diese Matrix wird Jacobimatrix $\mathbf{J}^i(\mathbf{r}^i, \mathbf{p}^i)$ genannt und besitzt folgenden Aufbau:

$$\mathbf{J}^i(\mathbf{r}^i, \mathbf{p}^i) = \mathbf{J}^i = \begin{pmatrix} \frac{\partial \mathbf{r}_1^i}{\partial \mathbf{p}_1^i} & \frac{\partial \mathbf{r}_1^i}{\partial \mathbf{p}_2^i} & \cdots & \frac{\partial \mathbf{r}_1^i}{\partial \mathbf{p}_P^i} \\ \frac{\partial \mathbf{r}_2^i}{\partial \mathbf{p}_1^i} & \frac{\partial \mathbf{r}_2^i}{\partial \mathbf{p}_2^i} & \cdots & \frac{\partial \mathbf{r}_2^i}{\partial \mathbf{p}_P^i} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{r}_N^i}{\partial \mathbf{p}_1^i} & \frac{\partial \mathbf{r}_N^i}{\partial \mathbf{p}_2^i} & \cdots & \frac{\partial \mathbf{r}_N^i}{\partial \mathbf{p}_P^i} \end{pmatrix} \tag{2.66}$$

Ausgehend von einer Parameterschätzung werden in jedem Iterationsschritt wie in den vorher vorgestellten Verfahren ein neuer Parametervektor berechnet. Die Aktualisierungsgleichung entspricht in etwa der des Newton-Verfahrens. Allerdings wird die Hessematrix hier durch das Produkt der Jacobimatrix mit seiner Transponierten ersetzt. Dieses Produkt wird auch als Gauß-Hesse-Matrix bezeichnet. Diese ist in der Praxis besser zu ermitteln und entspricht einer Annäherung an die Hessematrix. Das bedeutet eine Zeitersparnis bei der numerischen Berechnung. Mit dieser Approximation ergibt sich folgende Aktualisierungsgleichung für den Parametervektor:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \left(\left(\mathbf{J}^i\right)^T \mathbf{J}^i \right)^{-1} \left(\mathbf{J}^i\right)^T \mathbf{r}^i \tag{2.67}$$

Die Normalform von Gln. (2.67) lautet:

$$\mathbf{G}^i \Delta \mathbf{p}^i = - \left(\mathbf{J}^i\right)^T \mathbf{r}^i \tag{2.68}$$

mit $\mathbf{G}^i = \left(\mathbf{J}^i\right)^T \mathbf{J}^i$ und $\Delta \mathbf{p}^i = \mathbf{p}^{i+1} - \mathbf{p}^i$.

Bei Gln. (2.67) und (2.68) handelt es sich um die Aktualisierungsgleichungen für überbestimmte Probleme, in denen die Zahl der unbekannten Parameter P kleiner ist als die Zahl der Gleichungen und damit der Messpunkte N. Für den Fall eines unterbestimmten Gleichungssystems gilt nach [Fan04]:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \mathbf{J}^i \left(\mathbf{J}^i \left(\mathbf{J}^i\right)^T \right)^{-1} \mathbf{r}^i \tag{2.69}$$

2.2.5. Levenberg-Marquardt Verfahren

Das Levenberg-Marquardt Verfahren erfreut sich zunehmender Beliebtheit. Diese Technik kombiniert die beiden Optimierungsalgorithmen *Gradientenverfahren* und *Gauß-Newton Verfahren*. Dazu wird der stets positive Dämpfungsfaktor λ eingeführt. Die Aktualisierungsgleichung lautet:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \left(\left(\mathbf{J}^i\right)^T \mathbf{J}^i + \lambda \mathbf{I} \right)^{-1} \left(\mathbf{J}^i\right)^T \mathbf{r}^i \tag{2.70}$$

Hierbei ist $\mathbf{I}$ die Einheitsmatrix der Dimension $P \times P$. Umgestellt ergibt sich entsprechend die Normalform:

$$\left(\left(\mathbf{J}^i\right)^T \mathbf{J}^i + \lambda \mathbf{I} \right) \Delta \mathbf{p}^i = - \left(\mathbf{J}^i\right)^T \mathbf{r}^i \tag{2.71}$$

λ kann nun dazu benutzt werden, zwischen den beiden Verfahren zu wechseln. Je kleiner der Dämpfungsfaktor ist, desto weniger Einfluss hat er auf die Gleichung und das Verfahren nähert sich dem Gauß-Newton Verfahren an. Mit steigendem Dämpfungswert wechselt das Verfahren von der Gauß-Newton-Richtung zur Gradientenrichtung. Die Dämpfung λ kann nun mit jeder Iteration neu bestimmt werden, je nachdem, ob sich das Ergebnis verbessert hat (Residuum kleiner) oder nicht. Üblich ist eine Multiplikation bzw. Division mit Werten zwischen 2 und 10. Es gibt aber auch adaptive Methoden, die λ dynamisch verändern.
Der Vorteil des Verfahrens ist seine Robustheit gegen schlechte Startwerte und die überlineare Konvergenz in der Nähe des Optimums.

3. Implementierung

Im Kapitel über die theoretischen Grundlagen (Kapitel 2) wurden ausgehend von den Maxwellschen Gleichungen die Aktualisierungsgleichungen der FDTD (Kapitel 2.1) hergeleitet und die Prinzipien der numerischen Optimierung nichtlinearer Probleme (Kapitel 2.2) dargestellt. In diesem Abschnitt folgen Erläuterungen zur Implementierung dieser grundlegenden Verfahren. Hierbei werden zuerst die beiden verwendeten Programmiersprachen und die jeweiligen Entwicklungsumgebungen vorgestellt und erläutert, weshalb sie für die jeweilige Aufgabe ausgewählt wurden. Im Anschluss daran folgt die Beschreibung der prinzipiellen Abläufe innerhalb der Programme, wobei auf ausgewählte Punkte detaillierter eingegangen wird. Der Quellcode der FDTD sowie ausführliche Ablaufdiagramme sind im Anhang (B) und (C) sowie auf der beiliegenden CD-ROM[7] zu finden.

3.1. Verwendete Programmiersprachen

3.1.1. FORTRAN zur Implementierung der FDTD

„Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.“ (Antoine de Saint Exupery)

Bei FORTRAN handelte es sich um eine der ersten Hochsprachen für Computer. Hauptsächlich für numerische Berechnungen im wissenschaftlichen und ingenieurtechnischen Umfeld eingesetzt, wurde sie sehr restriktiv und minimalistisch entworfen. Eine handvoll Kontrollstrukturen, prozeduale und funktionale Modularisierung und, seit der Version Fortran90, eine dynamische Speicherverwaltung halten den Sprachumfang in überschaubaren Grenzen. Auch wenn eine Vielzahl an Bibliotheken für diese Programmiersprache existieren, lässt die kompakte Implementierung die Erstellung hochoptimierender Compiler zu. Ein weiterer Vorteil ist die kurze Einarbeitungszeit, um eigene Probleme in FORTRAN zu formulieren und zu lösen. Diese Eigenschaften haben dazu geführt, dass FORTRAN in seinen verschiedenen Versionen bis heute weite Verbreitung gefunden hat, wenn es um das sogenannte „numbercrunching“ geht. Für aktuelle, parallel ar-

[7] Inhaltsverzeichnis der CD befindet sich im Anhang (E)

beitende Computer gibt es mit der OpenMP [8] Bibliothek auch ein effektives Werkzeug, um in FORTRAN Programmen Nutzen aus der Parallelität ziehen zu können.

Intel® Fortran95 Compiler Intel® bietet auf der Homepage[9] seinen aktuellen Fortran Compiler in Versionen für Linux und Windows® an. Für Studenten ist der Download zur nichtkommerziellen Benutzung kostenfrei. In Kapitel (6.1) ist dessen Installation und die Integration in MATLAB® am Beispiel der Linux-Version dargestellt. Der Intel® Compiler besticht durch seine hervorragende Performance[10]. Weiterhin implementiert er die Fortran95 Spezifikation und unterstützt damit dynamische Arrays, welche durch die Verknüpfung von MATLAB® und FORTRAN für die vorliegende Arbeit von Vorteil waren. Da FORTRAN prinzipiell plattformübergreifend funktioniert und Intel® seinen Compiler auch für Windows® -Systeme anbietet, ist die Interoperalität gewährleistet.

gedit zur Quellcodeerstellung Als Entwicklungsumgebung kam keine dedizierte IDE zum Einsatz. Statt dessen wurde der Texteditor *gedit* von GNOME™ (Desktopmanager für Linuxsysteme) verwendet. Da dieser die Syntax-Hervorhebung für verschiedene Programmiersprachen unterstützt, gestaltete sich die Quellcode-Erstellung dennoch sehr komfortabel.

3.1.2. MATLAB® zur Implementierung der Optimierungsaufgabe

MATLAB® hat als umfangreiches Programmpaket zur Lösung verschiedenster numerischer Aufgaben weite Verbreitung gefunden. Es existiert eine Vielzahl an kostenfreien und kommerziellen Bibliotheken, die in MATLAB® integriert werden können, um dessen Funktionalität an die eigenen, speziellen Anforderungen anzupassen. Für die vorliegende Arbeit ist das Basispaket aber völlig ausreichend. Gründe für den Einsatz von MATLAB® waren unter anderem:

- Bekanntheit unter den Studenten
- Verfügbarkeit in den Einrichtungen des Fachbereiches
- Performance und Simplizität beim Lösen von Gleichungssystemen

[8] Ausführliche Informationen finden sich auf der Projektseite http://www.openmp.org
[9] Download unter http://www.intel.com/cd/software/products/asmo-na/eng/compilers/282048.htm
[10] Benchmarks sind unter http://www.polyhedron.co.uk zu finden

- Einfache Handhabung
- Umfangreiche Möglichkeiten zur visuellen Datenauswertung
- Möglichkeit zur Einbindung von (compilierten) FORTRAN Programmen

Da diese Diplomarbeit vollständig unter Linux bearbeitet wurde, kam auch eine entsprechende Programmversion (MATLAB® 7 R14 für Linux) zum Einsatz. Wie für den FORTRAN Compiler gilt auch hier, dass die erstellten Programme und Funktionen ohne größeren Aufwand auf Windows® basierte Systeme portiert werden können.

3.2. Realisierung

In Abbildung (3) ist die grundsätzliche Vorgehensweise dargestellt, um mittels Vorwärtsrechnung (FDTD) und nichtlinearer Optimierung die Verteilung der elektrischen Parameter eines Streukörpers zu rekonstruieren [Yee66]. Der gesamte Ablauf ist in MATLAB® realisiert. Die einzige Ausnahme davon stellt die FDTD dar. Deren Funktionalität wurde auf ein FORTRAN Programm abgebildet, welches sich von MATLAB® aus konfigurieren und ausführen lässt. Da die Implementierung der FDTD den Ausgangspunkt der Arbeiten darstellte, soll auch hier mit der Beschreibung des FORTRAN-Codes begonnen werden.

3.2.1. FDTD mittels FORTRAN

Es gibt bereits diverse Implementierungen der FDTD in FORTRAN. Da der Code von Kunz und Lübbers [KL93] im Fachbereich Verwendung findet, wurde die Implementierung für diese Arbeit darauf aufbauend realisiert. Aus diesem Grund findet sich auch der in Kapitel (2.1.5) vorgestellte Separate-Field Ansatz im vorliegenden Quellcode wieder. Aufgrund der speziellen Anforderungen (Kommunikation mit MATLAB®) wurde der FORTRAN77 Code auf die neuere Version Fortran9x umgestellt, um unter anderem dynamische Arrays benutzen zu können. Auch lässt sich das Programm nun vollständig von MATLAB® aus konfigurieren und hat deutlich an Lesbarkeit zugenommen. Dies vereinfacht auch die Anpassung an weiterführende Erfordernisse wie zum Beispiel die Implementierung der PML Randbedingungen für auf diese Arbeit aufbauende Untersuchungen. Weiterhin ist es möglich, ein- oder zweidimensionale Wellenausbreitung

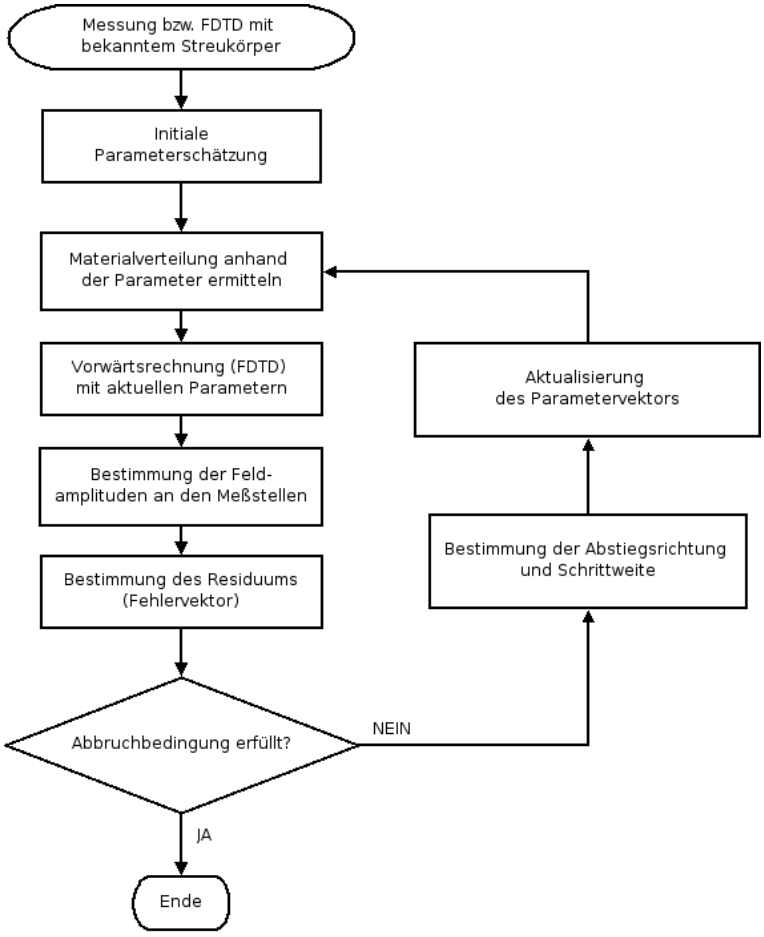

Abbildung 3: **Flussdiagram - Ablauf der Optimierung:** Dieses Flussdiagramm beschreibt die allgemeine Vorgehensweise der nichtlinearen Optimierungsverfahren. Ausgehend von einer initialen Schätzung der beteiligten Parameter wird in einem iterativen Verfahren die Kostenfunktion unter Einbeziehung der aktuellen Parameter berechnet und die Abweichungen des Ergebnisses an definierten Stützstellen mit den gemessenen Werten verglichen. Es folgt die Berechnung der sogenannten Abstiegsrichtung und der Schrittweite, um den Fehler durch Anpassung der Parameter zu minimieren. In der Art der Berechnung von Richtung und Schrittweite unterscheiden sich die verschiedenen Verfahren.

zu simulieren, indem lediglich bestimmte Feldkomponenten berechnet werden. Mehr zu den Verwendungsmöglichkeiten findet sich in Kapitel (6.2).
Der prinzipielle Ablauf der FDTD ist in Abbildung (4) dargestellt.

Konstante Faktoren Wie in den Aktualisierungsgleichungen Gln. (2.56 bzw. 2.20 und 2.22) ersichtlich ist, werden zur Berechnung der neuen Werte einer Feldkomponente andere Komponenten über konstante Faktoren miteinander verknüpft. Diese Faktoren brauchen nicht jedes mal zeitraubend berechnet zu werden. Stattdessen errechnet die Prozedur *SetupFactors* zu Beginn alle benötigten Faktoren für die Aktualisierungsgleichungen des Feldes sowie die der Gleichungen für die absorbierenden Randbedingungen. Vorher ermittelt die Prozedur *SetupConstants* unter anderem die verwendete Zeitschrittlänge Δt, um passend zu den gegebenen Abmessung der Yee-Zellen die Stabilitätsbedingung gemäß Gln. (2.26) einzuhalten.
Zu beachten ist hier der Umstand, dass einige der konstanten Faktoren von den elektrischen Eigenschaften ϵ_r und σ abhängen. Aus diesem Grund wird für jede auftretende Kombination dieser beiden Parameter ein Satz von Faktoren berechnet und jeweils in entsprechenden Arrays gespeichert.

Zeitschritt Kern einer jeden FDTD-Implementierung ist die zeitschrittweise Aktualisierung der einzelnen Feldkomponenten mittels der in Kapitel (2.1) hergeleiteten Aktualisierungsgleichungen. Dazu werden im ersten Halbschritt die Komponenten des elektrischen Feldes neu berechnet. Die entsprechenden Prozeduren lauten hier *CalcEiComponents* wobei das i für die x, y und z -Komponente steht. Im Anschluss daran folgt die Korrektur der Felder an den Rändern des Problemraumes. Die Prozedur *ABCCorrection* führt abhängig von der gewählten Konfiguration die entsprechende ABC-Korrektur aus, um ungewollte Reflexionen zu minimieren. Da das H-Feld im nächsten Halbschritt bereits mit den korrigierten Werten des E-Feldes berechnet wird, müssen die Gleichungen der absorbierenden Randbedingungen lediglich auf die elektrischen Feldkomponenten angewandt werden.
Aus dem selben Grund folgt vor der Aktualisierung des Magnetfeldes noch die optionale Injektion der Quellen, von denen ausgehend die Wellenausbreitung zu beobachten ist.
Als Besonderheit der vorliegenden Implementierung ist zu beachten, dass abhängig von zwei

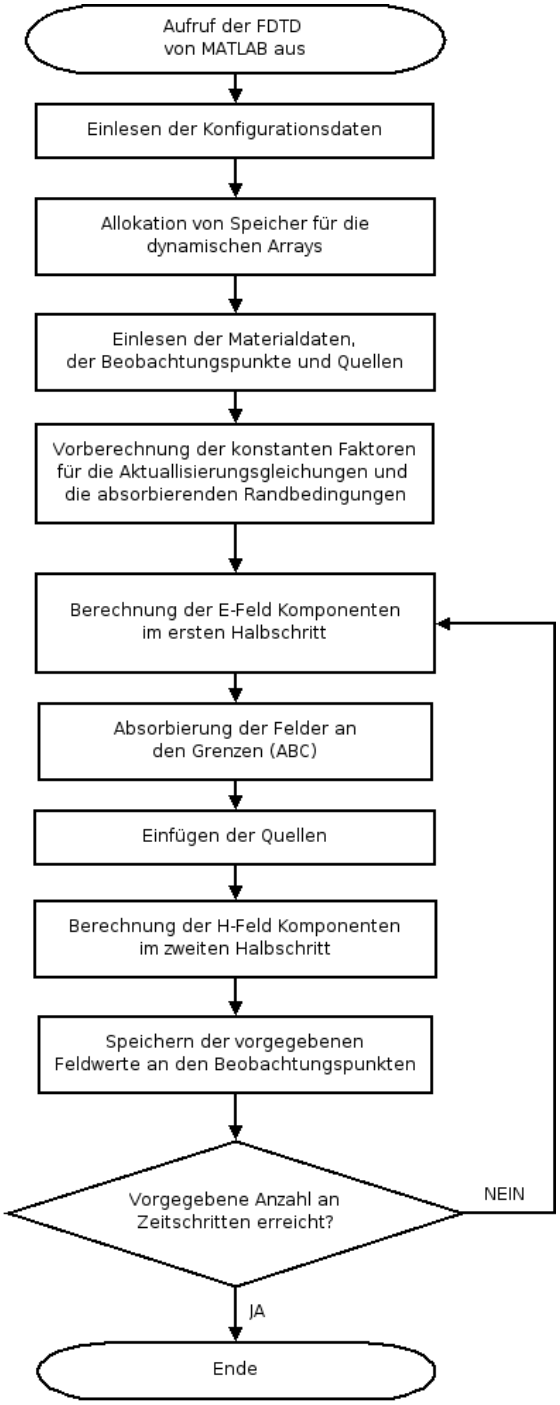

Abbildung 4: **Flussdiagram - Ablauf der FDTD:** Das FDTD-Verfahren besteht darin, für jeden Zeitschritt zunächst die Komponenten des elektrischen Feldes zu aktualisieren, danach die Quellen einzufügen, die Randwerte anzupassen und zuletzt die H-Feld Komponenten ausgehend von dem nun aktuellen E-Feld zu berechnen. Nach jedem Update folgt die Speicherung der definierten Feldwerte um eine spätere Auswertung zu ermöglichen.

Flags nur bestimmte Feldkomponenten aktualisiert werden. Dadurch ist es möglich, eine zweidimensionale (nur E_z, H_y und H_x und damit verbunden Ausbreitung in der $x-y$-Ebene) oder eine eindimensionale Ausbreitung (nur E_z und H_y und damit Ausbreitung in x-Richtung) zu simulieren.

Quellen Die verwendeten Gleichungen zur Aktualisierung der Felder lehnen sich, wie in Kapitel (3.2.1) erwähnt, an den Code von [KL93] an und ermöglichen dadurch die Verwendung des Separate-Field Ansatzes (Kapitel 2.1.5). Auch wenn sich durch dessen Benutzung Vorteile bezüglich der absorbierenden Randbedingungen ergeben, so wurde in dieser Arbeit dennoch darauf verzichtet. Die verwendete direkte Injektion von Quellen lässt es zu, mehrere unabhängige Quellen völlig frei zu platzieren. Dies erweist sich bei der Simulation von Linienquellen und der damit verbundenen Ausbreitung von Zylinderwellen als vorteilhaft. Diese Art von Feld ermöglicht eine anschauliche Visualisierung.
Die Injektion erfolgt dadurch, dass die Feldkomponenten am gewünschten Ort der Quelle mit den aktuellen Quellwerten überschrieben werden. Diese Werte berechnen sich durch die entsprechenden Funktionen. In der vorliegenden Arbeit sind Gauß-Puls-Quellen und harmonische, sinusförmige Quellen implementiert. Jede Quelle kann unabhängig von den anderen platziert und in Typ und Amplitude der einzelnen Komponenten konfiguriert werden.

Aktualisierung der Feldkomponenten Die einzelnen Aktualisierungsgleichungen aus Kapitel (2.1) sind jeweils in eigenen Prozeduren realisiert. Zu beachten ist hierbei, dass bei der Berechnung der E-Feld Werte unterschieden wird, ob es sich bei der aktuellen Zelle um Freiraum, perfekten elektrischen Leiter oder verlustbehaftet elektrisch leitendes Material handelt. Die Gleichungen für die ersten beiden Fälle sind sehr simpel und können schnell berechnet werden. Die benötigte Rechenzeit sinkt damit vor allem für Problemräume mit viel Freiraum. Liegt verlustbehaftetes Material vor, so werden die zum entsprechenden Material gehörigen Faktoren zur Berechnung herangezogen.
Die Gleichungen für das Magnetfeld gehen von magnetisch verlustfreien Materialien aus. Allerdings können die entsprechenden Gleichungen für den verlustbehafteten Fall problemlos nachimplementiert werden, falls dies erforderlich wird.

Absorbierende Randbedingungen Entsprechend zu den Herleitungen aus Kapitel (2.1.4) wurden die Mur'schen Randbedingungen erster und zweiter Ordnung implementiert. Ob und welche der beiden Versionen verwendet wird, kann von MATLAB® aus konfiguriert werden. Bei der Randbedingung erster Ordnung wird davon ausgegangen, dass die Welle senkrecht zur Wand auftrifft. Dementsprechend ist auch anhand des Quellcodes festzustellen, dass an einer Wand lediglich die Feldkomponenten betrachtet werden, die parallel zur jeweiligen Wand liegen. Da die Berechnung Werte vom vorherigen Zeitschritt benötigt, werden diese in entsprechenden Arrays gespeichert. Dies betrifft hier nur die Randzellen und die direkten, inneren Nachbarn.
Die Implementierung der Randbedingung zweiter Ordnung geschieht analog zu der erster Ordnung. Neben den umfangreicheren Aktualisierungsgleichungen und dem Umstand, dass die Werte von zwei vorherigen Zeitschritten benötigt werden ist hier noch zu beachten, dass die Gleichungen nicht auf die Zellen an den Kanten des Problemraumes angewendet werden können. Hier fehlen einige der benötigten Nachbarwerte. Für diese Zellen wird die Mur'sche Randbedingung erster Ordnung verwendet. Der Übersicht halber wurden die Berechnungen der 3 Ebenen (XY, YZ, XZ) auf eigene Prozeduren verteilt.

Beobachtungspunkte Um Informationen aus der Feldberechnung ziehen zu können, müssen Punkte definiert werden, deren Feldverlauf aufzuzeichnen ist. Dies ist nötig, da die FDTD die Raumzellen in jedem Schritt überschreibt. Für die vorliegende Arbeit wurden zwei verschiedene Arten von Beobachtungspunkten definiert. Die erste Art heißt *ObservationPoint*. Es ist möglich, beliebig viele Punkte über die gesamte Zeit beobachten zu lassen. Dazu muss der Ort der Zelle und die aufzuzeichnende Feldkomponente definiert werden. Die Aufzeichnung wird durch die Prozedur *Observe* realisiert, die in jedem Zeitschritt aufgerufen wird.
Die zweite Art von Beobachtungspunkten ergab sich aus der Aufgabenstellung. Für die Optimierung ist es nötig, an bestimmten Messstellen die Feldamplituden im eingeschwungenen Zustand zu extrahieren. Hierfür werden alle Feldkomponenten an den definierten Zellen aufgezeichnet. Um die eingeschwungene Amplitude in MATLAB® zu bestimmen, müssen mindestens zwei zeitlich aufeinanderfolgende Werte gespeichert werden [Fan04]. Zum Ausgleich von Ungenauigkeiten werden die letzten 10 Werte gespeichert, um in MATLAB® später eine Mittelung der berechneten Amplituden durchführen zu können. Die Speicherung dieser Daten übernimmt die Prozedur

FullObservation, die nur für die letzten 10 Zeitschritte aufgerufen wird.

Benötigte Resourcen Da für jede Zelle der Materialtyp und die 6 Feldkomponenten gespeichert werden müssen, wächst der Speicherplatzbedarf sehr schnell mit größer werdendem Problemraum. Auch die benötigten Rechenoperationen und damit die Rechenzeit steigen exponentiell an. Um dieses Problem aufzuzeigen, soll hier eine grobe Resourcenabschätzung vorgenommen werden.

Der benötigte Speicherplatzbedarf wird maßgeblich von den 7 Arrays bestimmt, welche die Informationen (Material, 6 Feldkomponenten) jeder einzelnen Zelle enthalten. Das Material wird durch einen 2 Byte Integerwert festgelegt. Für die Feldwerte werden 8 Byte Fließkommazahlen doppelter Genauigkeit verwendet. Die Abschätzung des Bedarfs an Hauptspeicher ergibt:

$$S_{RAM} \approx \frac{6\,(N_x \cdot N_y \cdot N_z) \cdot 8 + (N_x \cdot N_y \cdot N_z) \cdot 2}{1024 \cdot 1024} MByte \tag{3.1}$$

Dies führt für Problemräume mit einer Kantenlänge von $100\,m$ und einer Zellkantenlänge von $1\,m$ auf einen Bedarf von $S_{RAM} \approx 50\,MByte$. Eine einzelne Komponentenvariable hat damit in etwa eine Größe von $8\,MByte$. Der benötigte Speicher wächst bei kubischem Problemraum ungefähr mit $6 \cdot N_s^3$ [11]. Das Problem wird deutlich, wenn man betrachtet wieviel Speicher für die Berechnung eines Raumes mit $1\,km$ Kantenlänge benötigt wird. Je nach verwendeter Frequenz der Quellen kann es weiterhin nötig sein, die Zellabmessungen nach unten anzupassen. Ausgehend von einer Abmessung vom $1\,m$ für die Zellen[12] ergibt sich ein Speicherbedarf von $S_{RAM} \approx 45\,GByte$. Für ein einzelnes Array einer Feldkomponente sind damit knapp 8 GByte erforderlich. Das zeigt, dass neben der Verfügbarkeit entsprechender Mengen an Arbeitsspeicher auch auf ein Betriebssystem benötigt wird, das Variablen dieser Größe erlaubt[13].

Auch die Anzahl der benötigten Rechenoperationen steigt enorm mit der Größe des Problemraumes an. Den größten Anteil der Rechenzeit belegen die Aktualisierungsgleichungen für die Feldkomponenten. Für die Komponenten des E-Feldes sind zirka 10 Rechenoperationen notwendig. In der aktuellen Implementierung benötigt die Berechnung einer H-Feld Komponente

[11] N_s entspricht der Anzahl der Zellen in jede Raumrichtung
[12] Entspricht unter der Bedingung aus Gln. (2.23) einer maximalen Frequenz von $f_{max} = 30\,MHz$
[13] 64-Bit Systeme

lediglich 6 Operationen.

$$OP_{CPU} \approx 3\,(N_x \cdot N_y \cdot N_z) \cdot 10 + 3\,(N_x \cdot N_y \cdot N_z) \cdot 6 \tag{3.2}$$

Für den Fall eines kubischen Problemraumes ergibt sich eine Anzahl von $OP_{CPU} \approx 48N_s^3$ Operationen. Dieser Wert muss noch mit der Anzahl der benötigten Zeitschritte multipliziert werden (um beispielsweise einen eingeschwungenen Zustand zu erreichen). Wird die Anzahl der Operationen durch die Anzahl der doppelt genauen Fließkommaoperationen dividiert, die die verwendete CPU in der Sekunde durchführen kann, erhält man die Laufzeit für eine einzelne Vorwärtsrechnung.

$$T_{CPU} \approx \frac{48N_s^3 \cdot N_T}{FL_{dp,CPU}} \tag{3.3}$$

3.2.2. Schnittstelle zu MATLAB®

Damit die Funktionalität der FDTD-Lösung in MATLAB® zur Verfügung steht, muss der FORTRAN Code von MATLAB® aus übersetzt werden. Dafür ist die MATLAB® -Funktion *mex* verantwortlich. Wird der Code auf diese Weise übersetzt, so stehen innerhalb des FORTRAN Codes zusätzliche Schnittstellenfunktionen von MATLAB® zur Verfügung. Eine detaillierte Beschreibung findet sich auf der Mathworks Webseite [14] und in der MATLAB® -Hilfe.
Hier folgt jetzt die Beschreibung der Schnittstelle des FDTD-Programms, welche die Konfiguration von MATLAB® aus und die Rückgabe der beobachteten Daten zulässt.

Allgemeine Konfigurationsdaten Bei dem ersten zu übergebenden Parameter handelt es sich um ein eindimensionales Feld mit Angaben über:

- Raumausdehnung (N_x, N_y, N_z)
- Zellgröße (Δx, Δy, Δz)
- Anzahl der Zeitschritte (N_T)

[14] Mex-File Tutorial von Mathworks unter http://www.mathworks.com/support/tech-notes/1600/1605.html

- Frequenz und Amplitude der Quelle für den Separate-Field Ansatz
- Anzahl der Messpunkte (siehe Kapitel 3.2.1 - Beobachtungspunkte)
- Anzahl der allgemeinen Beobachtungspunkte (siehe Kapitel 3.2.1 - Beobachtungspunkte)
- Anzahl der unabhängigen Quellen
- Angabe über die zu verwendende absorbierende Randbedingung
- Angabe über die Verwendung des Separate-Field Ansatzes
- Angabe über die zu berechnenden Feldkomponenten um 2D/1D Ausbreitung zu simulieren

Diese Daten liefern grundsätzliche Informationen für die FDTD wie Raumausdehnung und Zellgröße. Außerdem dienen die Angaben dazu, innerhalb von FORTRAN den benötigten Speicher für die dynamischen Arrays zu reservieren.

Materialdaten Die nächsten beiden Parameter übergeben die Felder mit den verschiedenen Materialkonstanten. Dabei gehören die Werte mit gleichem Index jeweils zum gleichen Material. Die Arrays haben genausoviele Elemente, wie es verschiedene Materialien gibt. Das FORTRAN Programm nutzt diese Informationen zur Berechnung der Faktoren wie in Kapitel (3.2.1) dargestellt.

Materialverteilung Dazugehörig folgt der Parameter mit der Materialverteilung. Es handelt sich hier um ein dreidimensionales Feld, welches jeder Zelle im Problemraum einen Materialwert zuordnet. Dieser Wert entspricht dem Index, unter dem in den beiden vorherigen Parametern die zugehörigen Materialkonstanten zu finden sind. Mit diesem Parameter können also die Streukörper definiert werden.

Messpunkte Der nächste Parameter definiert die Stellen, an denen die Messwerterfassung erfolgen soll. Dieses zweidimensionale Array enthält für jeden Messpunkt eine Zeile, in der die Koordinaten der entsprechenden Zelle stehen. Wie in Kapitel (3.2.1) erklärt wurde, werden für

die hier definierten Zellen die jeweils letzten 10 Werte aufgezeichnet.

Beobachtungspunkte Der vorletzte Parameter ist ganz ähnlich aufgebaut. Er definiert die Stellen, an denen über alle Zeitschritte hinweg eine festgelegte Feldkomponente aufgezeichnet wird. Neben den Koordinaten der Zellen steht in den Zeilen noch die zu überwachende Feldkomponente. Als Besonderheit ist hier anzumerken, dass die E_z Komponente der gesamten $x-y$ Ebene aufgezeichnet wird, sollte dieses Feld eine einfache Zahl sein.

Quellen Zuletzt wird noch ein Parameter erwartet, der die Quellen beinhaltet. Neben den Koordinaten sind hier noch der Quellentyp und die Amplituden der einzelnen Feldkomponenten angegeben.

Rückgabewerte Nach erfolgter Vorwärtsrechnung gibt das FORTRAN Programm über diese Schnittstelle die über die letzten 10 Zeitschritte aufgezeichneten Komponenten der eingestellten Messstellen zurück.
In einem zweiten Rückgabewert übermittelt FORTRAN die intern berechnete Zeitschrittlänge. Diese wird von MATLAB® benötigt, um die Amplitude der Feldkomponenten an den Messstellen zu ermitteln [Fan04].
Als dritten und letzten Rückgabewert übergibt FORTRAN ein Array, welches die ausgewählten Werte der definierten Beobachtungspunkte in jedem Zeitschritt enthält. MATLAB® kann dieses Array nutzen, um die Feldausbreitung zu visualisieren.

3.2.3. Optimierung mittels MATLAB®

Zur Lösung des Optimierungsproblems wird MATLAB® benutzt. Den Ausgangspunkt hierbei bilden normalerweise real gemessene Werte. Da diese für die vorliegende Arbeit nicht zur Verfügung standen, wird einmalig eine Vorwärtsrechnung mittels FDTD durchgeführt um die Messungen zu simulieren. Für diese Rechnung wird ein Streukörper definiert, der im Anschluss durch

den Optimierungsalgorithmus rekonstruiert werden soll.

Konfiguration der FDTD Den Anfang bildet die Konfiguration der Parameter für die FDTD, wie sie in Kapitel (3.2.2) beschrieben wurde. Wichtig sind hier vor allem die Definition der Messpunkte (*Receiverpoints*) und der Quelle. Das Array der Permittivitäten wird mit $\epsilon_r = 1.1 \ldots 40$ vorbelegt. Die Leitfähigkeit wurde für die Versuche vernachlässigt und dessen Feld mit Nullen gefüllt.

Dual-Mesh Für die Rekonstruktion wurde ein einfaches Dual-Mesh Schema eingeführt. Das bedeutet hier, dass die Parameterzellen ein Vielfaches der Größe der FDTD-Zellen besitzen können. Dies ermöglicht es, den Rechenaufwand für die Optimierung in gewissen Grenzen steuern zu können. Zwar sinkt dadurch die Auflösung des Verfahrens, aber da die später benötigte Berechnung der Jacobimatrix ebensoviele Vorwärtslösungen erfordert wie Parameter vorhanden sind, senkt dieses Vorgehen die Berechnungszeit enorm. Zwar wäre es auch möglich die FDTD-Zellen einfach zu vergrößern, allerdings ist man hierbei an die aufzulösenden Wellenlängen gebunden. Unter *BigBlockSize* wird der Faktor angegeben, um den die Parameterzellen in jede Richtung größer sind als die FDTD Zellen. Es ist darauf zu achten, dass die Ausdehnung des FDTD-Raumes ein Vielfaches dieser Blockgröße darstellt.

Definition des Streukörpers Die Parameterzellen können nun einen beliebigen Materialwert zugewiesen bekommen. Die Funktion *ScanSpace4Scatterer* durchsucht den Parameterraum und erstellt ein Array mit den Parameterzellen, die einen Materialwert größer 1 zugeordnet bekommen haben[15]. Anschließend erstellt die Funktion *CreateFDTDGrid* aus dem Parameterraum die Materialzellen für das FDTD Gitter.

Vorwärtsrechnug (Pseudomessung) Mit diesen Daten kann nun die FDTD Routine des Fortranprogramms aufgerufen werden. Aus dem zurückgelieferten Daten werden dann die Feldam-

[15]Ein Wert von 0 bedeutet Freiraum und der Wert 1 steht für perfekte elektrische Leitfähigkeit

plituden an den Messstellen mittels eines Zwei-Punkt Extraktionsverfahrens [Fan04] ermittelt. Um Schwankungen auszugleichen wird dieses Verfahren nicht nur auf die zwei letzten Werte, sondern auf die letzten 10 Werte angewendet und anschließend der Mittelwert der extrahierten Amplituden gebildet. Diese Daten stellen nun sozusagen die gemessenen Daten dar.

Initiale Parameterschätzung Im ersten Schritt der eigentlichen Optimierung wird eine beliebige initiale Parameterverteilung angenommen, indem die Parameterzellen entsprechend belegt werden.

Residuum bestimmen Das Optimierungsverfahren berechnet iterativ in einer Schleife mit den jeweils aktuellen Parametersatz die Feldamplituden an den Messstellen (Vorwärtsrechnung und Amplitudenextraktion) und berechnet die Differenz zu den wahren, gemessenen Werten der Pseudomessung.

Jacobimatrix bestimmen Ist die Abweichung nicht klein genug, wird im Beispiel des Gauß-Newton-Verfahrens die Jacobimatrix bestimmt. Dies ist ein recht rechenaufwendiger Schritt, da jeder Parameter einzeln geändert wird, um mittels FDTD zu bestimmen, wie stark die Abweichungen an den Messstellen auf Änderungen dieses Parameters reagieren.

- Ändern eines Parameters
- Vorwärtsrechnung
- Zurücksetzen des geänderten Parameters
- Amplituden extrahieren
- Differenz zu den Amplituden des ungeänderten Parametervektors an allen Messstellen bestimmen
- Mit den Differenzen die entsprechende Spalte der Jacobimatrix belegen
- Vorgehen für alle Parameter wiederholen

Aktualisieren des Parametervektors Der letzte Schritt innerhalb der Iteration besteht in der Aktualisierung des Parameterverktors. Die Vorgehensweise hier hängt wiederum vom verwendeten Optimierungsalgorithmus ab. Allen gemein ist aber die Notwendigkeit der Lösung eines linearen Gleichungssystems. Mit den neuem Parametervektor geht es wieder an den Anfang der Schleife in die nächste Iteration. Die Optimierung ist beendet, sobald die Abweichung von den wahren Werten einen definierten Wert unterschreiten oder wenn keine weiteren Verbesserungen zu erwarten sind.

4. Versuche und Ergebnisse

Im folgenden Kapitel werden die Implementierungen der FDTD und der Optimierung untersucht. Es werden Simulationen von Feldausbreitungen vorgenommen, die Wirkungsweise und Performance der absorbierenden Randbedingungen getestet und abschließend verschiedene Streukörper mittels der Optimierungsalgorithmen rekonstruiert. Dazu wird jeweils das Ziel der Simulation erklärt, die wichtigsten Einstellungen der zugehörigen MATLAB® - Scripte aufgezeigt und Bilder zur Veranschaulichung dargestellt. Abschließend folgt in den einzelnen Abschnitten die Auswertung der Simulationen.
Sämtliche Versuche wurden über MATLAB®-Scripte realisiert, in denen auch die Erstellung und Benennung der Bilder automatisiert wurde. Alle Scripte sind auf der beiligenden CD zu finden, um die Versuche schnell und einfach nachvollziehen und an eigene Bedürfnisse anpassen zu können.

4.1. Test der Wellenausbreitung mittels FDTD

Die ersten Simulationen haben die Verifikation der prinzipiellen Funktionstüchtigkeit der FDTD-Implementierung zum Ziel. Ausgehend von der einfachen eindimensionalen Ausbreitung einer ebenen Welle werden die verschiedenen Modi bis hin zur kompletten 3D FDTD getestet. Als Quelle kommen je nach Versuch Flächen-, Linien- oder Punktquellen in ihrer harmonischen Version oder als Gauß-Puls zum Einsatz.

4.1.1. 1D Ausbreitung ebener Wellen

Theorie Der einfachste Fall einer Wellenausbreitung ergibt sich aus der eindimensionalen Schwingung der elektrischen und magnetischen Felder. Um diesen Fall zu verdeutlichen, werden verschiedene Annahmen getroffen. Zum ersten soll das E-Feld nur in der x-z-Ebene oszillieren und zum zweiten soll die Ausbreitungsrichtung des EM-Feldes parallel zur x-Achse liegen. Aus der

ersten Forderung ergeben sich:

$$\begin{aligned} \vec{E} &= \begin{pmatrix} E_x \\ 0 \\ E_z \end{pmatrix} \\ \vec{H} &= \begin{pmatrix} H_x \\ H_y \\ H_z \end{pmatrix} \end{aligned} \tag{4.1}$$

Da die Ausbreitungsrichtung immer orthogonal zu den Schwingungsebenen der beiden Felder steht, führt Forderung 2 dazu, dass E-Feld und H-Feld keine x-Komponente besitzen ($E_x = H_x = 0$) und sich Gln.(4.1) weiter vereinfacht.

$$\begin{aligned} \vec{E} &= \begin{pmatrix} 0 \\ 0 \\ E_z \end{pmatrix} \\ \vec{H} &= \begin{pmatrix} 0 \\ H_y \\ H_z \end{pmatrix} \end{aligned} \tag{4.2}$$

Weiterhin stehen auch elektrisches und magnetisches Feld senkrecht aufeinander. Das Skalarprodukt der beiden Feldvektoren muss also 0 ergeben.

$$\begin{aligned} \vec{E} \cdot \vec{H} = E_x H_x + E_y H_y + E_z H_z &= 0 \\ E_z H_z &= 0 \end{aligned} \tag{4.3}$$

Da $E_z \neq 0$ ist, muss in Gln. (4.3) $H_z = 0$ sein. Wir erhalten also ein Feld, dessen E-Feld lediglich eine z-Komponente und dessen H-Feld nur eine y-Komponente aufweist.

$$\begin{aligned} \vec{E} &= \begin{pmatrix} 0 \\ 0 \\ E_z \end{pmatrix} \\ \vec{H} &= \begin{pmatrix} 0 \\ H_y \\ 0 \end{pmatrix} \end{aligned} \tag{4.4}$$

Die Eigenschaften der sich so ausbreitenden Welle entspricht denen, die eine unendlich ausgedehnte Flächenquelle in der y-z-Ebene erzeugen würde. Es gibt natürlich noch weitere Kombinationen von E-Feld, H-Feld und Ausbreitungsrichtung, die sich durch Rotation des Koordinatensystems oder Anpassung der Ausgangsforderungen analog ergeben.

Simulation In dem FDTD Programm wurde diese Art der Wellenausbreitung implementiert, indem nur die beiden genannten Komponenten berechnet werden, sofern dem Programm der entsprechende Parameter übergeben wird.
Das zugehörige MATLAB® Scriptfile enthält unter anderem die in Listing (1) aufgezeigten FDTD Parameter, welche im folgendem kurz besprochen werden.

Listing 1: Matlab-Code - Versuch 1

```
YZdummy = 2;
max_x   = 150;
max_y   = 3;
max_z   = 3;
D1_only = 1;
% Set the whole x-axis to be observated
NoOfObsPoints = 2*max_x;
ObservationPoints = uint16(zeros(NoOfObsPoints,4));
for ObsPoint = 1:NoOfObsPoints/2
        ObservationPoints(ObsPoint,1) = 3;
```

```
        ObservationPoints(ObsPoint,2) = ObsPoint;
        ObservationPoints(ObsPoint,3) = YZdummy;
        ObservationPoints(ObsPoint,4) = YZdummy;
end
for ObsPoint = NoOfObsPoints/2+1:NoOfObsPoints
        ObservationPoints(ObsPoint,1) = 5;
        ObservationPoints(ObsPoint,2) = ObsPoint-NoOfObsPoints/2;
        ObservationPoints(ObsPoint,3) = YZdummy;
        ObservationPoints(ObsPoint,4) = YZdummy;
end
% Set an source in the centre of the problem space
Sources(1,2) = max_x/2;    % Source x-position
Sources(1,3) = YZdummy;    % Source y-position
Sources(1,4) = YZdummy;    % Source z-position
```

Die FDTD soll hier einen Raum von $150 \times 3 \times 3$ Zellen berechnen. In diesem Versuch soll die Wellenausbreitung entlang der 150 Zellen langen x-Achse dargestellt werden. Die Anzahl der Zellen in y- und z-Richtung kann aufgrund der vorliegenden Implementierung der FDTD nicht auf 1 minimiert werden. Der Wert *YZDummy* steht als Platzhalter für die y-z Lage der betrachteten x-Achse. Mittels des Parameters *D1_only* wird die FDTD angewiesen nur die E_z und H_y Komponenten zu berechnen. Um die Ausbreitung visualisieren zu können, werden die Werte der E_z und H_y Komponente entlang aller x-Koordinaten aufgezeichnet. Die entsprechenden Einstellungen sind in den Schleifenanweisungen des Listings (1) zu finden. Die Quelle wird an den linken Rand der x-Achse platziert.

Ergebnisse Die Simulationen mit einer Puls- und einer harmonischen Quelle führen zu den Bildern in Abbildung (5). Die Quelle befindet sich an der Stelle $x = 0$. Es ist gut zu erkennen, dass die Ausbreitungsrichtung senkrecht zu den Schwingungsrichtungen von E- und H-Feld stehen. Diese stehen untereinander wiederum senkrecht aufeinander. Um den Verlauf der Ausbreitung darzustellen, ist in den folgenden Abbildungen (6 und 7) die örtliche Verteilung der E_z-Komponente zu verschiedenen Zeitschritten zu sehen. Auf die Darstellung des Magnetfeldes wurde der Übersichtlichkeit halber verzichtet. Durch die Orthogonalität ist dessen Verlauf äqui-

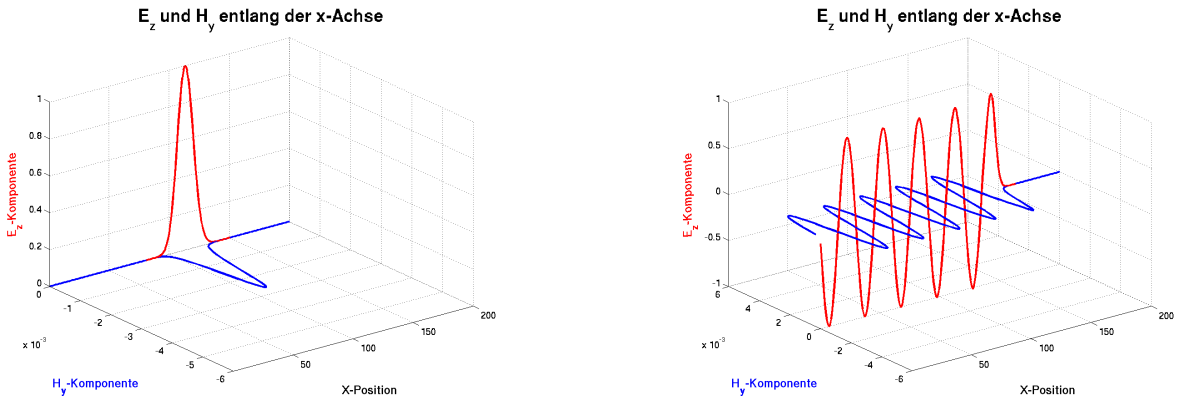

Abbildung 5: **E-Feld und H-Feld einer Flächenquelle (1D-FDTD):** Die Abbildungen zeigen die Verteilung des elektrischen und magnetischen Feldes entlang der x-Achse zu einem festen Zeitpunkt. Im linken Bild wurde eine Pulsquelle verwendet und im rechten eine harmonische Sinusquelle. Deutlich erkennbar ist die Orthogonalität der Schwingungsrichtungen von E- und H-Feld untereinander und bezüglich der Ausbreitungsrichtung.

valent. In den MATLAB® Scripten auf der beiliegenden CD-ROM ist der zeitliche Verlauf der Ausbreitung animiert dargestellt, um bei Bedarf eine bessere Veranschaulichung zu ermöglichen.

Um zu verdeutlichen, dass es sich bei der so simulierten Ausbreitung um eine ebene Welle handelt, wurde der Simulationsraum in die y-Richtung erweitert und entlang der y-Achse in jeder Zelle eine Quelle platziert. Der zeitliche Verlauf wurde für die gesamte x-y-Ebene aufgezeichnet und in einer Draufsicht geplottet. Die Ergebnisse sind in Abbildung (8) und (9) zu sehen. Man erkennt deutlich, dass die Intensität während der gesamten Ausbreitung konstant bleibt. Dies und die geradlinige Form der Ausbreitung sind Kennzeichen einer ebenen Welle.

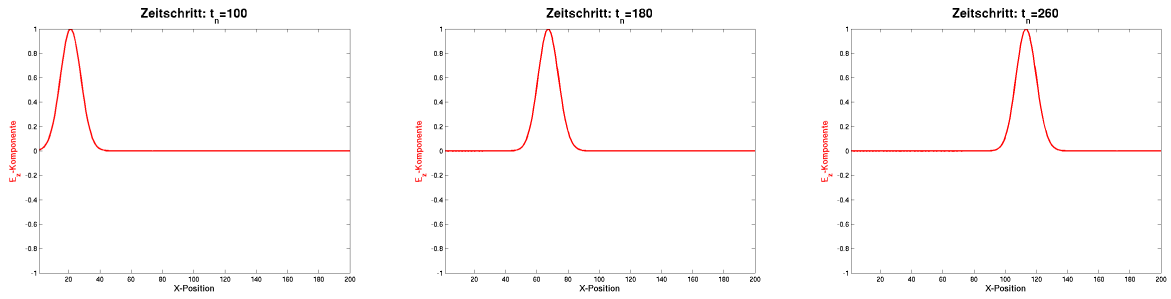

Abbildung 6: **Ausbreitung einer Flächen-Pulsquelle (1D-FDTD):** Örtliche Verteilung des E-Feldes einer Pulsquelle zu 3 Zeitpunkten.

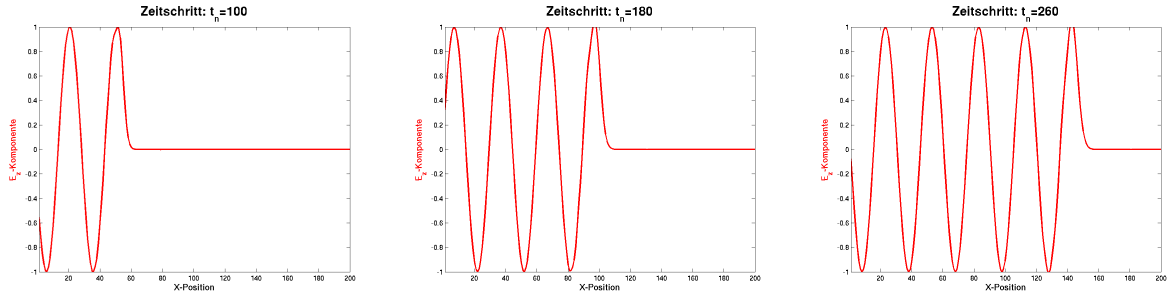

Abbildung 7: **Ausbreitung einer Flächen-Sinusquelle (1D-FDTD):** Örtliche Verteilung des E-Feldes einer Sinusquelle zu 3 Zeitpunkten.

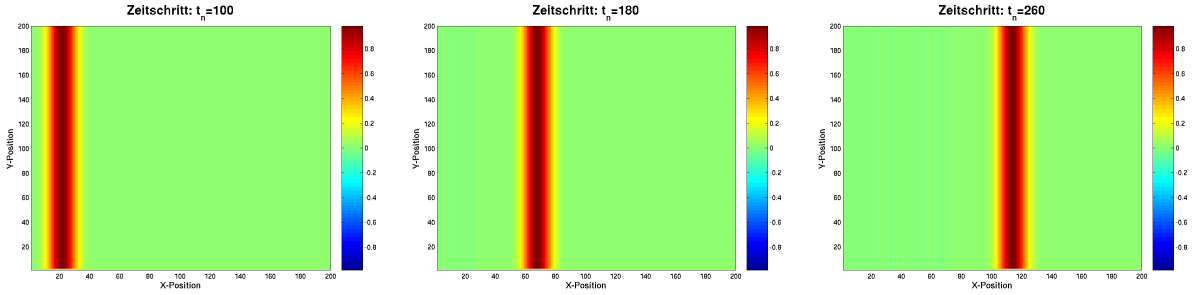

Abbildung 8: **Ebene Welle einer Pulsquelle (1D-FDTD):** Die Ausschnitte in diesen Abbildungen zeigen, wie sich ausgehend von einer Puls-Flächenquelle eine ebene Welle ausbreitet. Die unterschiedlichen Farben spiegeln die Amplitude wieder. Um dieses Bild zu erstellen, wurden entlang der y-Achse Punktquellen platziert. Die Ausbreitung wurde mittels der eindimensionalen FDTD berechnet, was zu den idealen Verlauf der ebenen Welle führt.

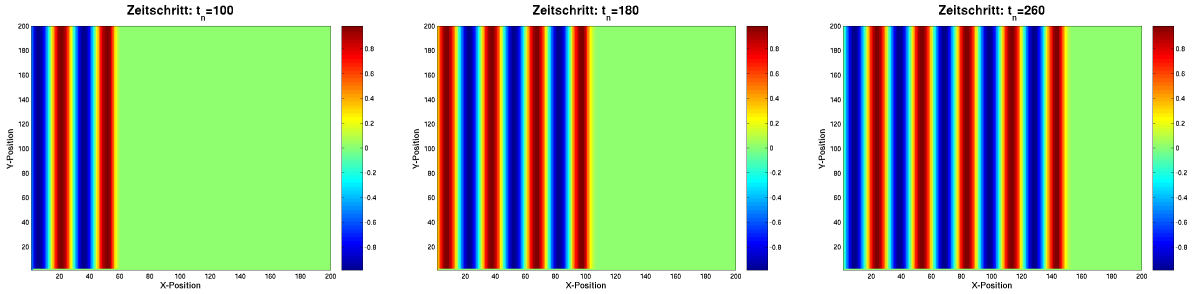

Abbildung 9: **Ebene Welle einer Sinus-Quelle (1D-FDTD):** Auch hier erfolgte die Berechnung mittels der eindimensionalen FDTD. Als Quellen wurden Sinus-Punktquellen entlang der y-Achse platziert. Man erkennt die Periodizität der Amplitude.

4.1.2. 2D Ausbreitung von Zylinderwellen

Theorie Analog zu Kapitel (4.1.1) wird hier zunächst ermittelt, welche Feldkomponenten zur Realisierung einer zweidimensionalen Ausbreitung berechnet werden müssen. Die resultierende Ausbreitung entspricht einer Feldverteilung[16], die eine ideale, unendlich ausgedehnte Linienquelle erzeugen würde. Im Gegensatz zur Wellenform in Kapitel (4.1.1)[17] wird hier eine Zylinderwelle erwartet.

Die Annahmen für den Fall der zweidimensionalen Ausbreitungen ähneln denen für den eindimensionalen Fall. Es wird gefordert, dass das E-Feld nur in der x-y-Ebene schwingt. Im Gegensatz zum 1D-Fall ist die Ausbreitungsrichtung jedoch nicht auf die x-Richtung beschränkt. Es wird lediglich gefordert, dass sie keine z-Komponente aufweist. Aus den genannten Forderungen ergeben folgende Feldvektoren:

$$\begin{aligned} \vec{E} &= \begin{pmatrix} E_x \\ 0 \\ E_z \end{pmatrix} \\ \vec{H} &= \begin{pmatrix} H_x \\ H_y \\ H_z \end{pmatrix} \end{aligned} \tag{4.5}$$

Wenn die Ausbreitungsrichtung sowohl eine x-, als auch eine y-Komponente besitzen soll, darf weder das E-Feld noch das H-Feld nur eine einzige Komponente in diese beiden Richtungen haben. Wäre dies der Fall, so kann aufgrund der Orthogonalität der Ausbreitungsrichtung[18] diese keine Komponente in die entsprechende Richtung besitzen. Für die Feldkomponenten in Gln. (4.5) heißt dies, dass das E-Feld in jedem Fall eine z-Komponente haben muss.

Die Orthogonalität der Felder untereinander führt wiederum auf folgende Gleichung:

$$\begin{aligned} \vec{E} \cdot \vec{H} = E_x H_x + E_y H_y + E_z H_z &= 0 \\ E_x H_x + E_z H_z &= 0 \end{aligned} \tag{4.6}$$

[16] zeitlich und örtlich
[17] Ebene Welle
[18] Bezüglich der beiden Felder

Die einzige Möglichkeit oben genannte Forderungen zu erfüllen besteht darin, $E_x = 0$ und $H_z = 0$ zu setzen. Die Feldvektoren vereinfachen sich damit zu:

$$\begin{aligned} \vec{E} &= \begin{pmatrix} 0 \\ 0 \\ E_z \end{pmatrix} \\ \vec{H} &= \begin{pmatrix} H_x \\ H_y \\ 0 \end{pmatrix} \end{aligned} \tag{4.7}$$

Wenn in den Anfangsbedingungen das H-Feld so definiert worden wäre, dass es nur in der y-z-Ebene schwingen darf, wären die resultierenden Feldvektoren entsprechend vertauscht. Weitere Möglichkeiten der zweidimensionalen Ausbreitung ergeben sich durch Rotation des Koordinatensystems und der entsprechenden Anpassung der Komponentenindizes.

Simulation Durch das entsprechende 2D Flag kann die FORTRAN FDTD Implementation dazu gebracht werden, nur die genannten E_z-, H_x- und H_y-Komponenten zu berechnen. Daraus resultiert die zweidimensionale Ausbreitung des Feldes der Quelle(n) in der x-y-Ebene in Form einer Zylinderwelle. Im Gegensatz zu Listing (1) muss die Ausdehnung in y-Richtung erhöht werden, da es sonst direkt zu störenden Reflexionen an den Grenzen kommt.
In Listing (2) sorgen die Einstellungen von *NoOfObsPoints* und *ObservationPoints* dafür, dass das FDTD-Programm die gesamte x-y-Ebene[19] aufzeichnet und an MATLAB® zurückgibt.
Die Quelle wurde diesmal in der Mitte des Problemraumes platziert, um die Ausbreitung der Zylinderwelle in alle Richtungen beobachten zu können.

[19] in Höhe der z-Koordinate $Zdummy = 2$

Listing 2: Matlab-Code - Versuch 2

```
Zdummy   = 2;
max_x    = 200;
max_y    = 200;
max_z    = 3;
D2_only  = 1;
% Set the whole x-y area to be observated
NoOfObsPoints       = 1;
ObservationPoints   = 1;
% Set an source in the centre of the problem space
Sources(1,2) = max_x/2;    % Source x-position
Sources(1,3) = max_y/2;    % Source y-position
Sources(1,4) = Zdummy;     % Source z-position
```

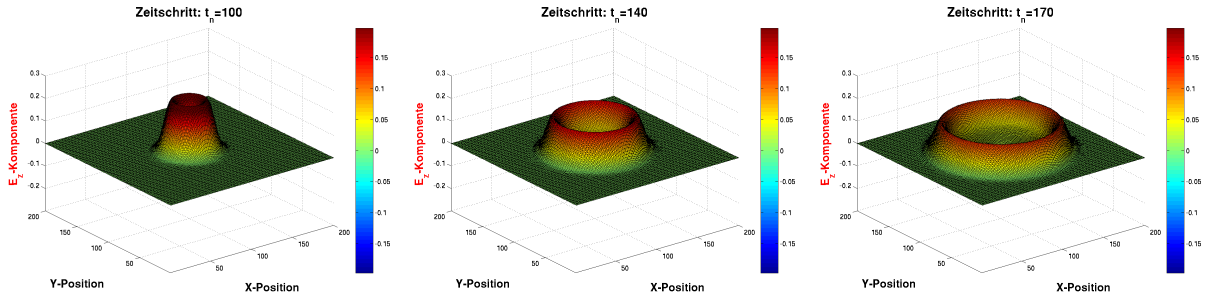

Abbildung 10: **Ausbreitung einer Linien-Pulsquelle (2D-FDTD):** Zweidimensionaler Plot einer sich ausbreitenden Zylinderwelle. Hervorgerufen wurde diese durch eine Punktquelle in der Mitte des Berechnungsraumes. Durch Verwendung der zweidimensionalen FDTD, hat diese Punktquelle die Wirkung einer endlos ausgedehnten Linienquelle in z-Richtung.

Ergebnisse Das Ergebnis der Simulation ist in den Abbildungen (10) und (11) dargestellt. Die Darstellungsweise in den Abbildungen (12) und (13) vermitteln einen besseren Eindruck von dem Charakter der zylindrischen Wellenausbreitung im Fall der zweidimensionalen FDTD.

In diesen Bildern ist neben der zylindrischen Ausbreitungsform auch eine andere Eigenschaft von Zylinderwellen zu erkennen. Im Gegensatz zur ebenen Welle nimmt die Intensität mit wachsender Entfernung von der Quelle ab. Hierbei gilt, dass die transportierte Energie einer Welle

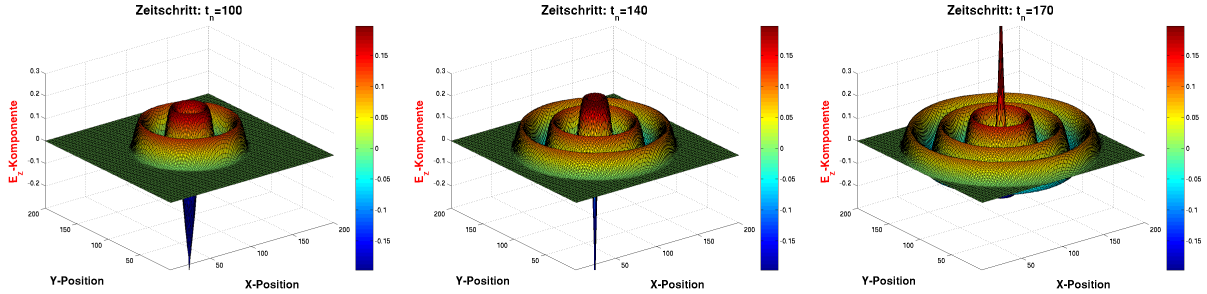

Abbildung 11: **Ausbreitung einer Linien-Sinusquelle (2D-FDTD):** Zweidimensionaler Plot sich ausbreitender Zylinderwellen. Ursache der Ausbreitung ist eine Sinusquelle.

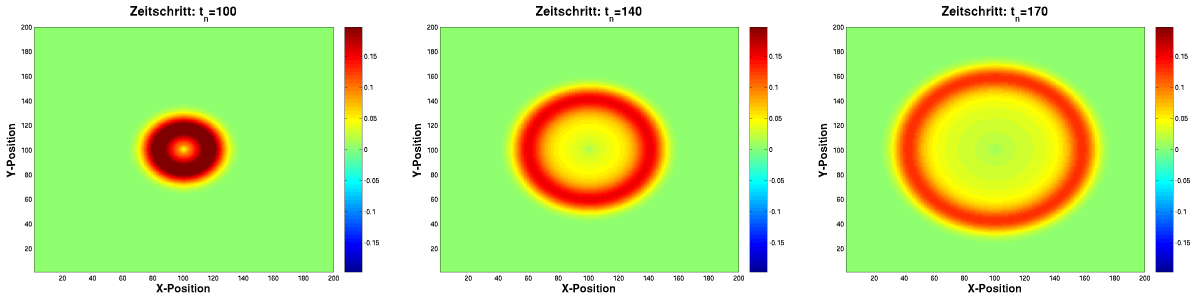

Abbildung 12: **Zylinderwelle einer Pulsquelle (2D-FDTD):** Dieses Bild zeigt die Ausbreitung der Zylinderwelle. Gut zu erkennen ist hier die Abnahme der Intensität mit zunehmender Entfernung zur Quelle.

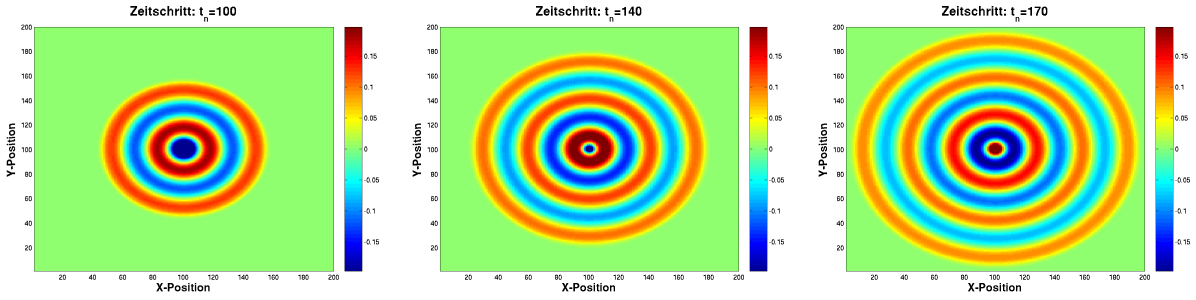

Abbildung 13: **Zylinderwelle einer Sinusquelle (2D-FDTD):** Die Sinusquelle im Mittelpunkt der Ebene hat die Ausbreitung entsprechender Zylinderwellen zur Folge. Auch hier ist die Amplitudenabnahme erkennbar.

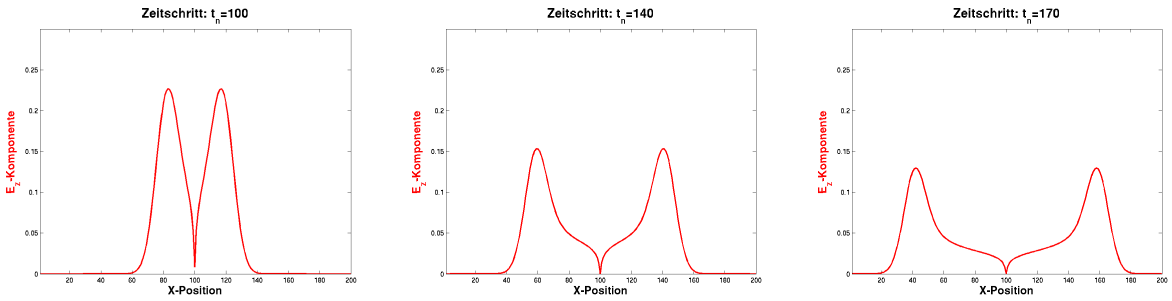

Abbildung 14: **Puls entlang der x-Achse (Zylinderwelle) (2D-FDTD):** Um die Abnahme der Amplitude zu untersuchen, wurde die Pulsamplitude entlang der x-Achse aufgezeichnet. Die Maximalwerte an jedem Punkt wurden extrahiert und in Abbildung (15) aufgetragen.

proportional zum Quadrat der Amplitude der Welle ist[20]. Im Umkehrschluss gilt also:

$$A \sim \sqrt{E} \tag{4.8}$$

Für die Energie gilt, dass sie sich mit zunehmenden Abstand von der Quelle auf immer größere Kreise verteilt. Die Amplitude in einem beliebigen Punkt auf einem solchen Kreis ist demnach indirekt proportional zur Wurzel des Kreisradius und damit zum Abstand zur Quelle. Am Beispiel der Amplitude der Feldkomponente E_z gilt:

$$E_z \sim \frac{1}{\sqrt{r}} \tag{4.9}$$

Um zu verifizieren, dass die FDTD Implementation auch diesen Aspekt abzubilden vermag, wurde die Amplitude eines Pulses über den Abstand von der Quelle, wie in den Abbildungen (14) gezeigt, ermittelt und in Abbildung (15) aufgetragen. Dort sind auch die Werte für zwei Entfernungen eingetragen. Da der zweite Wert die doppelte Entfernung zur Quelle hat, lässt sich der Zusammenhang einfach durch das Verhältnis der beiden Amplituden bestätigen. Allgemein gilt:

$$\frac{E_z(r_2)}{E_z(r_1)} = \frac{\sqrt{r_1}}{\sqrt{r_2}} \tag{4.10}$$

[20] http://www.glenbrook.k12.il.us/gbssci/phys/CLass/waves/u10l2c.html

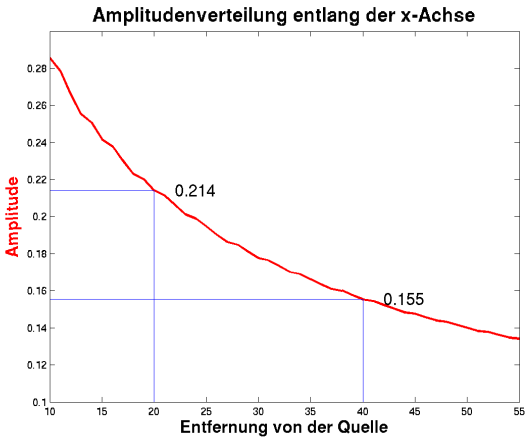

Abbildung 15: **Pulsamplitude einer Zylinderwelle entlang des Radius:** Um die Abnahme der Amplitude mit zunehmenden Abstand von der Quelle mit den theoretischen Werten vergleichen zu können, wurde diese an zwei Punkten gemessen. Der zweite Punkt lag dabei doppelt so weit von der Quelle entfernt, wie der erste. Das Verhältnis der beiden Messwerte stimmt mit den theoretischen Wert, wie er für Zylinderwellen erwartet wird, überein.

Da in o.g. Fall $r_2 = 2r_1$ gilt, vereinfacht sich Gln. (4.10) zu:

$$\frac{E_z(2r_1)}{E_z(r_1)} = \frac{\sqrt{r_1}}{\sqrt{2r_1}} = \frac{1}{\sqrt{2}} = 0.707 \tag{4.11}$$

Werden die in Abbildung (15) eingetragenen Werte ins Verhältnis zueinander gesetzt, ergibt das:

$$\frac{E_z(40)}{E_z(20)} = \frac{0.155}{0.214} = 0.724 \tag{4.12}$$

Dieser Wert entspricht in etwa den Erwartungswert von 0.707 aus Gln. (4.11).

4.1.3. 3D Ausbreitung

Theorie Soll die FDTD in ihrer kompletten Funktionalität zur dreidimensionalen Berechnung verwendet werden, so müssen alle Feldkomponenten betrachtet werden. Die Herleitung wurde im Kapitel (2.1) ausführlich dargestellt und resultierte in den Aktualisierungsgleichungen (2.20) und (2.22). Ausgehend von einer Punktquelle pflanzt sich eine Kugelwelle fort. Wie bei der Zylinderwelle verteilt sich die Energie der Quelle, hier allerdings auf eine Kugelfläche, die mit dem Abstand zur Quelle quadratisch wächst. Die Feldamplituden in jedem Punkt nehmen mit

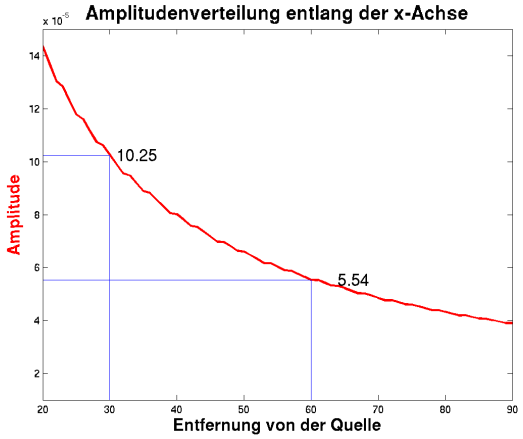

Abbildung 16: **Pulsamplitude einer Kugelwelle entlang des Radius (3D-FDTD):** Auch für die Abnahme der Amplituden sich ausbreitender Kugelwellen wurden die theoretischen Verhältnisse mit den simulierten Werten verglichen. Das Verhältnis der Amplituden zweier Messpunkte entspricht dem theoretischen Wert für Kugelwellen.

dem Faktor $\frac{1}{r}$ ab. Abbildung (16) zeigt die berechnete Pulsamplitude auf den Radius aufgetragen. Das Verhältnis der beiden angegebenen Werte entspricht mit

$$\frac{E_z(60)}{E_z(30)} = \frac{5.54}{10.25} \approx 0.51 \tag{4.13}$$

annähernd den theoretischen Wert von:

$$\frac{E_z(2r_1)}{E_z(r_1)} = \frac{1}{2} = 0.5 \tag{4.14}$$

In den weiteren Simulationen werden die Quellen entsprechend so platziert, damit wie in den vorherigen Betrachtungen annähernd ebene Wellen und Zylinderwellen entstehen. Die dargestellten Bilder lassen sich gut mit denen aus den vorherigen Versuchen vergleichen, um den Unterschied zwischen unendlich ausgedehnten Quellen[21] und solchen endlicher Ausdehnung zu erkennen. Gleichzeitig soll damit die Funktionstüchtigkeit der 3D-FDTD gezeigt werden.

Simulationen Um das FORTRAN Programm in den kompletten 3D-FDTD Modus zu versetzten, müssen zunächst die Kantenlängen auf einen vernünftigen, der Berechnung angepassten, Wert gesetzt werden. Für die Simulationen hier wurde ein kubischer Raum gewählt, dessen

[21] wie sie in Kapitel (4.1.1) und (4.1.2) durch die 1D/2D FDTD realisiert wurden

Grenzen weit genug auseinander liegen, um störende Reflexionen weitgehend zu vermeiden.[22] Die Flags zur Beschränkung der FDTD auf ein- und zweidimensionale Ausbreitung bleiben gelöscht. Für die beiden Versuche wurden die Quellen einmal als z-Linie in der Mitte der x-y-Ebene und einmal als y-z Fläche am linken Rand der x-Achse angeordnet.

Listing 3: Matlab-Code - Versuch 3

```
max_x    = 200;
max_y    = 200;
max_z    = 200;
D1_only  = 0;
D2_only = 0;
% Set the whole x-y area to be observated
NoOfObsPoints        = 1;
ObservationPoints    = 1;
% Set an line-source in the centre of the x-y-plane
for Source = 1:max_z
        Sources(Source,2)=max_x/2;    % Source x-position
        Sources(Source,3)=max_y/2;    % Source y-position
        Sources(Source,4)=Source;     % Source z-position
end
```

Ergebnisse Abbildung (17) und (18) zeigen die Ergebnisse mit der simulierten Linienquelle. Verglichen mit den Abbildungen (10) und (12) aus Kapitel (4.1.2) sind keine Unterschiede auszumachen. Dies ist der großzügigen Ausdehnung des Raumes zu verdanken. Würden in z-Richtung aber weniger Zellen Verwendung finden, so wäre nach einigen Zeitschritten sehr wohl eine Abweichung zum idealen Fall zu erkennen, da die Reflexionen schneller an den beobachteten Stellen zu Fehlern führen würden.

Die Abbildungen (20) und (21) für die Flächenquelle zeigen bereits Unterschiede im Detail, im Vergleich zu den Bildern (6) und (8) aus Kapitel (4.1.1). Die Abweichungen rühren zum Großteil von Reflexionen an den Rändern her. Diese überlagern sich mit dem eigentlichen Feld und pflanzen sich von den Rändern aus nach innen fort. Um diese Vermutung zu untermauern,

[22] Diesbezügliche Betrachtungen folgen im nächsten Abschnitt

wurde in einem zweiten Versuch die Mur'sche Randbedingung zweiter Ordnung aktiviert. Die Abbildungen (22), (23) und (24) zeigen, wie der Einfluss der Reflexionen abgenommen hat.

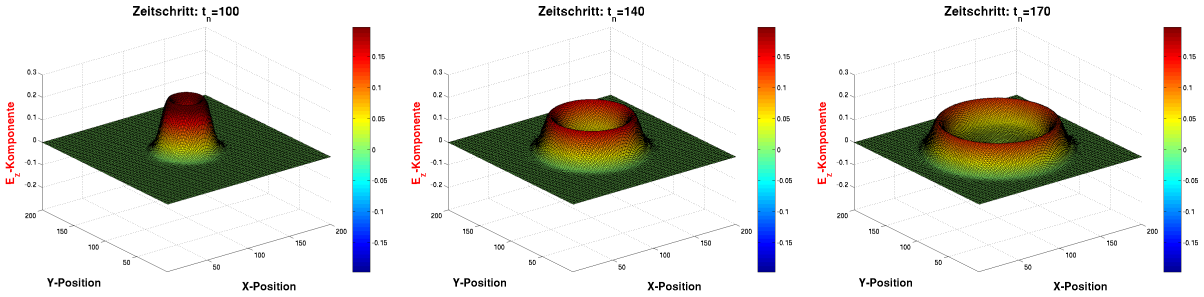

Abbildung 17: **3D Plot einer endlichen Puls-Linienquelle (3D-FDTD):** Zum Testen der dreidimensionalen FDTD wurden Pulsquellen entlang der z-Achse platziert um eine endlich ausgedehnte Linienquelle zu simulieren.

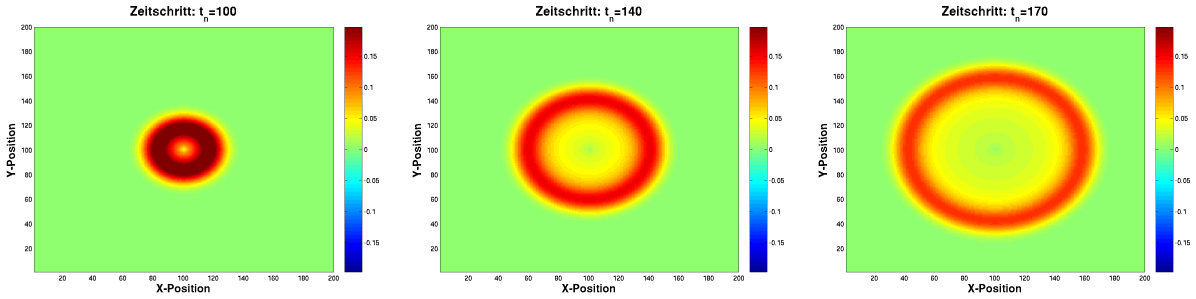

Abbildung 18: **Zylinderwelle einer endlichen Puls-Linienquelle (3D):** Auch diese Bilder der sich ausbreitenden Zylinderwelle verhalten sich wie erwartet.

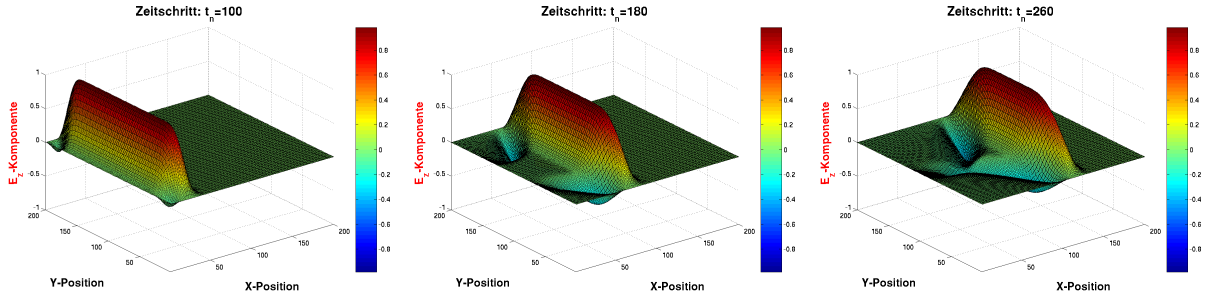

Abbildung 19: **3D Plot einer endlichen Puls-Flächenquelle 1'st Order Mur (3D-FDTD):** Für diese Bilder wurden in jedem Punkt entlang der linken y-z-Fläche Pulsquellen platziert. Diese Anordnung soll eine endlich ausgedehnte Flächenquelle simulieren. Wie an diesen Bildern zu sehen ist, gibt es Abweichungen zum erwarteten Aussehen. Diese sind auf Reflexionen an den Rändern zurückzuführen. Für diese Bilder wurde die Mur'sche ABC **erster Ordnung** verwendet.

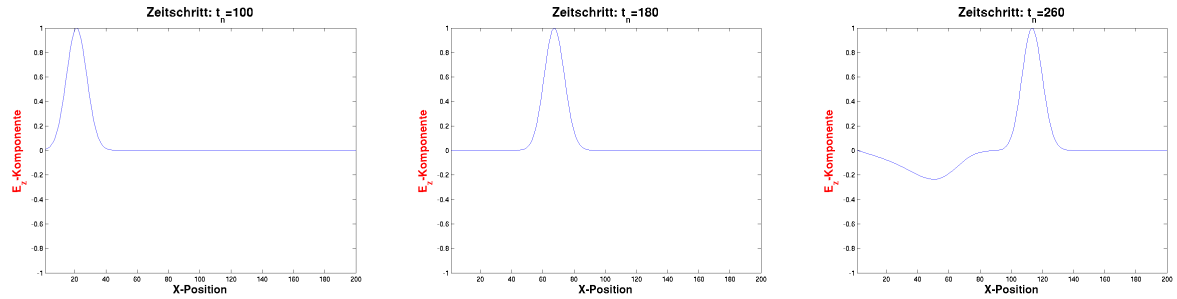

Abbildung 20: **1D Plot einer endlichen Puls-Flächenquelle 1'st Order Mur (3D-FDTD):** Der Schnitt entlang der x-Achse durch die Mitte der y-z-Ebene zeigt, dass sich die Reflexionen nach einer bestimmten Zeit bis in die Mitte ausgebreitet haben und auch hier als Fehler auftreten.

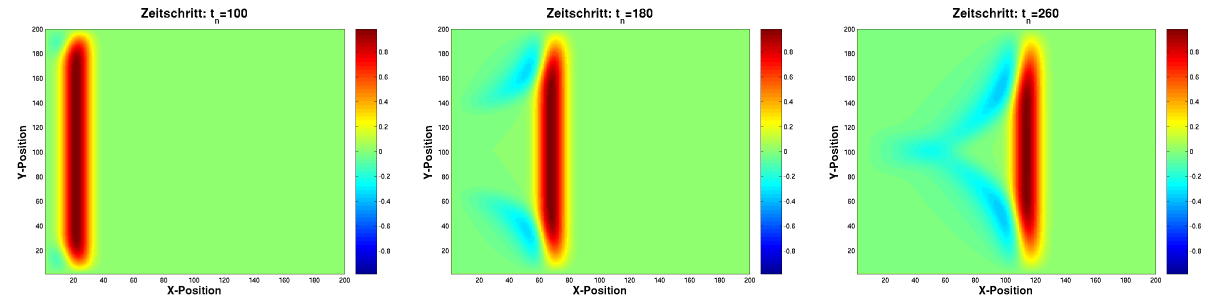

Abbildung 21: **Ebene Welle einer endlichen Puls-Flächenquelle 1'st Order Mur (3D-FDTD):** Hier ist gut zu erkennen, wie sich der Fehler vom Rand her ausbreitet und nach innen fortpflanzt.

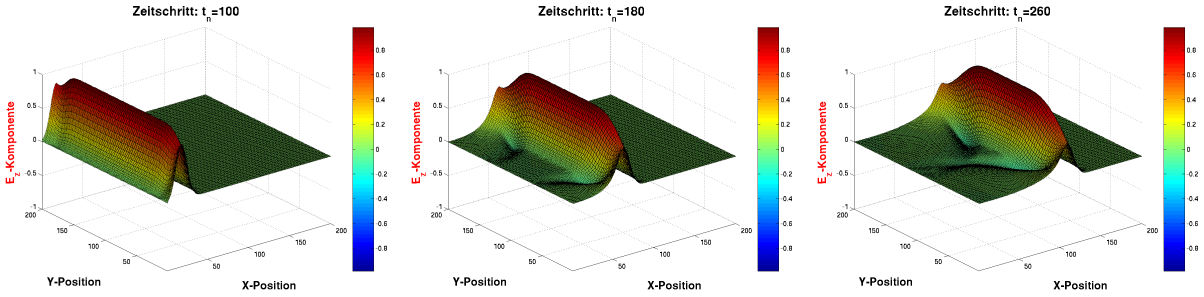

Abbildung 22: **3D Plot einer endlichen Puls-Flächenquelle - 2'nd Order Mur (3D-FDTD):** Anhand dieses Beispiels soll auch die endliche Absorptionsperformance der beiden absorbierenden Randbedingungen gezeigt werden. Hier kam die Mur'sche ABC **zweiter Ordnung** zum Einsatz.

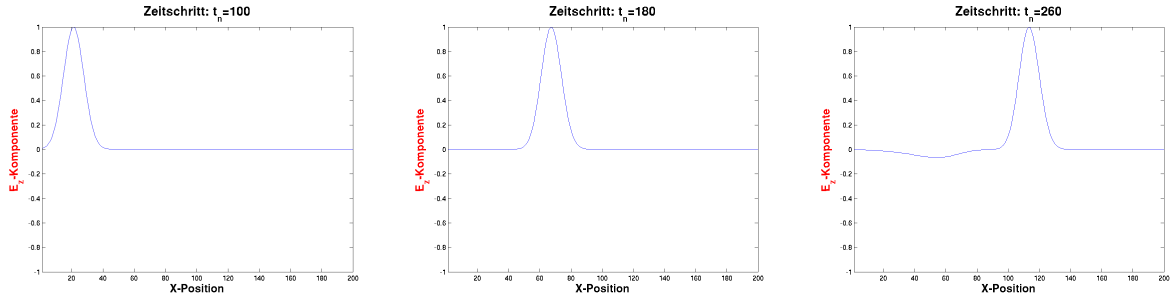

Abbildung 23: **1D Plot einer endlichen Puls-Flächenquelle - 2'nd Order Mur (3D-FDTD):** Im 1D Plot ist zu erkennen, dass die Amplitude des Reflexionsfehlers geringer ausfällt als bei Verwendung der Mur'schen ABC erster Ordnung.

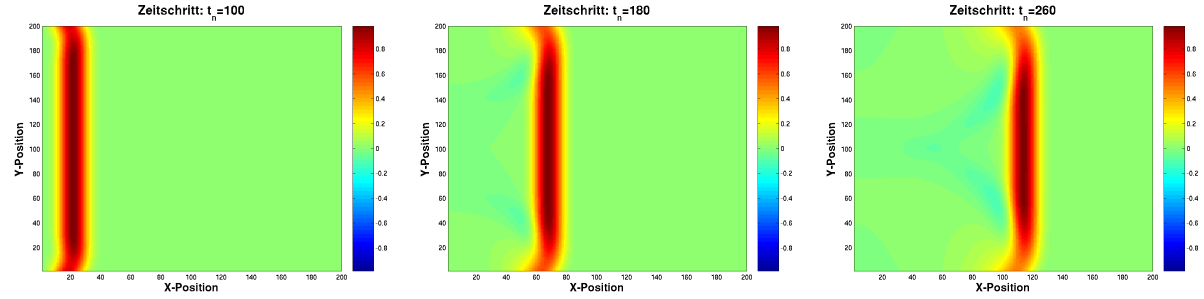

Abbildung 24: **Ebene Welle einer endlichen Puls-Flächenquelle - 2'nd Order Mur (3D-FDTD):** Dieses Bild entspricht dem einer ebenen Welle weit eher als bei Verwendung der Mur'schen ABC erster Ordnung. Auch die Fehler durch Reflexionen liegen im „grünen Bereich".

4.2. Weiterführende Tests bezüglich der FDTD

Nachdem die Wellenausbreitung von verschiedenen Quellen untersucht wurde, folgen nun einige Betrachtungen, die die FDTD erst wirklich nutzbar machen. Hierbei handelt es sich um die Beeinflussung des Feldes durch Streukörper innerhalb des Betrachtungsraumes und zum anderen um die Wirksamkeit absorbierender Randbedingungen, die es ermöglichen den Betrachtungsraum möglichst klein zu halten.

4.2.1. Verhalten an dielektrischen Grenzflächen

Theorie Trifft eine Welle auf ein Objekt, das andere dielektrische Eigenschaften als das momentane Medium aufweist, kommt es zu mehreren Effekten. Zum einen wird aufgrund der unterschiedlichen Wellenimpedanzen der beiden Medien ein Teil der Energie reflektiert. Das Verhältnis von reflektierter zu eintreffender Energie wird Reflexionsfaktor genannt. Der übrige Teil der Wellenenergie tritt in das zweite Medium ein und pflanzt sich dort weiter fort. Die Ausbreitungsgeschwindigkeit der Welle in den beiden Medien wird unterschiedlich sein, da die Phasengeschwindigkeit einer Welle von den dielektrischen Materialeigenschaften abhängig ist.
Im Folgenden werden die Gleichungen zur Bestimmung des Reflexionsfaktors, des Transmissionsfaktors und der Ausbreitungsgeschwindigkeit aufgezeigt. In dem Ergebnisteil werden diese dann mit den Simulationsergebnissen verglichen.
Der Feldwellenwiderstand Γ eines Mediums ist durch folgende Gleichung bestimmt:

$$\Gamma = \frac{1}{n}\sqrt{\frac{\mu_0}{\epsilon_0}} \tag{4.15}$$

Die Brechzahl n ist dabei wie folgt definiert und kann mit Hilfe der Bestimmungsgleichung für die Lichtgeschwindigkeit $c = \sqrt{\mu\epsilon}$ umgestellt werden.

$$\begin{aligned} n = \frac{c_0}{c} &= \frac{\sqrt{\mu\epsilon}}{\sqrt{\mu_0\epsilon_0}} \\ &= \sqrt{\mu_r\epsilon_r} \end{aligned} \tag{4.16}$$

In den in dieser Arbeit betrachteten Fällen entspricht die Permeabilitätszahl der des freien Raumes und fällt daher aus den Betrachtungen heraus. Daraus resultiert aus Gln. (4.15) und

Gln. (4.16):

$$\Gamma = \frac{1}{\sqrt{\epsilon_r}} \sqrt{\frac{\mu_0}{\epsilon_0}} \tag{4.17}$$

Weiterhin ist der Reflexionsfaktor R definiert als:

$$R = \frac{\Gamma_2 - \Gamma_1}{\Gamma_2 + \Gamma_1} \tag{4.18}$$

Für den Transmissionsfaktor T gilt:

$$T = \frac{2\Gamma_2}{\Gamma_2 + \Gamma_1} = 1 + R \tag{4.19}$$

Simulation In den Simulationen wurde zunächst die eindimensionale FDTD verwendet, um den Verlauf des Pulses mit reflektiertem und transmittiertem Anteil gut darstellen zu können. Auch wurden die entsprechenden Amplituden gekennzeichnet und quantitativ erfasst, um die Ergebnisse mit den theoretischen Werten auf Gln. (4.18) und Gln. (4.19) vergleichen zu können. Ein zweiter Versuch berechnet den Verlauf einer Zylinderwelle, die einen Streukörper passieren muss. Die dafür nötigen Einstellungen sind bereits aus den vorherigen Kapiteln (4.1.1) und (4.1.2) bekannt. Hinzugekommen ist lediglich die Definition des Streuobjekts (Siehe Listing 4).

Listing 4: Matlab-Code - Versuch 4

```
% für die 1D FDTD
for ScatX = 100:130
EMat(ScatX,YZdummy,YZdummy) = 30;
end

% bzw. für die 2D FDTD
for ScatX = 150:180
        for ScatY = 120:180
                EMat(ScatX,ScatY,Zdummy) = 30;
        end
end
```

Ergebnisse In Abbildung (25) ist der Verlauf einer Pulsquelle zu sehen. Die schwarzen Linien kennzeichnen die Grenzflächen vom Freiraum zum Streukörper und wieder in den Freiraum. Man erkennt die reflektierten und transmittierten Anteile. Ebenfalls in den Bildern eingetragen sind die Amplituden. Diese lassen sich ins Verhältnis zur einfallenden Amplitude setzen, um die Reflexionsfaktoren und Transmissionsfaktoren am Eingang und Ausgang des Streukörpers zu bestimmen.

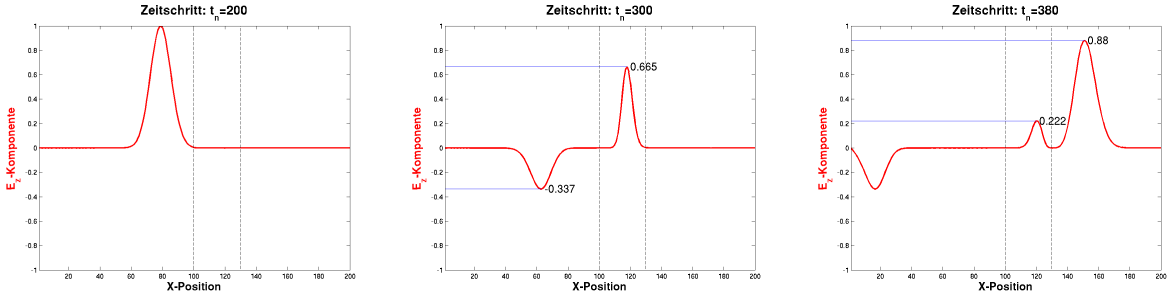

Abbildung 25: **Reflexion und Transmission am Streukörper:** Hier sind die Amplituden der transmittierten und reflektierten Welle eingetragen, um diese mit den theoretischen Werten vergleichen zu können.

$$\begin{aligned} R_{e,sim} &= \frac{-0.337}{1} = -0.337 \\ T_{e,sim} &= \frac{0.665}{1} = 0.665 \\ R_{a,sim} &= \frac{0.222}{0.665} = 0.334 \\ T_{a,sim} &= \frac{0.88}{0.665} = 1.323 \end{aligned} \tag{4.20}$$

Zum Vergleich folgt die Berechnung der Faktoren auf theoretischem Wege, wie am Anfang dieses Abschnittes hergeleitet. Laut Gln. (4.18) und Gln. (4.19) gilt:

$$\begin{aligned} R_{theor} &= \frac{\Gamma_2 - \Gamma_1}{\Gamma_2 + \Gamma_1} \\ T_{theor} &= \frac{2\Gamma_2}{\Gamma_2 + \Gamma_1} = 1 + R \end{aligned} \tag{4.21}$$

Für den Freiraum gilt $\epsilon_r = 1$ und damit erhält man den Wellenwiderstand für den freien Raum Γ_F:

$$\Gamma_F = \sqrt{\frac{\mu_0}{\epsilon_0}} = 120\pi\Omega \tag{4.22}$$

Der Wellenwiderstand des Streukörpers Γ_S beträgt für den gewählten Fall $\epsilon_r = 4$:

$$\Gamma_F = \frac{1}{\sqrt{4}}\sqrt{\frac{\mu_0}{\epsilon_0}} = 60\pi\Omega \tag{4.23}$$

Für die Eingangs- und Ausgangsseite ergeben sich damit:

$$\begin{aligned} R_{e,theor} &= \frac{60\pi\Omega - 120\pi\Omega}{60\pi\Omega + 120\pi\Omega} = -\frac{1}{3} = -0.33 \\ T_{e,theor} &= \frac{2 \cdot 60\pi\Omega}{60\pi\Omega + 120\pi\Omega} = \frac{2}{3} = 0.66 \\ R_{a,theor} &= \frac{120\pi\Omega - 60\pi\Omega}{60\pi\Omega + 120\pi\Omega} = \frac{1}{3} = 0.33 \\ T_{a,theor} &= \frac{2 \cdot 120\pi\Omega}{60\pi\Omega + 120\pi\Omega} = \frac{4}{3} = 1.33 \end{aligned} \tag{4.24}$$

Der Vergleich mit den simulierten Werten ergibt eine große Übereinstimmung, sodass der Implementation der FDTD auch in diesem Sinne Funktionstüchtigkeit bescheinigt werden kann.

Ein Blick auf die Abbildung (26) zeigt, dass der reflektierte Puls außerhalb des Streukörpers innerhalb der gleichen Zeit ungefähr den doppelten Weg zurückgelegt hat. Die Gln. (4.25) bestätigt, dass sich die Ausbreitungsgeschwindigkeit gegenüber der Lichtgeschwindigkeit im Vakuum für $\epsilon_r = 4$ halbiert.

$$c = \frac{1}{\sqrt{\epsilon_r}}\frac{1}{\sqrt{\mu_0\epsilon_0}} = \frac{1}{\sqrt{4}}\frac{1}{\sqrt{\mu_0\epsilon_0}} = \frac{1}{2}c_0 \tag{4.25}$$

Der Vollständigkeit halber zeigt Abbildung (27) den Verlauf des Pulses in der Umgebung eines Streukörpers in einem zweidimensionalen Plot. Der Einfluss des zusätzlich eingezeichneten Streukörpers ist gut zu erkennen.

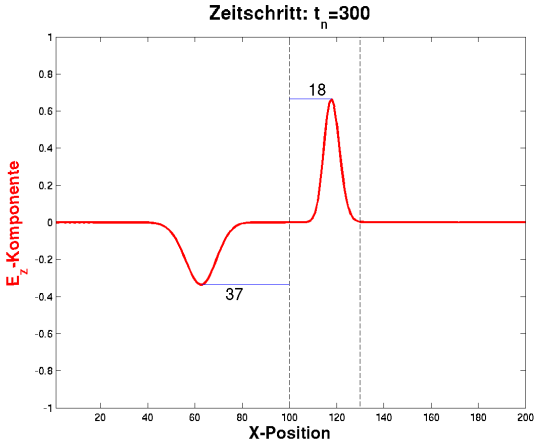

Abbildung 26: **Geschwindigkeit der reflektierten und transmittierten Welle:** Durch die zurückgelegten Wegstrecken der transmittierten und reflektierten Welle vom Ort der Reflexion lässt sich das Geschwindigkeitsverhältnis bestimmen und mit dem theoretischen Wert vergleichen.

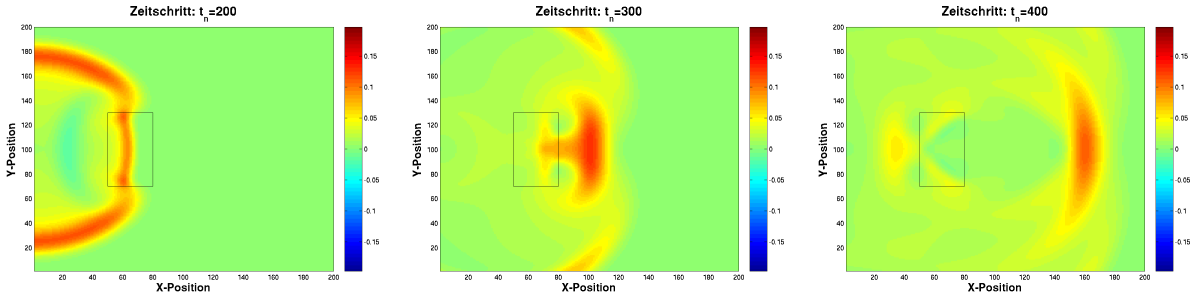

Abbildung 27: **Reflexion und Transmission am Streukörper:** Der Blick auf die x-y Ebene zeigt den Einfluss des Streukörpers (hier schwarz eingezeichnet) auf die Ausbreitung einer Zylinderwelle.

4.2.2. Absorbing Boundary Conditions

Hier folgen nun einige Versuche, um die Funktionstüchtigkeit und Wirkung der absorbierenden Randbedingungen nach Mur zu testen. Die dahinterstehenden theoretischen Grundlagen wurden in Kapitel (2.1.4) besprochen und dargelegt.

In Abbildung (28) ist die E_z und H_y Komponente vor und nach der Reflexion an den Raumgren-

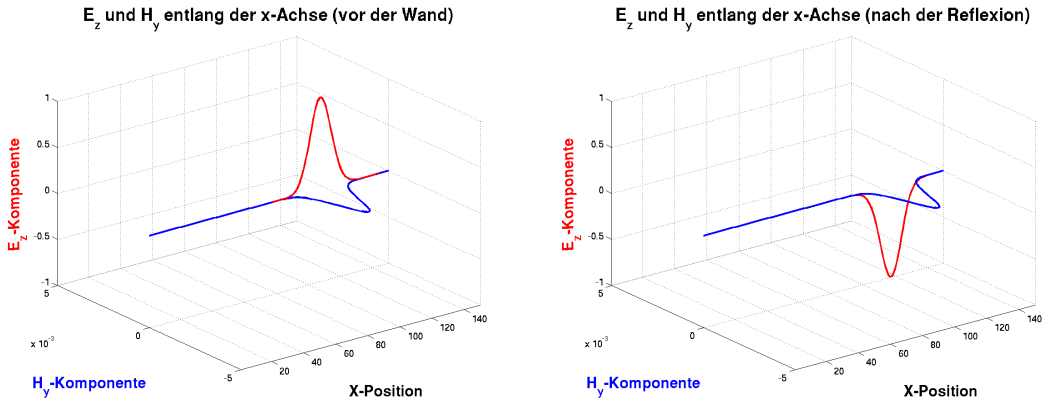

Abbildung 28: **Reflexion am Rand:** Beide Felder werden am Rand vollständig reflektiert.

zen dargestellt. Man erkennt, dass die gesamte Energie reflektiert wird. Diese Art die Grenzen zu verwenden ist annehmbar, falls die Felder in einem abgeschlossenen, perfekt elektrisch leitenden Körper simuliert werden sollen. In den meisten aller Fälle sollen die Felder so berechnet werden, als würden sie sich im unendlich ausgedehnten Raum befinden. Dazu ist es nötig, sogenannte absorbierende Randbedingungen in die FDTD-Implementierung einzuführen, die für eine möglichst große Dämpfung an den Rändern sorgen, um die ankommenden Wellen zu absorbieren. In Abbildung (29) wurde Mur's Randbedingung erster Art implementiert. Da die Welle senkrecht auf die Wand auftrifft, wird sie komplett absorbiert. Die Bilder in Abbildung (30) zeigen das Feld, nachdem dieses den Rand bereits erreicht hat. Das erste Bild zeigt das Feld ohne absorbierende Ränder. Man erkennt starke Reflexionen. Die beiden anderen Bilder zeigen das Feld, wenn die Randbedingungen nach Mur verwendet werden. Die Verteilungen kommen schon näher an das zu erwartende Feld (Keine Feldkomponenten mehr vorhanden) heran. Die Abbildung (31) zeigt den Fehler im Vergleich zum realen Verlauf[23]. Es ist erkennbar, dass der größte Fehler von den Ecken herrührt und sich von dort diagonal Richtung Zentrum ausbreitet.

[23]Dieser wurde mithilfe eines größeren FDTD-Raumes simuliert

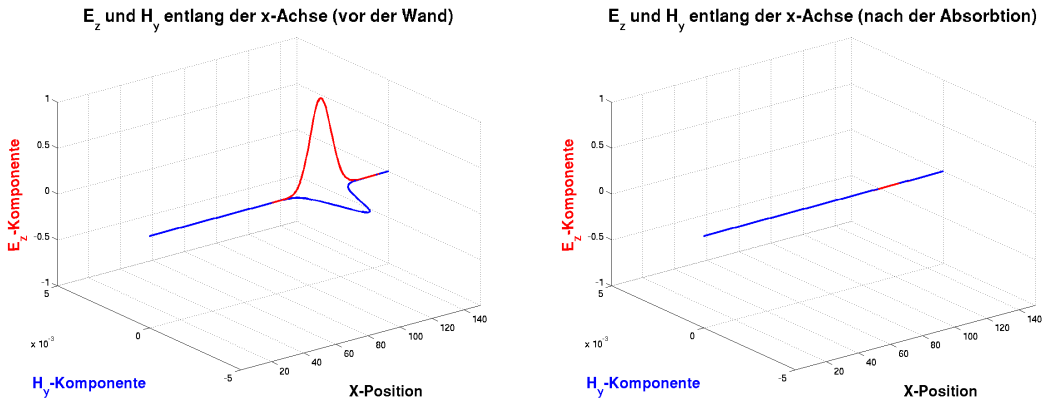

Abbildung 29: **Absorption am Rand:** Beide Felder werden am Rand vollständig absorbiert.

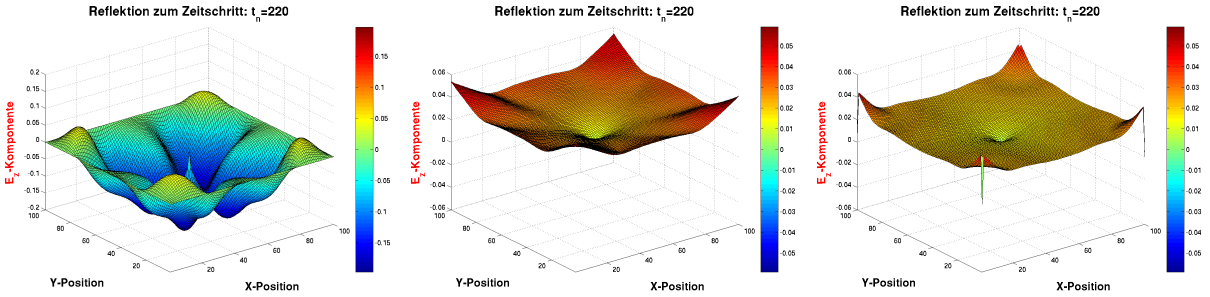

Abbildung 30: **Reflexionen am Rand:** Hier ist die Wirkung der absorbierenden Randbedingungen erkennenbar. Das Bild links entsteht, wenn keine ABC verwendet und ein Puls an den Rändern reflektiert wird. Die beiden anderen Bilder zeigen den gleichen Zeitpunkt bei Verwendung der 1'st und 2'nd Order Mur. Der Puls wird hierbei größtenteils absorbiert. Die größten Fehler treten an den Ecken auf, da die Wellen hier im spitzesten Winkel einfallen.

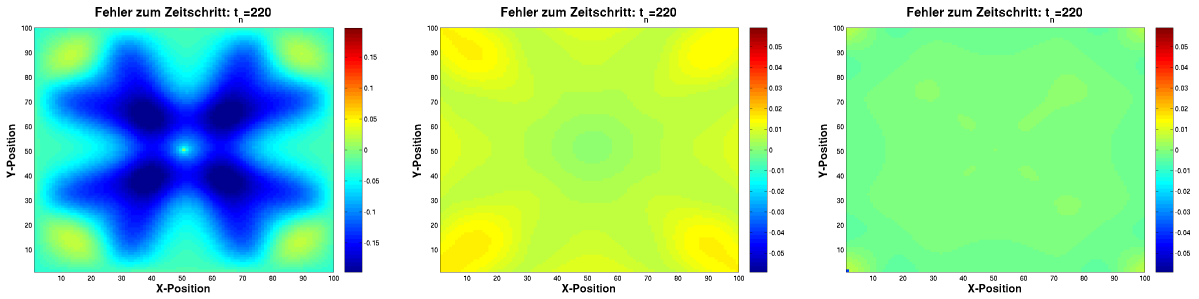

Abbildung 31: **Reflexionen am Rand:** Hier ist der gleiche Sachverhalt wie in Abbildung (30) dargestellt. Allerdings sieht man nicht die Feldamplituden, sondern die Differenz zu den idealen Werten(vollständige Absorption) und damit den Fehler. Das rechte Bild entspricht der Second Order Mur und zeigt die geringsten Reflexionsfehler

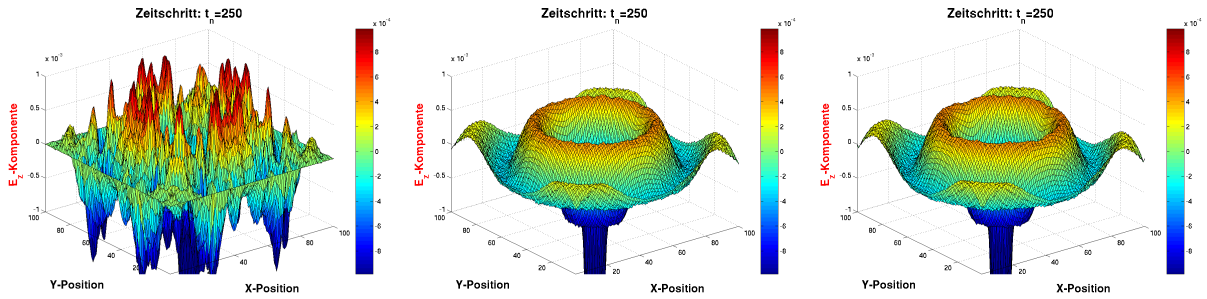

Abbildung 32: **Reflexionen am Rand:** Hier ist das Feld einer harmonische Sinusquelle dargestellt. Durch die vielen Reflexionen ist bei Vernachlässigung der Ränder das wahre Feld nicht mehr erkennbar. Bei Verwendung der absorbierenden Randbedingungen stimmt das resultierende Feld auch nach vielen Zeitschritten noch mit dem wahren Feld überein.

Um die Notwendigkeit des Einsatzes absorbierender Randbedingungen noch bildlicher aufzuzeigen, wurde für die Abbildung (32) eine Sinus-Linienquelle verwendet. Die Bilder zeigen die Felder nach etlichen Zeitschritten in einem eingeschwungenen Zustand. Im ersten der Bilder ist von den sinusförmigen Zylinderwellen nichts mehr zu erkennen da keine Absorption am Rand stattgefunden hat. Die Reflexionen überlagern das wahre Feld und führen zu einem unbefriedigenden Ergebnis. Anders stellt sich die Sache unter Verwendung der Mur'schen Randbedingungen dar. Hier ist die wirkliche Feldverteilung auch nach vielen Zeitschritten noch gut erkennbar.

4.3. Optimierung und Parameter-Rekonstruktion

In diesem Abschnitt werden die Versuche zur Rekonstruktion der Parameter und deren Ergebnisse vorgestellt. Um die prinzipielle Vorgehensweise der Rekonstruktion durch Optimierung darzustellen, wird zunächst ein Streukörper erstellt, der lediglich zwei verschiedene dielektrische Eigenschaften als Parameter besitzt. Der Ort und das Aussehen des Streukörpers sind bekannt, sodass die zu minimierende Kostenfunktion nur von zwei Parametern abhängig ist. Der Vorteil dieser Vorgehensweise besteht in der Möglichkeit der Visualisierung des Fehlergebirges.
Im zweiten Teil sind die dielektrischen Eigenschaften des Raumes frei verteilt und damit Ort und Form der Streukörper nicht bekannt. Das Verfahren muss nun versuchen eine große Anzahl von Parametern so zu optimieren, dass die Kostenfunktion möglichst klein wird, um das Bild des Streukörpers zu rekonstruieren. Aufgrund der begrenzten Rechenleistung wird in allen Versuchen auf die zweidimensionale FDTD zurückgegriffen.

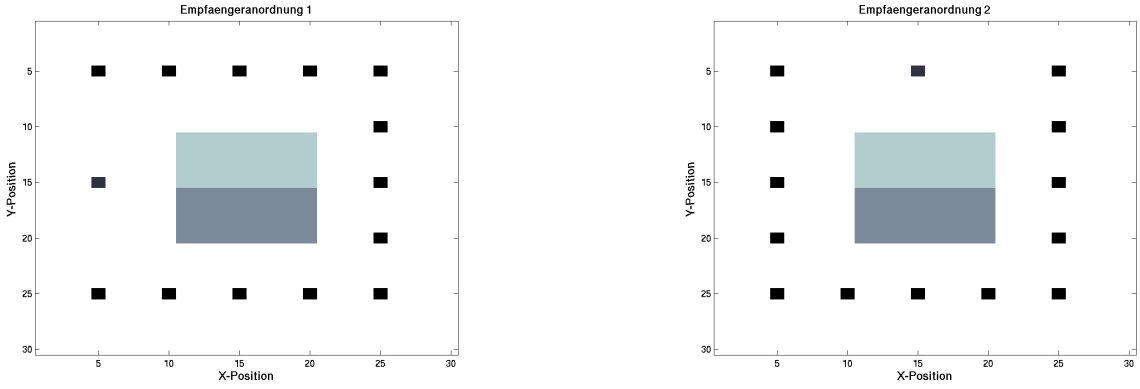

Abbildung 33: **Raumanordnung:** In der Mitte des Berechnungsraumes sind zwei Parameterzellen angeordnet. Um diese ist an einer Seite die Quelle zu sehen und an den anderen Seiten sind Empfänger in regelmäßigem Abstand platziert.

4.3.1. Kostenfunktion und Fehlerverteilung

Hier wird ein Streukörper in einem 2D Berechnungsraum platziert. Die Anordnung der Quelle und der Empfänger ist Abbildung (33) zu entnehmen. Für die beiden Anordnungen muss jeweils eine Vorwärtsrechnung durchgeführt werden. Der Streukörper besteht aus zwei Parameterzellen, die eine andere Form und Größe als die FDTD Zellen aufweisen können. Dieses Verfahren wird Dual-Mesh-Schema genannt und führt dazu, dass die Auflösung von FDTD- und Parameter-Mesh nicht übereinstimmen müssen. Das wiederum kann die Komplexität der Optimierungsaufgabe erheblich reduzieren, da mitunter weit weniger Parameter bestimmt werden müssen, als dies der Fall bei gleicher Auflösung wäre. In diesem Beispiel besteht eine Parameterzelle aus 10 FDTD-Zellen in x-Richtung und aus 5 FDTD-Zellen in y-Richtung. Obwohl hier die Parameterzellen eine ganzzahlig vielfache Größe der FDTD-Zellen aufweisen, so ist dies nicht zwingend nötig. Mit dreieckigen bzw. tetraedischen Parameterzellen lassen sich beispielsweise runde Streukörper weitaus besser modellieren, als dies mit quadratischen bzw. kubischen Zellen möglich ist.

In diesem Versuch wurden die Parameterzellen mit jeder möglichen Kombination an Permittivitätszahlen von $1.1 \leq \epsilon_r \leq 40$ in Zentelschritten beschrieben und die Felder an den Empfängern für jede der beiden Anordnungen aufgenommen und gespeichert. Mit diesen Daten ist es möglich für jede beliebige Parameterkombination die Fehlerverteilung zu bestimmen und darzustellen. Dazu muss lediglich definiert werden, wie die Parameterzellen in Wahrheit aussehen sollen. Da-

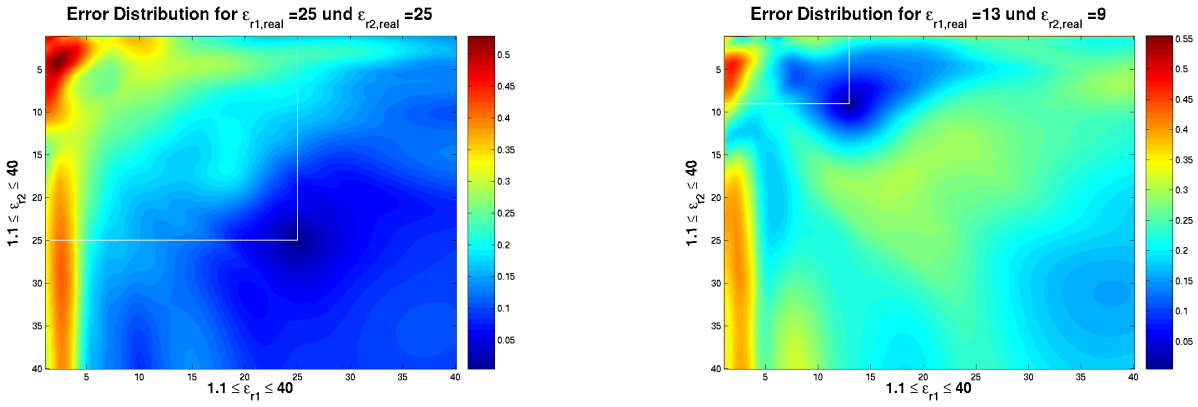

Abbildung 34: **Fehlerverteilung:** Hier ist die Fehlerverteilung für zwei verschiedene wahre Parameterkonfigurationen dargestellt. Man erkennt für jede andere Konfiguration den zugehörigen Fehler. Ebenfalls ist hier zu erkennen, dass die Verteilung mitunter mehrere lokale Minimalstellen hat, was dazu führt, dass die Optimierungsalgorithmen nicht für jeden Startwert das globale Minimum (und damit die wahre Parameterkonfiguration) finden können.

von ausgehend wird für jede andere Parameterkombination die Abweichung der Felder an den Empfängern von denen des wahren Zustandes berechnet und als Fehlerwert dargestellt. Somit ergibt sich ein 2-D Plot, der die Fehlerverteilung sichtbar macht. Diese ist stark abhängig von den gewählten wahren Parametern und für 2 Beispiele in Abbildung (34) dargestellt. Unter Umständen ergibt sich ein multimodales Gebiet mit vielen lokalen Minimalstellen. Dieser Umstand ist bei der Verwendung deterministischer Optimierungsverfahren zu berücksichtigen, da diese mitunter lediglich lokale Minimalstellen finden, deren zugehörige Parameter weit weg von der wahren Parameterverteilung liegen können.

4.3.2. Optimierung mit zwei Parametern

In diesem Versuch sollen die untersuchten Optimierungsverfahren von verschiedenen Startwerten aus die wahren Parameter rekonstruieren. Als wahre Parameterverteilung kamen diejenigen aus dem ersten Bild in Abbildung (34) zum Einsatz. Jeder Iterationsschritt wurde aufgezeichnet und in der Fehlerverteilung als Linie sichtbar gemacht. Damit lässt sich die Arbeitsweise der Optimierungsalgorithmen gut verfolgen. Die entsprechenden Bilder sind in den Abbildungen (35), (37) und (39) für verschiedene Startwerte dargestellt. Weiterhin wurden die Restfehler in jedem Schritt festgehalten und in den Abbildungen (36), (38) und (40) dargestellt. Bei diesen

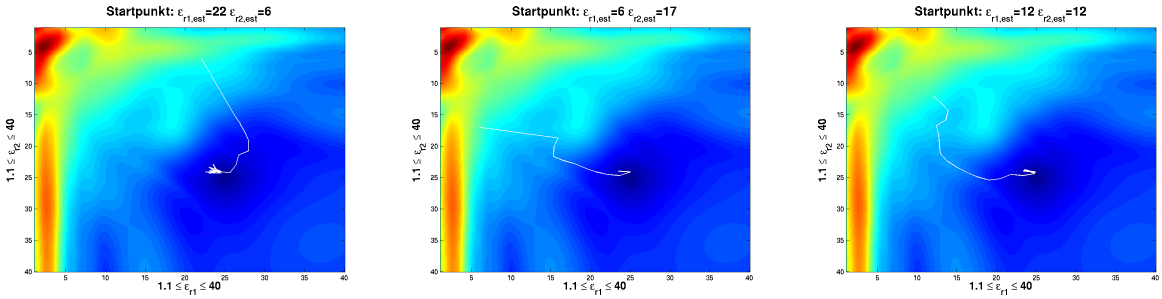

Abbildung 35: **Optimierung mittels SD-Verfahrens:** Hier ist der Weg dargestellt, den das SD-Verfahren während der einzelnen Iterationsschritte für 3 verschiedene Startpunkte abschreitet.

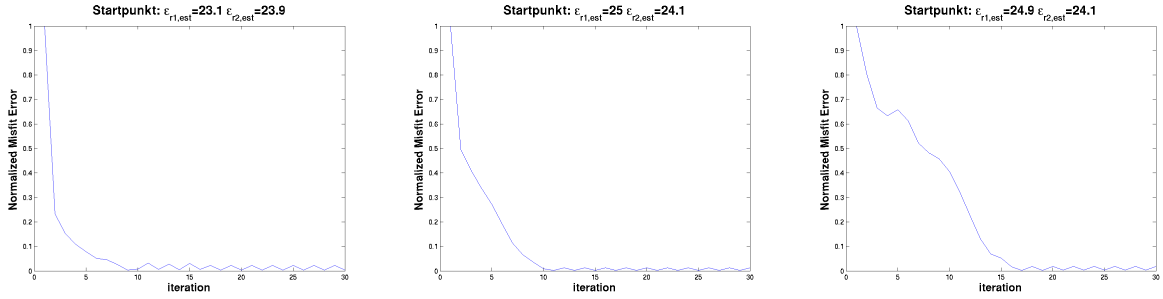

Abbildung 36: **NME des SD-Verfahrens:** Entlang der Iterationsschritte ist hier der relative Fehler aufgetragen.

Bildern handelt es sich um eine normierte Darstellung des aktuellen Fehlers auf denjenigen der Startwerte. Als Fehlerkennzahl wurde, wie in den anderen Versuchen auch, die euklidische Norm benutzt. Der normierte Fehler (NME) der aktuellen Iteration i ergibt sich aus der Formel [Fan04]:

$$NME = \frac{\|E_z^i - E_z^{meas}\|_2^2}{\|E_z^1 - E_z^{meas}\|_2^2} \tag{4.26}$$

In den Abbildungen (35) und (36) ist zu sehen, dass das Gradientenverfahren vor allem in der Nähe des Optimums (je nach Anpassung der kleinsten Schrittweite) zu Oszillationen neigt.
Um deutlich zu machen, dass die Verfahren nicht immer das globale Minimum und damit die wahren Parameter finden, zeigt Abbildung (41) zwei Beispiele, bei denen dies nicht gelingt. Das macht deutlich, dass die deterministischen Verfahren bei multimodalen Kostenfunktionen[24] stark von den gewählten Startparametern abhängig sind.

[24] Also solchen mit mehreren lokalen Minima

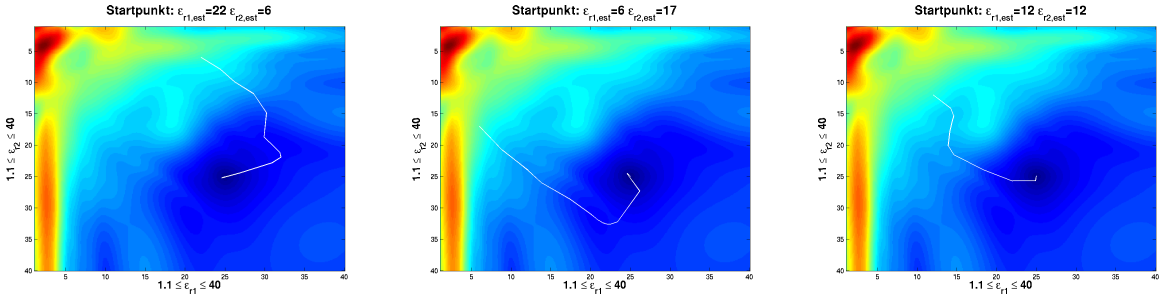

Abbildung 37: **Optimierung mittels GN-Verfahrens:** Hier ist der Weg dargestellt, den das GN-Verfahren während der einzelnen Iterationsschritte für 3 verschiedene Startpunkte abschreitet.

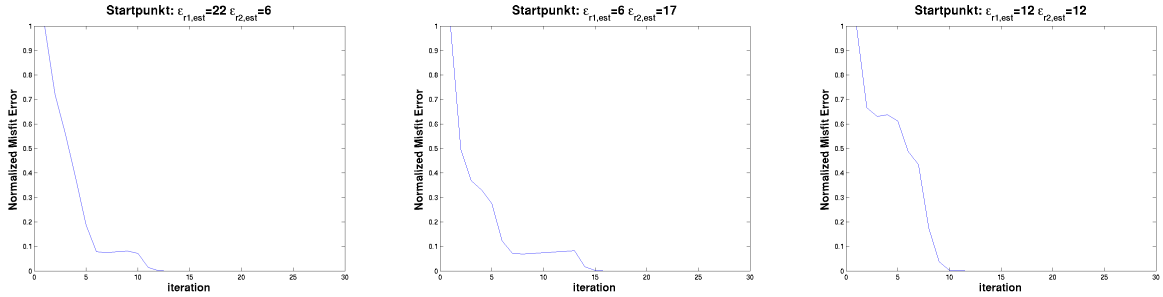

Abbildung 38: **NME des GN-Verfahrens:** Entlang der Iterationsschritte ist hier der relative Fehler aufgetragen.

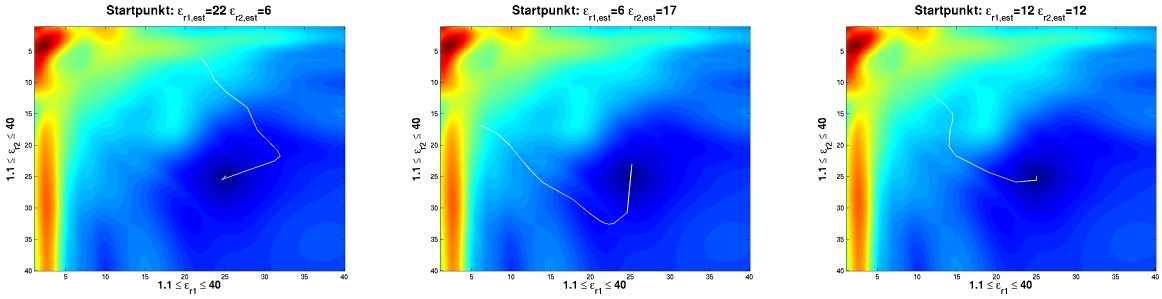

Abbildung 39: **Optimierung mittels LM-Verfahrens:** Hier ist der Weg dargestellt, den das LM-Verfahren während der einzelnen Iterationsschritte für 3 verschiedene Startpunkte abschreitet.

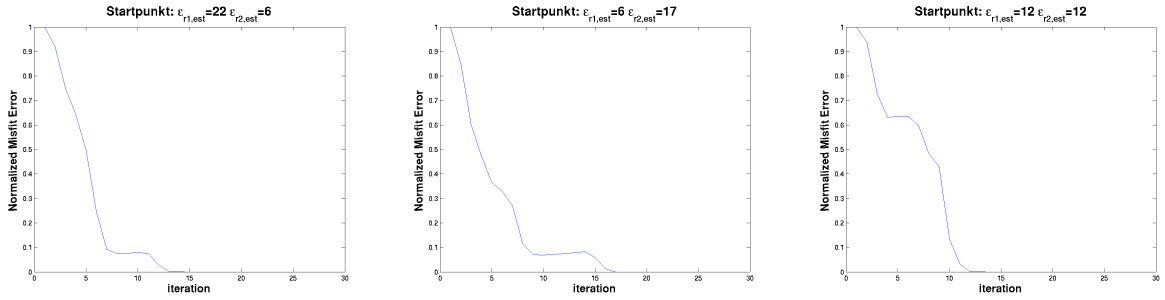

Abbildung 40: **NME des LM-Verfahrens:** Entlang der Iterationsschritte ist hier der relative Fehler aufgetragen.

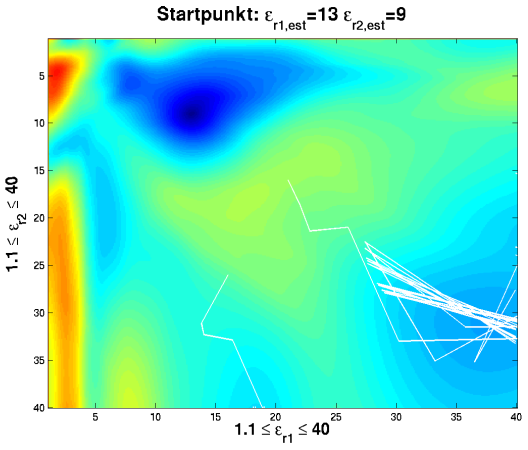

Abbildung 41: **Festfahren in lokalen Minima:** Hier sieht man, wie für manche Startpunkte nur lokale Minimalstellen gefunden werden.

4.3.3. Rekonstruktion der Parameterverteilung

In diesen abschließenden Versuchen soll die Parameterverteilung eines Streukörpers rekonstruiert werden, der aus mehreren Parameterzellen besteht. Diese wurden in der Mitte des Berechnungsraumes platziert. Der Körper umfasst 25 Parameterzellen, welche wiederum aus jeweils 4 FDTD Zellen zusammengesetzt sind. Die insgesamt 16 Empfänger sind um den Körper herum angeordnet. Jeder Empfänger dient einmal als Quelle. Die Anordnung des Raumes ist in Abbildung (42) dargestellt. Hierbei ist auch die Verteilung der Parameter erkennbar. Der Körper besteht aus einem Material mit der Permittivität $\epsilon_{r1} = 26$. Darin eingeschlossen ist ein Teil mit der Permittivität $\epsilon_r 2 = 28$. Ausgehend von der initialen, homogenen Parameterschätzung $\epsilon_r = 28$

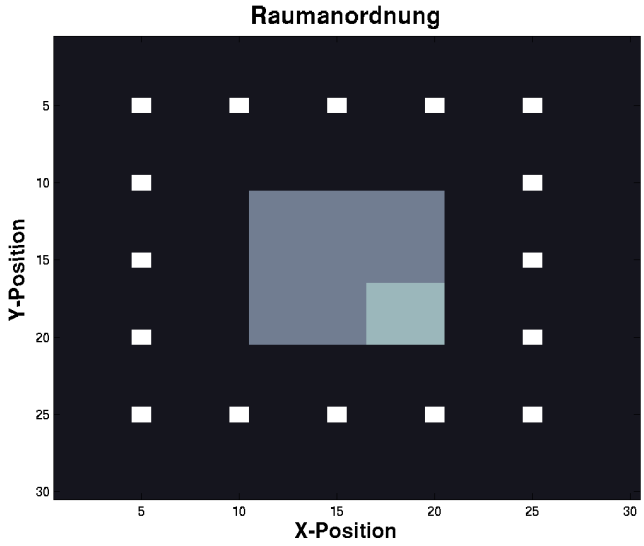

Abbildung 42: **Raumanordnung für Parameterrekonstruktion:** Zur Rekonstruktion der dargestellten Parameterverteilung werden 16 Empfänger benutzt. Jeder dieser Empfänger dient auch einmal als Quelle. Eine Parameterzelle besteht aus 4 FDTD Zellen und der gesamte Streukörper aus 25 solcher Parameterzellen.

wurden für jeden Optimierungsalgorithmus 20 Iterationen durchlaufen. Die rekonstruierte Parameterverteilung und der Verlauf der relativen Fehler sind in den Abbildungen (43) bis (45) dargestellt. Anhand des relativen Fehlers über den Iterationsschritten sind die Eigenschaften der einzelnen Algorithmen gut erkennbar. Während das Gradientenverfahren zu Oszillation neigt, rekonstruieren die anderen beiden Verfahren die wahre Parameterverteilung recht schnell und in diesem Fall genau. Das Gauß-Newton Verfahren ist etwas schneller, da es nicht gedämpft wird.

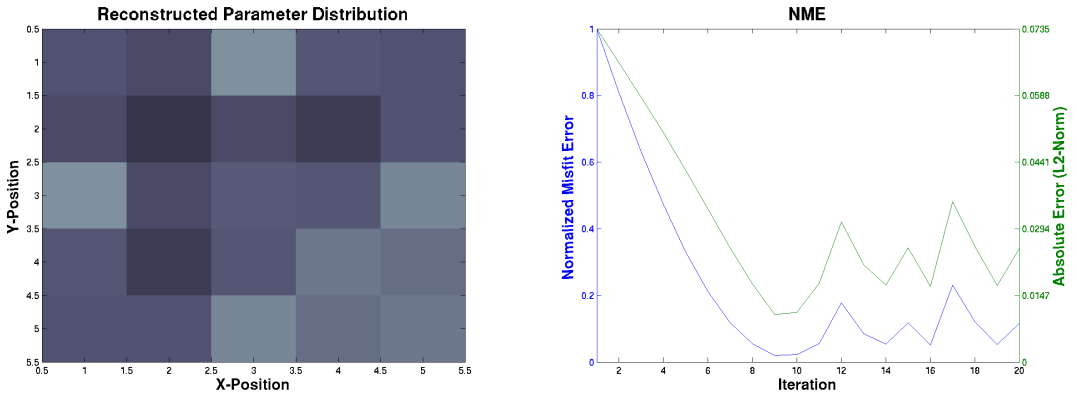

Abbildung 43: **Rekonstruktion und relativer Fehler (SD):** Das Ergebnis der Rekonstruktion durch das SD-Verfahren nach 20 Iterationsschritten und die relativen Fehler während der Iterationen.

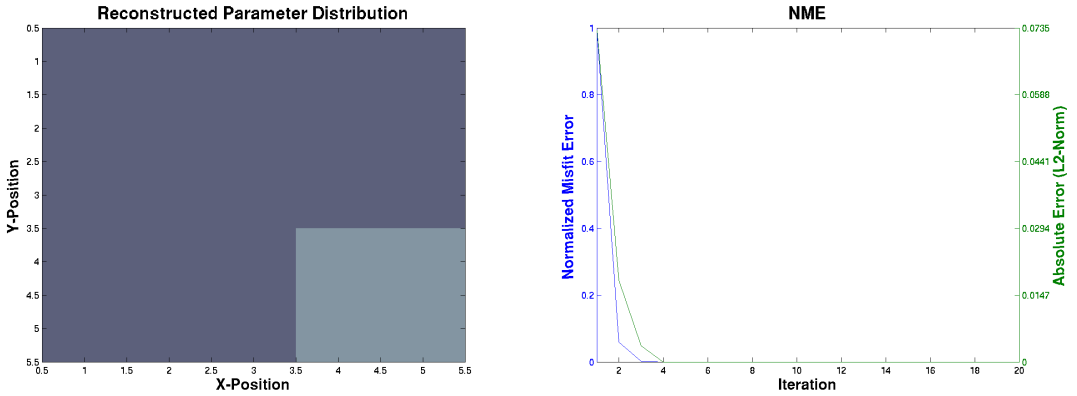

Abbildung 44: **Rekonstruktion und relativer Fehler (GN):** Das Ergebnis der Rekonstruktion durch das SD-Verfahren nach 20 Iterationsschritten und die relativen Fehler während der Iterationen.

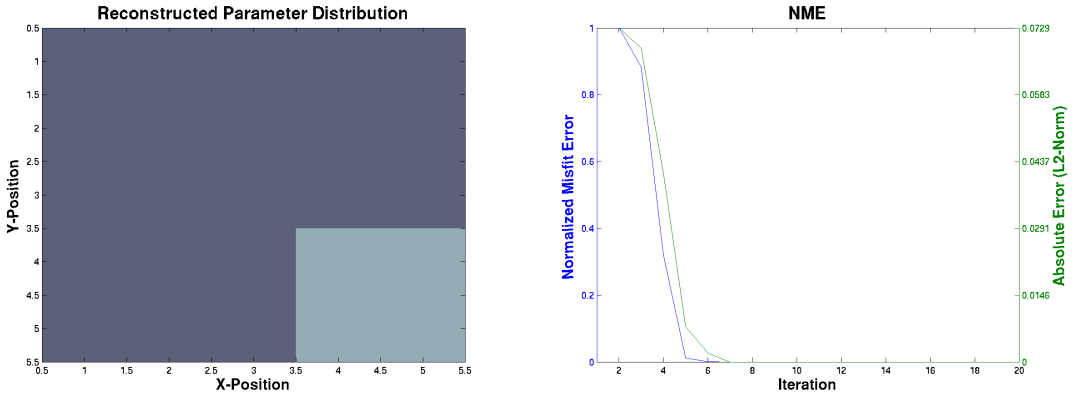

Abbildung 45: **Rekonstruktion und relativer Fehler (LM):** Das Ergebnis der Rekonstruktion durch das SD-Verfahren nach 20 Iterationsschritten und die relativen Fehler während der Iterationen.

Abbildung (46) zeigt eine weitere zu rekonstruierende Parameterverteilung. Als Rekonstruktionsalgorithmus kam das LM-Verfahren zum Einsatz. Der Streukörper besitzt mehrere Einschlüsse mit verschiedenen Permittivitäten von $\epsilon_r = 21$ bis $\epsilon_r = 31$. Die Parameterzellen entsprechen in diesem Versuch den FDTD Zellen. Der Streukörper besteht damit aus 100 zu rekonstruierenden Parameterzellen. Die Abbildungen (47) und (48) zeigen die Parameterverteilung zu verschiedenen Iterationsschritten. Das Grundmaterial und sämtliche Einschlüsse wurden nach 10 Iterationen erfolgreich rekonstruiert. Diese Versuche zeigen, dass die Parameterrekonstruktion mit den verwendeten Optimierungsverfahren prinzipiell möglich ist.

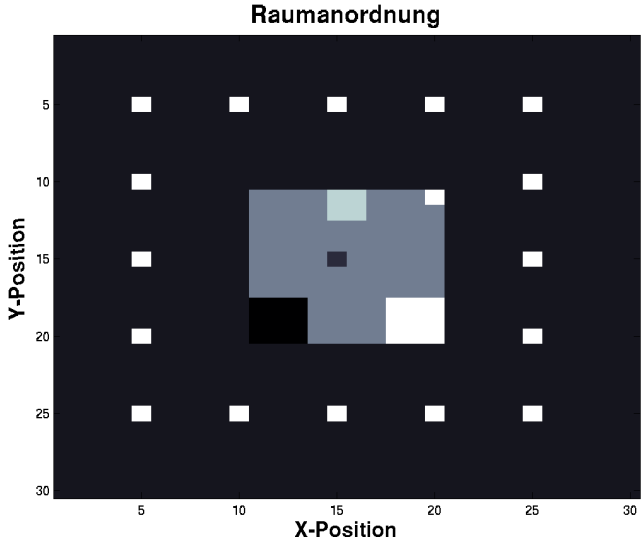

Abbildung 46: **Raumanordnung für Parameterrekonstruktion:** Zur Rekonstruktion der dargestellten Parameterverteilung werden 16 Empfänger benutzt. Jeder dieser Empfänger dient auch einmal als Quelle. Eine Parameterzelle besteht aus 1 FDTD Zelle und der gesamte Streukörper aus 100 solcher Parameterzellen.

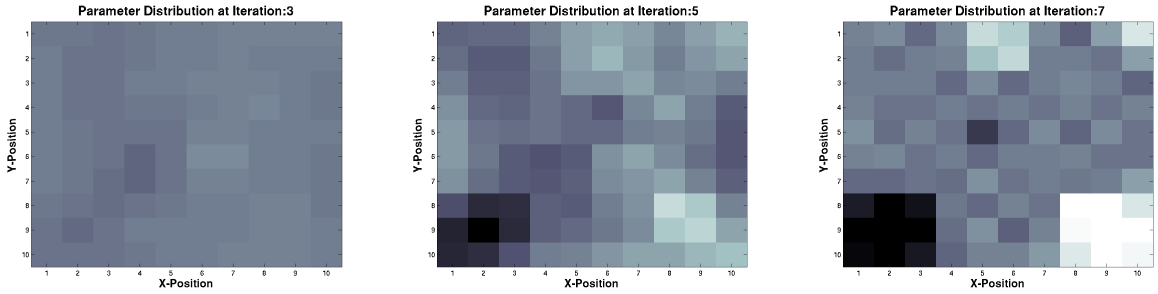

Abbildung 47: **Parameterverteilung bei Iterationsschritt 3, 5 und 7**

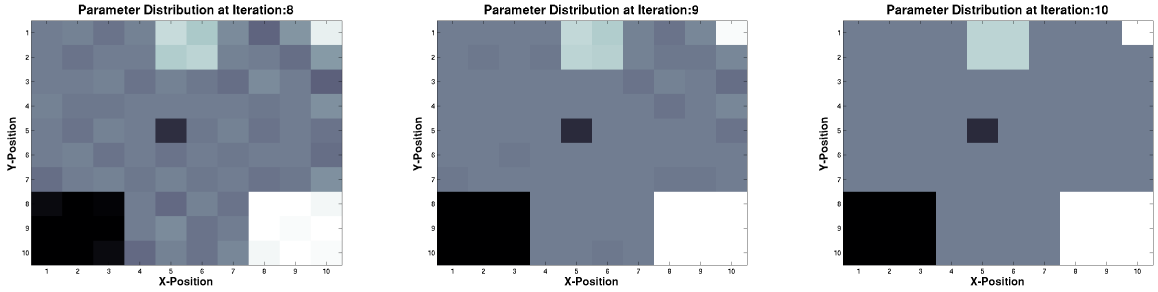

Abbildung 48: **Parameterverteilung bei Iterationsschritt 8, 9 und 10**

5. Zusammenfassung und Ausblick

Als Ergebnis der Versuche bleibt festzuhalten, dass die Lösung des inversen Streuproblems durch Anwendung deterministischer Optimierungsverfahren und der FDTD als Vorwärtsrechnung möglich ist. Im Hinblick auf das Ziel, die dielektrischen Parameter eines Kometenkerns zu rekonstruieren sind allerdings noch einige Verbesserungen vorzunehmen. Am wichtigsten hierbei ist die Beschleunigung der Berechnungen. Um dies zu erreichen muss der Berechnungsraum möglichst klein gehalten werden. Hierfür ist es wichtig, performante absorbierende Randbedingungen zu implementieren. Die Einbindung der PML bzw. GPML (Generalized Perfectly Matched Layer) ist einer der wichtigsten Schritte, die hierfür in Angriff genommen werden müssen. Eine weitere Aufgabe besteht in der Parallelisierung der FDTD, um die Möglichkeiten aktueller PCs und Großrechner auszunutzen.

Bezüglich der Optimierungsmethoden besteht ein interessanter Ansatz in der Verwendung stochastischer Algorithmen, die es auch bei multimodalen Problemen ermöglichen das globale Optimum zu finden.

Bezüglich der deterministischen Optimierungsverfahren sind weitere Untersuchungen und Verbesserungen möglich. In der vorliegenden Arbeit wurde lediglich ein zweidimensionales Problem gelöst. Desweiteren fand ausschließlich die Amplitude der Z-Komponente des elektrischen Feldes bei der Auswertung des Fehlers Verwendung. Durch Einbeziehen der anderen Feldkomponenten (6 im Falle der dreidimensionalen Ausbreitung einer Kugelwelle) und der Phasenlagen lassen sich noch mehr Informationen aus den Empfängern nutzbar machen.

Es soll nochmals angemerkt werden, dass der mögliche Einsatz dieser Verfahren sehr eng mit der künftigen Entwicklung der Mikrorechentechnik verknüpft ist. Wie in dieser Arbeit gezeigt wurde, stellt die FDTD für große Betrachtungsräume, wie sie für einen Kometenkern benötigt werden, enorme Anforderungen sowohl an CPU-Leistung, als auch an den Arbeitsspeicher.

6. Tutorials

6.1. Installation des Intel® Fortran Compiler und Integration in MATLAB®

Download Die aktuelle Version des *Intel® Fortran Compiler for Linux* kann direkt von der Intel® Homepage [25] bezogen werden. Der Button *Free Non-Commercial Download* führt nach akzeptieren der beiden Einschränkungen für freie, nichtkommerzielle Software zu einer Auswahlseite, auf der ein Link zur aktuellen Compilerversion führt. Nach Eingabe einer gültigen E-Mail Adresse erhält man eine Seriennummer und den Link zur Downloadseite[26]. Auf dieser muss sich für eine Architektur[27] entscheiden und das entsprechende Paket heruntergeladen werden. Nach einigen Minuten geht auf der angegebenen E-Mail Adresse eine Bestätigungsmail ein, in der auch eine Lizenzdatei mitgeliefert wird. Diese wird bei der Installation benötigt, falls während dieser keine Internetverbindung besteht.
Für Windows® bietet Intel® leider keine freie Version an. Allerdings besteht hier die Möglichkeit, eine zeitlich beschränkte Evaluierungsversion[28] zu benutzen.

Installation Zur Installation unter Linux wird das heruntergeladene *tar.gz* Archiv in ein temporäres Verzeichnis kopiert. Dort kann die Datei mittels

```
tar -xzvf dateiname.tar.gz
```

entpackt werden. Anschließend wird in das neu erstellte Verzeichnis gewechselt und die Installationsroutine mittels

```
./install.sh
```

aufgerufen. Im Hauptmenü werden verschiedene Optionen zur Auswahl angeboten. Hier findet sich auch eine Installationsanleitung falls Hilfe benötigt wird. Nach dem Start der Installation wird die Seriennummer abgefragt. Ist bereits eine ältere Version installiert, so kann dessen Lizenzdatei weiterverwendet werden. Falls das Programm keine Verbindung zum Intel® -Server aufbauen kann, besteht auch die Möglichkeit die in der Bestätigungsmail enthaltene Lizenzdatei

[25] http://www.intel.com/cd/software/products/asmo-na/eng/compilers/282048.htm
[26] Beides wird auch nochmals in einer Bestätigungsmail mitgeteilt
[27] Hier wurde *Product for IA-32* gewählt
[28] Erhältlich unter http://www.intel.com/cd/software/products/asmo-na/eng/219690.htm

zu verwenden.

Die Setuproutine untersucht nun das System. Wird dabei festgestellt, dass der verwendete Kernel oder die glibc-Bibliothek nicht unterstützt werden, hat man die Wahl das Setup abzubrechen und das System entsprechend zu aktualisieren[29] oder den Compiler dennoch zu installieren. Es folgt die Auswahl des Installationstyps. *Typical* installiert den Compiler und Debuger. Unter *Custom* kann gewählt werden, welche Komponenten installiert werden sollen. Nachdem die Lizenzbestimmungen gelesen und akzeptiert (*accept*) wurden, kann der Installationspfad gewählt werden. Standardmäßig ist */opt/intel/fc/versinsnummer/* ausgewählt. Eine abschließende Bestätigung startet den Kopiervorgang. Damit nicht bei jeder Benutzung des Compilers[30] der komplette Pfad angegeben werden muss, empfiehlt es sich dafür zu sorgen, dass der Ordner */opt/intel/fc/versionsnummer/bin* in der *PATH* Variable enthalten ist. Auch sollte sich der Ordner */opt/intel/fc/versionsnummer/lib* im Library Suchpfad *LD_LIBRARY_PATH* befinden. Die Konfiguration der Umgebungsvariablen hängt von der jeweiligen Distribution ab.

MATLAB® zur Benutzung des Intel® Compilers konfigurieren Damit MATLAB® den Intel® Compiler zur Erstellung der Mex-Files benutzt, muss das Programm entsprechend konfiguriert werden. Unter Linux funktioniert das durch Anpassung der Datei *mexopts.sh*, welche sich im *bin* Verzeichnis des MATLAB® Installationsordners befindet. In dieser Datei sind die Namen und Optionen der verschiedenen Compiler angegeben. Dabei ist für jede unterstützte Architektur (Sun, HP, x86, x86-64) ein Konfigurationsblock vorhanden. Im vorliegenden Fall handelt es sich um ein 32-Bit x86 System. Entsprechend sind die Einstellungen im Block *glnx86* vorzunehmen. Hier interessieren auch nur die Werte für den Fortran-Compiler. Diese wurden für die vorliegende Arbeit folgendermaßen angepasst:

```
FC='/opt/intel/fortran/bin/ifort'
FFLAGS='-fPIC -u -w95 -w'
FLIBS='$RPATH $MLIBS -L/opt/intel/fortran/lib -lm'
FOPTIMFLAGS='-03'
FDEBUGFLAGS='-g'
```

[29] Die Vorgehensweise unterscheidet sich je nach verwendeter Distribution
[30] Der Name des Compilers lautet *ifort*

Wurde der Intel® Compiler an einen anderen Ort installiert, so sind die Pfade entsprechend anzupassen.

Nach diesen Arbeiten ist es möglich, Fortranprogramme von MATLAB® aus mittels des Befehls *mex source.f90* zu kompilieren. Innerhalb des FORTRAN Programmes sind damit auch die verschiedenen MEX-Funktionen von MATLAB® zugänglich, was die Kommunikation von MATLAB® und FORTRAN-Programm erst effizient gestaltet.

6.2. Konfiguration der FDTD

Das FDTD-FORTRAN Programm wurde so entworfen, dass es sich weitgehend von MATLAB® aus konfigurieren und bedienen lässt. Da die entsprechende Schnittstelle bereits in Kapitel (3.2.2) beschrieben wurde, folgen an dieser Stelle lediglich einige Hinweise, die einen schnellen Einstieg in erste eigene Versuche ermöglichen sollen.

Für den Anfang ist es sinnvoll zunächst eine zweidimensionale Berechnung durchführen zu lassen. Zum einen sind die dabei entstehenden Felder leichter vorstellbar und die Ergebnisse der FDTD damit einfacher zu interpretieren, und zum anderen kann durch die Beschränkung der Raumausdehnung in z-Richtung sehr viel Rechenzeit und Arbeitsspeicher eingespart werden. Dadurch ist es auch möglich den Raum in die anderen beiden Richtungen weiter auszudehnen, falls dies nötig ist. Zu beachten ist hierbei, dass die z-Ausdehnung dennoch mindestens 3 Zellen umfassen muss, damit die internen Schleifen in ihrer derzeitigen Form keine ungültigen Arrayelemente indizieren und Speicherzugriffsfehler provozieren. Bei allen folgenden Einstellungen, die eine feste Koordinatenangabe verlangen, ist hier der Wert 2 für die z-Koordinate anzugeben. Weiterhin ist darauf zu achten, dass das Flag *D2_only* wirklich gesetzt ist. Wenn dies nicht geschehen ist, so führt die dreidimensionale Berechnung dazu, dass sofort Reflexionen an den nun sehr nahe liegenden Wänden auftreten, die jegliche sinnvolle Simulation verhindern.

Als nächstes ist es für den Anfang hilfreich, sich den Verlauf des Feldes auch der gesamten Ebene anzusehen. Dies geschieht dadurch, dass man die Anzahl der Beobachtungspunkte auf 1 setzt und auch das Feld mit den Beobachtungspunkten selber nur mit einer Zahl belegt. Im FORTRAN Programm wird die Dimension dieses Feldes abgefragt und damit festgestellt, ob einzelne Punkte konfiguriert sind, oder ob die gesamte x-y-Ebene aufgezeichnet werden soll. Im letzteren

Tabelle 2: FDTD-Beispieloptionen

Element	Beschreibung	Beispielwert
max_x	Anzahl der Zellen in x-Richtung	200
max_y	Anzahl der Zellen in y-Richtung	200
max_z	Anzahl der Zellen in z-Richtung	3
delta_x	Ausdehnung der Zellen in x-Richtung in m	1
delta_y	Ausdehnung der Zellen in y-Richtung in m	1
delta_z	Ausdehnung der Zellen in z-Richtung in m	1
time_steps	Anzahl der zu berechnenden Zeitschritte	170
frequency	Frequenz der Quelle, falls der Separate-Field-Ansatz verwendet wird (nicht verwendet)	0
amp	Amplitude der Quelle, falls der Separate-Field-Ansatz verwendet wird (nicht verwendet)	0
NoOfReceivers	Anzahl der definierten Empfänger	1
NoOfObsPoints	Anzahl der definierten Beobachtungspunkte	1
NoOfSources	Anzahl der definierten Quellen	1
ABCType	Art der absorbierenden Randbedingung	1
FullXYObservationZ	Die z-Koordinate, wenn eine gesamte x-y-Ebene beobachtet werden soll	2
ScatteredFieldOnly	Soll der Separate-Field-Ansatz verwendet werden? (nicht benutzt)	0
D2only	2D-FDTD (nur E_z, H_x undH_y werden berechnet)	1
D1only	1D-FDTD (nur E_z undH_y werden berechnet)	0

Fall wird in der momentanen Version dabei nur die Z-Komponente des E-Feldes beachtet. Wenn dies unerwünscht ist, kann durch Anpassung des Quellcodes aber leicht jedes beliebige Verhalten implementiert werden.

Wird nun noch eine einzelne Quelle[31] in der Mitte des Raumes platziert, so können erste Versuche mit der FDTD unternommen werden. Die Tabelle (3) zeigt die Einstellungen um die Ausbreitung einer Gauß'schen Pulsquelle zu beobachten.

Soll zum Beispiel ein Streukörper die freie Ausbreitung der entstehenden Zylinderwelle stören, so müssen die entsprechenden Zellen mit Werten für die Permittivität beschrieben werden. Listing (5) zeigt den MATLAB®-Code, um die obige Ausbreitung rechts der Quelle entsprechend zu beeinflussen und das Ergebnis animiert darzustellen.

[31] In der aktuellen Implementierung der 2D-FDTD ist nur die E_z-Komponente relevant

Tabelle 3: FDTD-Beispielparameter

Parameter	Beschreibung	Abmessungen	Beispielwerte
FDTD Parameter	Beinhaltet diverse Konfigurationswerte für die FDTD	1×17	siehe Tabelle (6.2)
epsilon_r	Array mit den möglichen Permittivitätszahlen	$1 \times N_M$	(1.11.2...39.940)
sigmaMAT	Array mit den möglichen Leitfähigkeiten	$1 \times N_M$	(00..00)
EMat	Array mit den dielektrischen Eigenschaften (Zeiger auf das entsprechende Element in den epsilon_r/sigmaMAT Arrays) der einzelnen Zellen	$max\_x \times max\_y \times max\_x$	alle 0 für freien Raum
ReceiverPoints	Array mit den Empfängern. Je eine Zeile pro Empfänger. Angabe der Koordinaten. Für diese Koordinaten werden die Werte der 10 letzten Zeitschritte für alle Feldkomponenten zurückgeliefert	$N_R \times 3$	(112)
ObservationPoints	Array mit den Beobachtungspunkten. Je eine Zeile pro Beobachtungspunkt. Angabe der Komponente und der Koordinaten. Für diese Koordinaten werden die Werte aller Zeitschritte für die gewählte Komponente zurückgeliefert.	$N_O \times 4$ oder 1×1	1
Sources	Array mit den Quellen. Je eine Zeile pro Quelle. Angabe des Quelltyps, der Koordinaten und der Amplituden der jeweiligen E-Komponente.	$N_S \times 7$	(11001002001)

Listing 5: Matlab-Code - Sample

```
% Set the standard FDTD Options here
max_x       = 200;      % X-Size in cells
max_y       = 200;      % Y-Size in cells
max_z       = 3;        % Z-Size in cells
delta_x     = 1;        % X-CellSize in m
delta_y     = 1;        % Y-CellSize in m
delta_z     = 1;        % Z-CellSize in m
time_steps  = 170;      % No. of TimeSteps
frequency   = 0;        % Just for Separate Field
amp         = 0;        % Just for Separate Field
ABC         = 1;        % Use first order Mur
FullXYObs_Z = 2;        % this x-y-plane will be observed
SFOnly  = 0;            % Full Field Mode
D2_only     = 1;        % Only E_Z, H_X and H_Y -> 2D
D1_only     = 0;        % Not 1D
% Set the whole x-y (at z=2)area to be observated
NoOfObsPoints       = 1;
ObservationPoints   = 1;
% Set an source in the centre of the problem space
NoOfSources  = 1;               % yust 1 source
Sources      = uint16(zeros(NoOfSources,7));
Sources(1,1) = 1;               % SourceType (1=pulse)
Sources(1,2) = max_x/2; % Source x-position
Sources(1,3) = max_y/2; % Source y-position
Sources(1,4) = 2;               % Source z-position
Sources(1,5) = 0;               % E_x-component amplitude
Sources(1,6) = 0;               % E_y-component amplitude
Sources(1,7) = 1;               % E_z-component amplitude
% Sets one Receiver (not required but have 2b defined)
NoOfReceivers=1;
ReceiverPoints=uint16(zeros(NoOfReceivers,3));
ReceiverPoints(1,1)=1;  % x-position
```

```
ReceiverPoints(1,2)=1;  % y-position
ReceiverPoints(1,3)=2;  % z-position
% Assemble the Parameter-Field
FDTDParams = [max_x max_y max_z delta_x delta_y delta_z
        time_steps frequency amp NoOfReceivers NoOfObsPoints
        NoOfSources ABC FullXYObs_Z ScatteredFieldOnly D2_only
        D1_only];
% Create the Cell-Field with the Scatterer
EMat        = uint16(zeros(max_x,max_y,max_z));
epsilon_r   = [1.1:0.1:40];      % 1.1 <= epsilon_r <= 40
sigmaMAT    = epsilon_r;
for i=1:length(sigmaMAT)
sigmaMAT(i)=0.0;          % not used here
end
for x=140:160
        for y=1:200
                EMat(x,y)=30  % the index in epsilon_r
        end
end
% Do the FDTD and animation for an puls source
[Field, delta_t, ObservationData]=fdtd3D(FDTDParams,
        epsilon_r, sigmaMAT, EMat, ReceiverPoints,
        ObservationPoints, Sources);
% Animate the propagation
x=[1:max_x];
y=[1:max_y];
for ts=1:time_steps
        surf(x,y,ObservationData(:,:,ts)')
        pause(0.05)
end
```

6.3. Erweitern des FDTD-Programms

Da die FDTD im Quellcode vorliegt und FORTRAN eine leicht zu erlernende Programmiersprache ist, kann die FDTD problemlos um weitere Features erweitert werden. Denkbar sind weitere und performantere absorbierende Randbedingungen oder die vollständige Implementierung und Verwendung des Separate-Field Ansatzes. Aber auch kleinere Änderungen wie zum Beispiel die 1D/2D-FDTD mit der Berechnung alternativer Feldkomponenten oder die Rückgabe weiterer Werte an MATLAB® sind einfach vorzunehmen. Zu beachten ist hierbei lediglich die Einhaltung der Spezifikation der Schnittstelle zwischen MATLAB® und FORTRAN. Aus diesem Grund folgen an dieser Stelle einige Ausführungen diesbezüglich, um den Einstieg in dieses Thema zu vereinfachen. Damit der Datenaustausch mit MATLAB® funktioniert, muss das FORTRAN Programm eine sogenannte Gateway-Routine implementieren. Diese besitzt die 4 Parameter *nlhs, plhs, nrhs* und *prhs*. Die Variablen mit *p* am Anfang stellen Pointer auf die zu übergebenden Daten dar. Hierbei handelt es sich um Arrays von Feldern. Über Pointer wird auf die Daten zugegriffen. *lhs* bzw. *rhs* stehen für *left hand side* und *right hand side*. Ersterer ist der Zeiger auf die an FORTRAN übergebenen Daten. Über den zweiten Pointer werden Daten an MATLAB® zurückgegeben. Um nun ein als Parameter übergebenes Array einzulesen, kann man sich der Mex-Funktionen bedienen. Über *mxGetM(prhs(i))* bzw. *mxGetN(prhs(i))* können die Dimensionen des Feldes ermittelt werden. *mxGetPr(prhs(i))* liefert nun einen Zeiger auf den Beginn des Feldes, welches mit *CALL mxCopyPtrToReal8(ptr, destinationArray, ElementCount)* oder *CALL mxCopyPtrToInt2(ptr, destinationArray, ElementCount)* in ein lokales Feld kopiert werden kann. Hierbei ist darauf zu achten, dass das lokale Feld vom gleichen Typ ist, wie das MATLAB® -Feld. Um Daten zurückzugeben, muss dies den Zeigerarray *plhs* mitgeteilt werden. Mittels *plhs(j) = mxCreateDoubleMatrix(M,N,0)* eine Matrix angelegt, welche die zurückzugebenden Daten aufnehmen soll. Nachdem man sich wiederum mittels *mxGetPr(plhs(j))* einen Zeiger auf die eben erstellte Matrix besorgt hat, kann das lokale Feld mit *CALL mxCopyReal8ToPtr(sourceArray, ptr, ElementCount)* kopiert werden. Die genaue Syntax und weitere Beispiele finden sich im Quellcode der FDTD. Weiterhin gibt es auf der Homepage[32] sowie in der MATLAB® -Hilfe weiterführende Informationen zu diesen Thema.

[32] http://www.mathworks.com/support/tech-notes/1600/1605.html

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[Abe04] Erik Abenius.
Time-Domain Inverse Electromagnetic Scattering using FDTD and Gradient-based Minimization.
Master's thesis, Royal Institute of Technology, 2004.

[Dec02] Thomas Deck.
Ultraschalltomographie.
Master's thesis, Berufsakademie Mannheim, September 2002.

[Els96] Frank Elsner.
Fortran 90 - die neuen Konzepte in Beispielen, November 1996.

[EN04] Elke and Michael Niedermair.
LATEX Praxisbuch.
Franzis Verlag, 2004.

[Fan04] Qianqian Fang.
Computational Methods for Microwave Medical Imaging.
PhD thesis, Dartmouth Collage, Dezember 2004.

[Fen] Xiaobing Feng.
Absorbing Boundary Conditions for electromagnetic wave propagation.

[Gon04] Prof. Dr.-Ing. Karl-Heinz Gonschorek.
Vorlesungsscript - Theoretische Elektrotechnik, 2004.

[Jan05] Britta Janssen.
Eine Entwicklungsumgebung für iterative Eigenwertverfahren in MATLAB, Dezember 2005.

[KL93] Karl S. Kunz and Raymond J. Luebbers.
Finite Difference Time Domain Method for Electromagnetics.
CRC Press, 1993.

[Mey05] Tobias Meyer.

Microwave Imaging of High-Contrast Objects.

PhD thesis, Otto-von-Guericke Universität Magdeburg, 2005.

[Ple05] Ana-Catalina Plesa.

Nichtlineare Optimierung mit dem Levenberg-Marquad-Algorithmus, April 2005.

[Qui01] Kevin Quinn.

The Newton Raphson Algorithm for Function Optimization.

October 2001.

[Sch00] Bernd Schandl.

A Package to Create a Nomenclature, 2000.

[Str02] Daniel D. Strauss.

Eine kurze Einführung in MATLAB, Oktober 2002.

[Tup97] Rodrigo Cordeiro Tupynamba.

Analyse unt Entwurf von passiven Hochfrequenzschaltungen mit Hilfe der FDTD-Methode.

PhD thesis, Technische Universität Hamburg-Harburg, 1997.

[Yee66] K. S. Yee.

Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media.

IEEE Transactions on Antennas and Propagation, pages 302–307, 1966.

[ZKGY05] G. Zheng, A. A. Kishk, A. W. Gilisson, and A.B. Yakovlev.

Implementations of Mur's Absorbing Boundaries with periodic structures to speed up the design process using Finite-Difference Time-Domain method.

Progress In Electromagnetics Research, pages 101–114, 2005.

A. Aktualisierungsgleichungen

A.1. FDTD Gleichungen - Allgemein

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_x^n(i,j,k) &+ \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_z^n(i,j,k) - H_z^n(i,j-1,k)\right)\\
&- \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_y^n(i,j,k) - H_y^n(i,j,k-1)\right)\\
E_y^{n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_y^n(i,j,k) &+ \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_x^n(i,j,k) - H_x^n(i,j,k-1)\right)\\
&- \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_z^n(i,j,k) - H_z^n(i-1,j,k)\right)\\
E_z^{n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_z^n(i,j,k) &+ \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_y^n(i,j,k) - H_y^n(i-1,j,k)\right)\\
&- \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_x^n(i,j,k) - H_x^n(i,j-1,k)\right)
\end{aligned}
\tag{A.1}
$$

$$
\begin{aligned}
H_x^{n+1}(i,j,k) = H_x^n(i,j,k) &+ \frac{\Delta t}{\mu}\frac{1}{\Delta z}\left(E_y^{n+1}(i,j,k+1) - E_y^{n+1}(i,j,k)\right)\\
&- \frac{\Delta t}{\mu}\frac{1}{\Delta y}\left(E_z^{n+1}(i,j+1,k) - E_z^{n+1}(i,j,k)\right)\\
H_y^{n+1}(i,j,k) = H_y^n(i,j,k) &+ \frac{\Delta t}{\mu}\frac{1}{\Delta x}\left(E_z^{n+1}(i+1,j,k) - E_z^{n+1}(i,j,k)\right)\\
&- \frac{\Delta t}{\mu}\frac{1}{\Delta z}\left(E_x^{n+1}(i,j,k+1) - E_x^{n+1}(i,j,k)\right)\\
H_z^{n+1}(i,j,k) = H_z^n(i,j,k) &+ \frac{\Delta t}{\mu}\frac{1}{\Delta y}\left(E_x^{n+1}(i,j+1,k) - E_x^{n+1}(i,j,k)\right)\\
&- \frac{\Delta t}{\mu}\frac{1}{\Delta x}\left(E_y^{n+1}(i+1,j,k) - E_y^{n+1}(i,j,k)\right)
\end{aligned}
\tag{A.2}
$$

A.2. FDTD Gleichungen - Separate Field

$$
\begin{aligned}
E_x^{s,n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_x^{s,n}(i,j,k) \quad &+ \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_z^{s,n}(i,j,k) - H_z^{s,n}(i,j-1,k)\right) \\
&- \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_y^{s,n}(i,j,k) - H_y^{s,n}(i,j,k-1)\right) \\
&- \quad \frac{\sigma\,\Delta t}{\epsilon+\sigma\,\Delta t}E_x^{i,n+1}(i,j,k) \\
&- \quad \frac{(\epsilon-\epsilon_0)\Delta t}{\epsilon+\sigma\,\Delta t}dE_x^{i,n+1}(i,j,k)
\end{aligned}
$$

$$
\begin{aligned}
E_y^{s,n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_y^{s,n}(i,j,k) \quad &+ \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta z}\left(H_x^{s,n}(i,j,k) - H_x^{s,n}(i,j,k-1)\right) \\
&- \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_z^{s,n}(i,j,k) - H_z^{s,n}(i-1,j,k)\right) \\
&- \quad \frac{\sigma\,\Delta t}{\epsilon+\sigma\,\Delta t}E_y^{i,n+1}(i,j,k) \\
&- \quad \frac{(\epsilon-\epsilon_0)\Delta t}{\epsilon+\sigma\,\Delta t}dE_y^{i,n+1}(i,j,k)
\end{aligned}
$$

$$
\begin{aligned}
E_z^{s,n+1}(i,j,k) = \frac{\epsilon}{\epsilon+\sigma\,\Delta t}E_z^{s,n}(i,j,k) \quad &+ \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta x}\left(H_y^{s,n}(i,j,k) - H_y^{s,n}(i-1,j,k)\right) \\
&- \quad \frac{\Delta t}{\epsilon+\sigma\,\Delta t}\frac{1}{\Delta y}\left(H_x^{s,n}(i,j,k) - H_x^{s,n}(i,j-1,k)\right) \\
&- \quad \frac{\sigma\,\Delta t}{\epsilon+\sigma\,\Delta t}E_z^{i,n+1}(i,j,k) \\
&- \quad \frac{(\epsilon-\epsilon_0)\Delta t}{\epsilon+\sigma\,\Delta t}dE_z^{i,n+1}(i,j,k)
\end{aligned}
\tag{A.3}
$$

$$
\begin{aligned}
H_x^{s,n+1}(i,j,k) = H_x^{s,n}(i,j,k) &- \frac{\Delta t}{\mu}\frac{1}{\Delta y}\left(E_z^{s,n}(i,j+1,k) - E_z^{s,n}(i,j,k)\right) \\
&+ \frac{\Delta t}{\mu}\frac{1}{\Delta z}\left(E_y^{s,n}(i,j,k+1) - E_y^{s,n}(i,j,k)\right) \\
H_y^{s,n+1}(i,j,k) = H_y^{s,n}(i,j,k) &- \frac{\Delta t}{\mu}\frac{1}{\Delta z}\left(E_x^{s,n}(i,j,k+1) - E_x^{s,n}(i,j,k)\right) \\
&+ \frac{\Delta t}{\mu}\frac{1}{\Delta x}\left(E_z^{s,n}(i+1,j,k) - E_z^{s,n}(i,j,k)\right) \\
H_z^{s,n+1}(i,j,k) = H_z^{s,n}(i,j,k) &- \frac{\Delta t}{\mu}\frac{1}{\Delta x}\left(E_y^{s,n}(i+1,j,k) - E_y^{s,n}(i,j,k)\right) \\
&+ \frac{\Delta t}{\mu}\frac{1}{\Delta y}\left(E_x^{s,n}(i,j+1,k) - E_x^{s,n}(i,j,k)\right)
\end{aligned}
\tag{A.4}
$$

A.3. ABC Gleichungen - First Order MUR

x-y-Ebenen

$$
\begin{aligned}
E_x^{n+1}(i,j,1) &= E_x^n(i,j,2) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_x^{n+1}(i,j,2) - E_x^n(i,j,1)\right) \\
E_x^{n+1}(i,j,z_m) &= E_x^n(i,j,z_m-1) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_x^{n+1}(i,j,z_m-1) - E_x^n(i,j,z_m)\right)
\end{aligned}
$$

$$
\begin{aligned}
E_y^{n+1}(i,j,1) &= E_y^n(i,j,2) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_y^{n+1}(i,j,2) - E_y^n(i,j,1)\right) \\
E_y^{n+1}(i,j,z_m) &= E_y^n(i,j,z_m-1) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_y^{n+1}(i,j,ax_z-1) - E_y^n(i,j,z_m)\right)
\end{aligned}
\tag{A.5}
$$

x-z-Ebenen

$$
\begin{aligned}
E_x^{n+1}(i,1,k) &= E_x^n(i,2,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_x^{n+1}(i,2,k) - E_x^n(i,1,k)\right) \\
E_x^{n+1}(i,y_m,k) &= E_x^n(i,y_m-1,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_x^{n+1}(i,y_m-1,k) - E_x^n(i,y_m,k)\right)
\end{aligned}
$$

$$
\begin{aligned}
E_z^{n+1}(i,1,k) &= E_z^n(i,2,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_z^{n+1}(i,2,k) - E_z^n(i,1,k)\right) \\
E_z^{n+1}(i,y_m,k) &= E_z^n(i,y_m-1,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_z^{n+1}(i,y_m-1,k) - E_z^n(i,y_m,k)\right)
\end{aligned}
\tag{A.6}
$$

y-z-Ebenen

$$
\begin{aligned}
E_y^{n+1}(1,j,k) &= E_y^n(2,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_y^{n+1}(2,j,k) - E_y^n(1,j,k)\right) \\
E_y^{n+1}(x_m,j,k) &= E_y^n(x_m-1,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_y^{n+1}(x_m-1,j,k) - E_y^n(x_m,j,k)\right)
\end{aligned}
$$

$$
\begin{aligned}
E_z^{n+1}(1,j,k) &= E_z^n(2,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_z^{n+1}(2,j,k) - E_z^n(1,j,k)\right) \\
E_z^{n+1}(x_m,j,k) &= E_z^n(x_m-1,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_z^{n+1}(x_m-1,j,k) - E_z^n(x_m,j,k)\right)
\end{aligned}
\tag{A.7}
$$

A.4. ABC Gleichungen - Second Order MUR

x-y-Ebenen

$$
\begin{aligned}
E_x^{n+1}(i,j,1) = \; & - \; E_x^{n-1}(i,j,2) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_x^{n+1}(i,j,2) + E_x^{n-1}(i,j,1)\right) \\
& + \; \frac{2\Delta z}{c_0\,\Delta t + \Delta z}\left(E_x^{n}(i,j,1) + E_x^{n}(i,j,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i+1,j,1) - 2E_x^{n}(i,j,1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i-1,j,1) + E_x^{n}(i+1,j,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(-2E_x^{n}(i,j,2) + E_x^{n}(i-1,j,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i,j+1,1) - 2E_x^{n}(i,j,1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i,j-1,1) + E_x^{n}(i,j+1,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(-2E_x^{n}(i,j,2) + E_x^{n}(i,j-1,2)\right)
\end{aligned}
\tag{A.8}
$$

$$
\begin{aligned}
E_x^{n+1}(i,j,z_m) = \; & - \; E_x^{n-1}(i,j,z_m-1) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_x^{n+1}(i,j,z_m-1) + E_x^{n-1}(i,j,z_m)\right) \\
& + \; \frac{2\Delta z}{c_0\,\Delta t + \Delta z}\left(E_x^{n}(i,j,z_m) + E_x^{n}(i,j,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i+1,j,z_m) - 2E_x^{n}(i,j,z_m)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i-1,j,z_m) + E_x^{n}(i+1,j,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(-2E_x^{n}(i,j,z_m-1) + E_x^{n}(i-1,j,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i,j+1,z_m) - 2E_x^{n}(i,j,z_m)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(E_x^{n}(i,j-1,z_m) + E_x^{n}(i,j+1,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(-2E_x^{n}(i,j,z_m-1) + E_x^{n}(i,j-1,z_m-1)\right)
\end{aligned}
\tag{A.9}
$$

$$
\begin{aligned}
E_y^{n+1}(i,j,1) = \; & - \; E_y^{n-1}(i,j,2) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_y^{n+1}(i,j,2) + E_y^{n-1}(i,j,1)\right) \\
& + \; \frac{2\Delta z}{c_0\,\Delta t + \Delta z}\left(E_y^{n}(i,j,1) + E_y^{n}(i,j,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i+1,j,1) - 2E_y^{n}(i,j,1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i-1,j,1) + E_y^{n}(i+1,j,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(-2E_y^{n}(i,j,2) + E_y^{n}(i-1,j,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i,j+1,1) - 2E_y^{n}(i,j,1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i,j-1,1) + E_y^{n}(i,j+1,2)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(-2E_y^{n}(i,j,2) + E_y^{n}(i,j-1,2)\right)
\end{aligned}
\tag{A.10}
$$

$$
\begin{aligned}
E_y^{n+1}(i,j,z_m) = \; & - \; E_y^{n-1}(i,j,z_m-1) + \frac{c_0\,\Delta t - \Delta z}{c_0\,\Delta t + \Delta z}\left(E_y^{n+1}(i,j,z_m-1) + E_y^{n-1}(i,j,z_m)\right) \\
& + \; \frac{2\Delta z}{c_0\,\Delta t + \Delta z}\left(E_y^{n}(i,j,z_m) + E_y^{n}(i,j,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i+1,j,z_m) - 2E_y^{n}(i,j,z_m)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i-1,j,z_m) + E_y^{n}(i+1,j,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(-2E_y^{n}(i,j,z_m-1) + E_y^{n}(i-1,j,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i,j+1,z_m) - 2E_y^{n}(i,j,z_m)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta z)}\left(E_y^{n}(i,j-1,z_m) + E_y^{n}(i,j+1,z_m-1)\right) \\
& + \; \frac{\Delta z (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta z)}\left(-2E_y^{n}(i,j,z_m-1) + E_y^{n}(i,j-1,z_m-1)\right)
\end{aligned}
\tag{A.11}
$$

x-z-Ebenen

$$
\begin{aligned}
E_x^{n+1}(i,1,k) = \; & - \; E_x^{n-1}(i,2,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_x^{n+1}(i,2,k) + E_x^{n-1}(i,1,k)\right) \\
& + \; \frac{2\Delta y}{c_0\,\Delta t + \Delta y}\left(E_x^n(i,1,k) + E_x^n(i,2,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i+1,1,k) - 2E_x^n(i,1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i-1,1,k) + E_x^n(i+1,2,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(-2E_x^n(i,2,k) + E_x^n(i-1,2,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i,1,k+1) - 2E_x^n(i,1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i,1,k-1) + E_x^n(i,2,k+1)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(-2E_x^n(i,2,k) + E_x^n(i,2,k-1)\right)
\end{aligned}
\tag{A.12}
$$

$$
\begin{aligned}
E_x^{n+1}(i,y_m,k) = \; & - \; E_x^{n-1}(i,y_m-1,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_x^{n+1}(i,y_m-1,k) + E_x^{n-1}(i,y_m,k)\right) \\
& + \; \frac{2\Delta y}{c_0\,\Delta t + \Delta y}\left(E_x^n(i,y_m,k) + E_x^n(i,y_m-1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i+1,y_m,k) - 2E_x^n(i,y_m,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i-1,y_m,k) + E_x^n(i+1,y_m-1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(-2E_x^n(i,y_m-1,k) + E_x^n(i-1,y_m-1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i,y_m,k+1) - 2E_x^n(i,y_m,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_x^n(i,y_m,k-1) + E_x^n(i,y_m-1,k+1)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(-2E_x^n(i,y_m-1,k) + E_x^n(i,y_m-1,k-1)\right)
\end{aligned}
\tag{A.13}
$$

$$
\begin{aligned}
E_z^{n+1}(i,1,k) = \; & - \; E_z^{n-1}(i,2,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_z^{n+1}(i,2,k) + E_z^{n-1}(i,1,k)\right) \\
& + \; \frac{2\Delta y}{c_0\,\Delta t + \Delta y}\left(E_z^{n}(i,1,k) + E_z^{n}(i,2,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i+1,1,k) - 2E_z^{n}(i,1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i-1,1,k) + E_z^{n}(i+1,2,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(-2E_z^{n}(i,2,k) + E_z^{n}(i-1,2,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i,1,k+1) - 2E_z^{n}(i,1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i,1,k-1) + E_z^{n}(i,2,k+1)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(-2E_z^{n}(i,2,k) + E_z^{n}(i,2,k-1)\right)
\end{aligned}
\tag{A.14}
$$

$$
\begin{aligned}
E_z^{n+1}(i,y_m,k) = \; & - \; E_z^{n-1}(i,y_m-1,k) + \frac{c_0\,\Delta t - \Delta y}{c_0\,\Delta t + \Delta y}\left(E_z^{n+1}(i,y_m-1,k) + E_z^{n-1}(i,y_m,k)\right) \\
& + \; \frac{2\Delta y}{c_0\,\Delta t + \Delta y}\left(E_z^{n}(i,y_m,k) + E_z^{n}(i,y_m-1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i+1,y_m,k) - 2E_z^{n}(i,y_m,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i-1,y_m,k) + E_z^{n}(i+1,y_m-1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(-2E_z^{n}(i,y_m-1,k) + E_z^{n}(i-1,y_m-1,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i,y_m,k+1) - 2E_z^{n}(i,y_m,k)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta x^2 (c_0\,\Delta t + \Delta y)}\left(E_z^{n}(i,y_m,k-1) + E_z^{n}(i,y_m-1,k+1)\right) \\
& + \; \frac{\Delta y (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta y)}\left(-2E_z^{n}(i,y_m-1,k) + E_z^{n}(i,y_m-1,k-1)\right)
\end{aligned}
\tag{A.15}
$$

y-z-Ebenen

$$
\begin{aligned}
E_y^{n+1}(1,j,k) = \; & - \; E_y^{n-1}(2,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_y^{n+1}(2,j,k) + E_y^{n-1}(1,j,k)\right) \\
& + \; \frac{2\Delta x}{c_0\,\Delta t + \Delta x}\left(E_y^{n}(1,j,k) + E_y^{n}(2,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(1,j+1,k) - 2E_y^{n}(1,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(1,j-1,k) + E_y^{n}(2,j+1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(-2E_y^{n}(2,j,k) + E_y^{n}(2,j-1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(1,j,k+1) - 2E_y^{n}(1,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(1,j,k-1) + E_y^{n}(2,j,k+1)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(-2E_y^{n}(2,j,k) + E_y^{n}(2,j,k-1)\right)
\end{aligned}
\tag{A.16}
$$

$$
\begin{aligned}
E_y^{n+1}(x_m,j,k) = \; & - \; E_y^{n-1}(x_m-1,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_y^{n+1}(x_m-1,j,k) + E_y^{n-1}(x_m,j,k)\right) \\
& + \; \frac{2\Delta x}{c_0\,\Delta t + \Delta x}\left(E_y^{n}(x_m,j,k) + E_y^{n}(x_m-1,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(x_m,j+1,k) - 2E_y^{n}(x_m,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(x_m,j-1,k) + E_y^{n}(x_m-1,j+1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(-2E_y^{n}(x_m-1,j,k) + E_y^{n}(x_m-1,j-1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(x_m,j,k+1) - 2E_y^{n}(x_m,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_y^{n}(x_m,j,k-1) + E_y^{n}(x_m-1,j,k+1)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(-2E_y^{n}(x_m-1,j,k) + E_y^{n}(x_m-1,j,k-1)\right)
\end{aligned}
\tag{A.17}
$$

$$
\begin{aligned}
E_z^{n+1}(1,j,k) = \; & - \; E_z^{n-1}(2,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_z^{n+1}(2,j,k) + E_z^{n-1}(1,j,k)\right) \\
& + \; \frac{2\Delta x}{c_0\,\Delta t + \Delta x}\left(E_z^n(1,j,k) + E_z^n(2,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(1,j+1,k) - 2E_z^n(1,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(1,j-1,k) + E_z^n(2,j+1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n - 2E_z^n(2,j,k) + E_z^n(2,j-1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(1,j,k+1) - 2E_z^n(1,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(1,j,k-1) + E_z^n(2,j,k+1)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(-2E_z^n(2,j,k) + E_z^n(2,j,k-1)\right)
\end{aligned} \tag{A.18}
$$

$$
\begin{aligned}
E_z^{n+1}(x_m,j,k) = \; & - \; E_z^{n-1}(x_m-1,j,k) + \frac{c_0\,\Delta t - \Delta x}{c_0\,\Delta t + \Delta x}\left(E_z^{n+1}(x_m-1,j,k) + E_z^{n-1}(x_m,j,k)\right) \\
& + \; \frac{2\Delta x}{c_0\,\Delta t + \Delta x}\left(E_z^n(x_m,j,k) + E_z^n(x_m-1,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(x_m,j+1,k) - 2E_z^n(x_m,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(x_m,j-1,k) + E_z^n(x_m-1,j+1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(-2E_z^n(x_m-1,j,k) + E_z^n(x_m-1,j-1,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(x_m,j,k+1) - 2E_z^n(x_m,j,k)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta y^2 (c_0\,\Delta t + \Delta x)}\left(E_z^n(x_m,j,k-1) + E_z^n(x_m-1,j,k+1)\right) \\
& + \; \frac{\Delta x (c_0\Delta t)^2}{2\Delta z^2 (c_0\,\Delta t + \Delta x)}\left(-2E_z^n(x_m-1,j,k) + E_z^n(x_m-1,j,k-1)\right)
\end{aligned} \tag{A.19}
$$

B. Diagramme

B.1. Flussdiagramm - Implementierung der FDTD

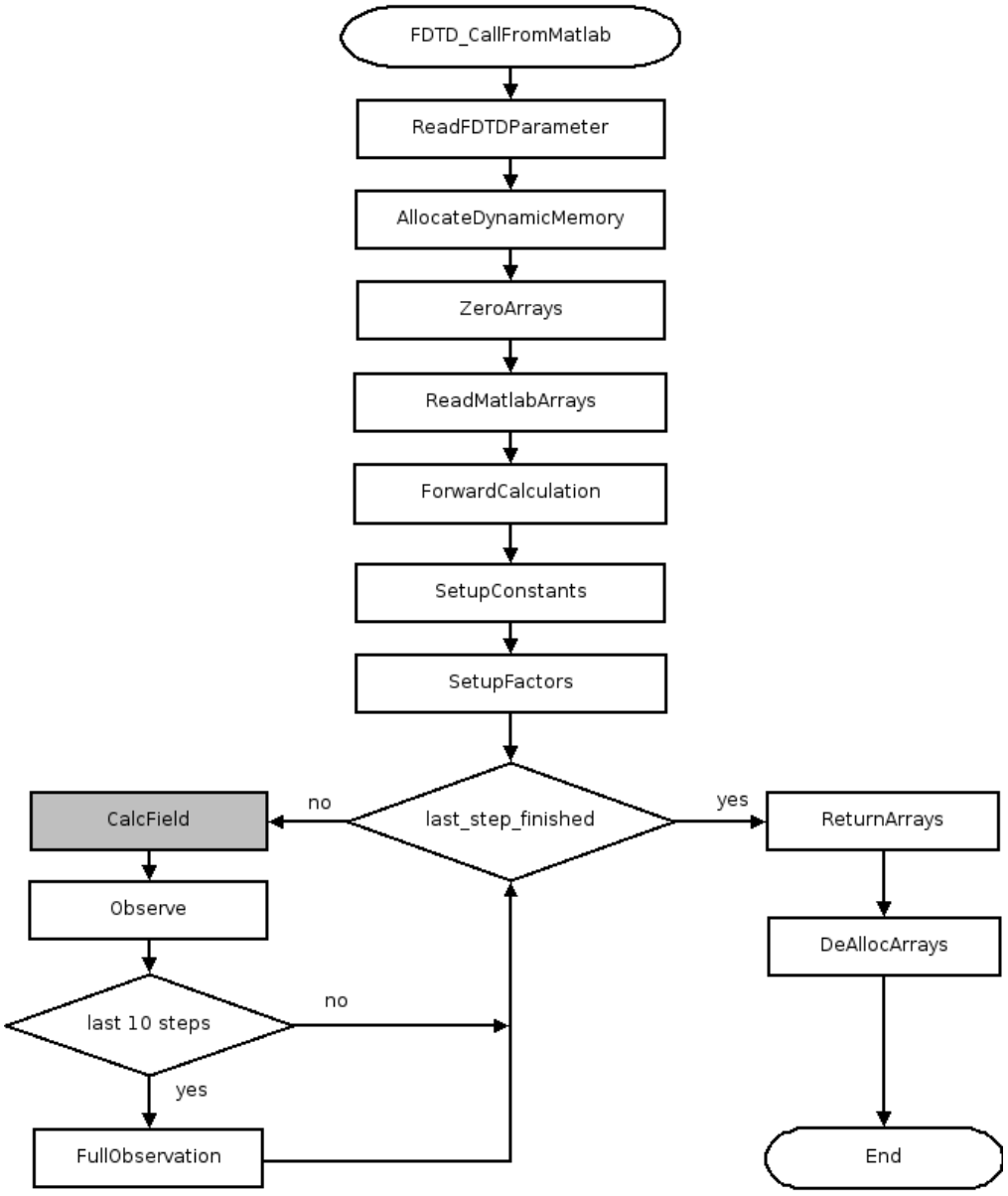

Abbildung 49: **Flussdiagramm - Implementierung der FDTD**

B.2. Flussdiagramm - FDTD(CalcField)

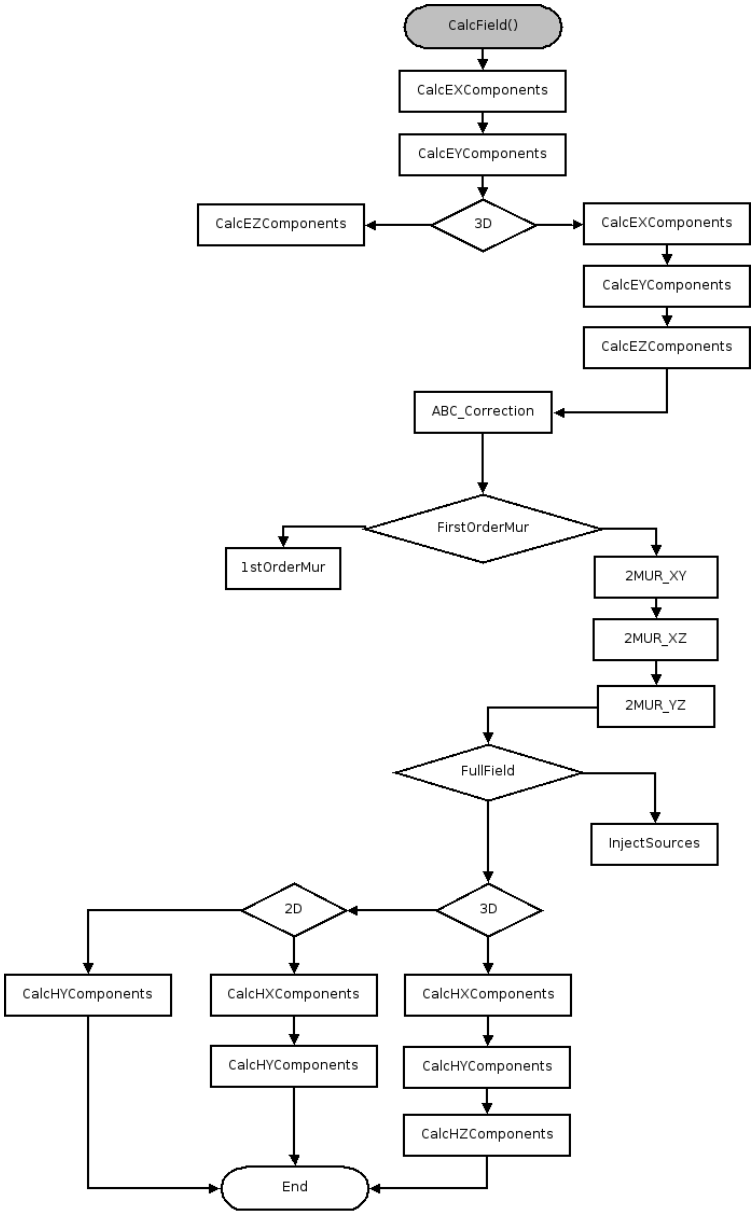

Abbildung 50: **Flussdiagramm - CalcField(FDTD)**

B.3. Flussdiagramm - Optimierung - allgemein

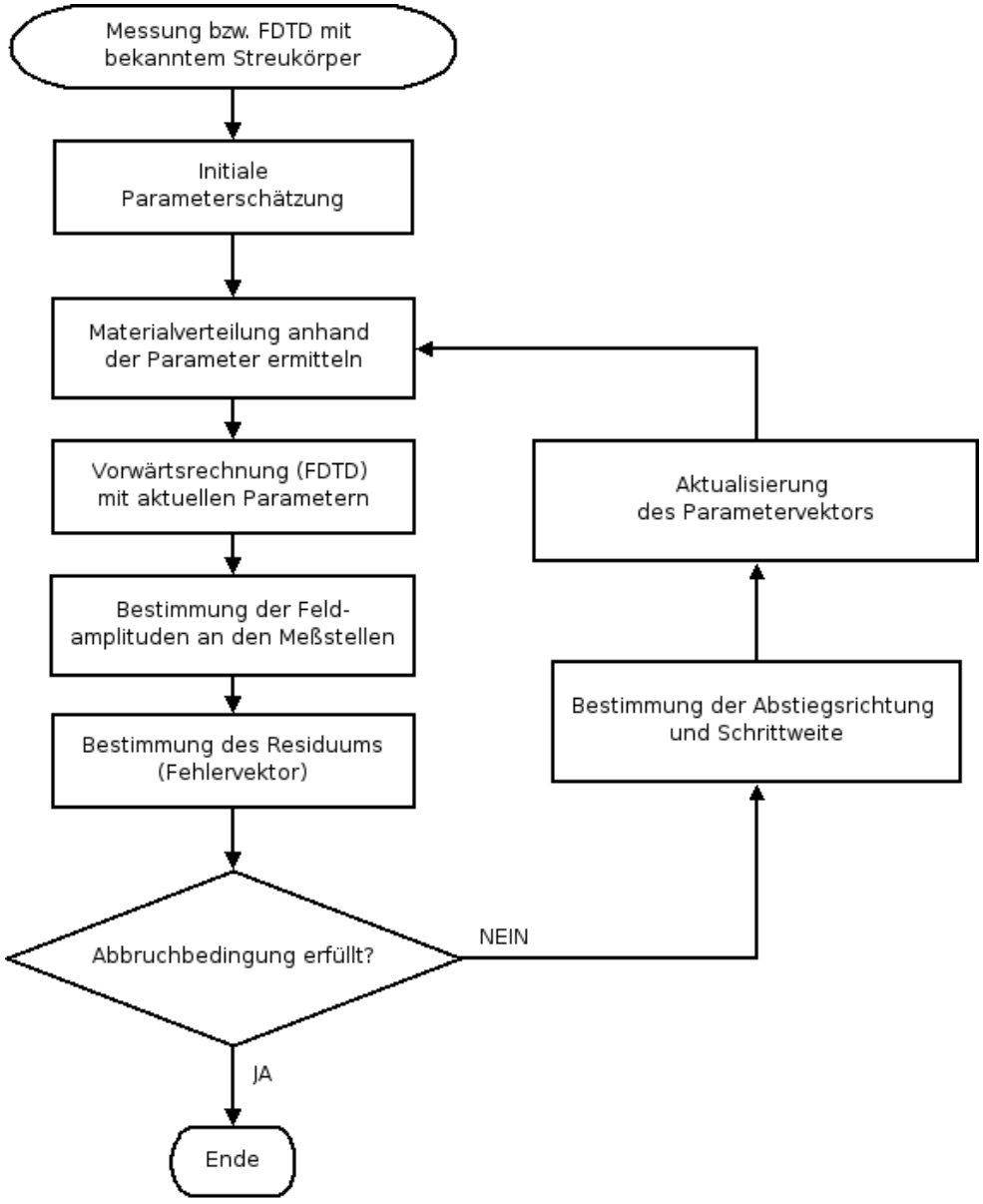

Abbildung 51: **Flussdiagramm - Implementierung der Optimierung**

B.4. Flussdiagramm - Gradientenverfahren

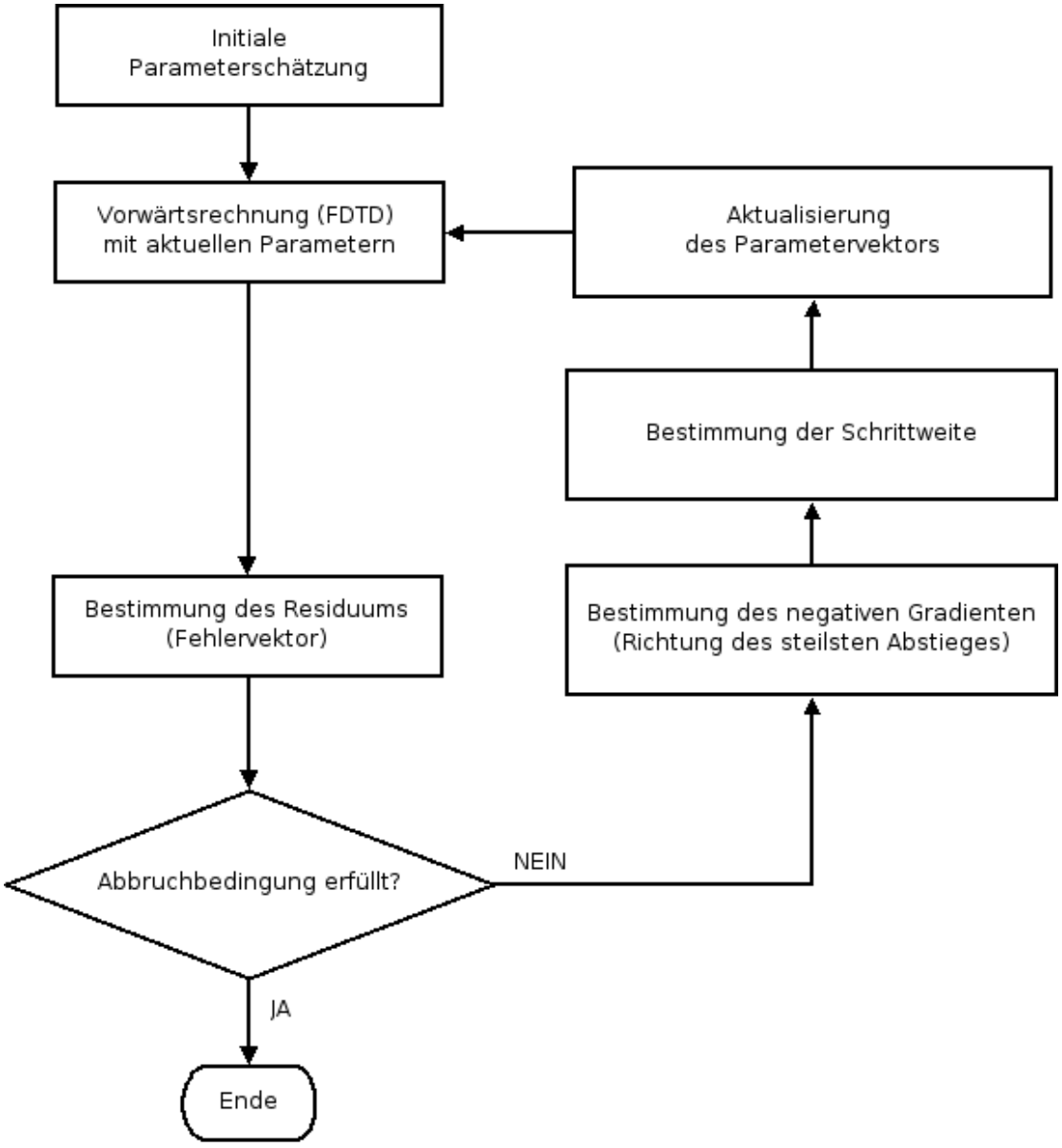

Abbildung 52: **Flussdiagramm - Optimierung mittels SD-Verfahren**

B.5. Flussdiagramm - Gauß-Newton Verfahren

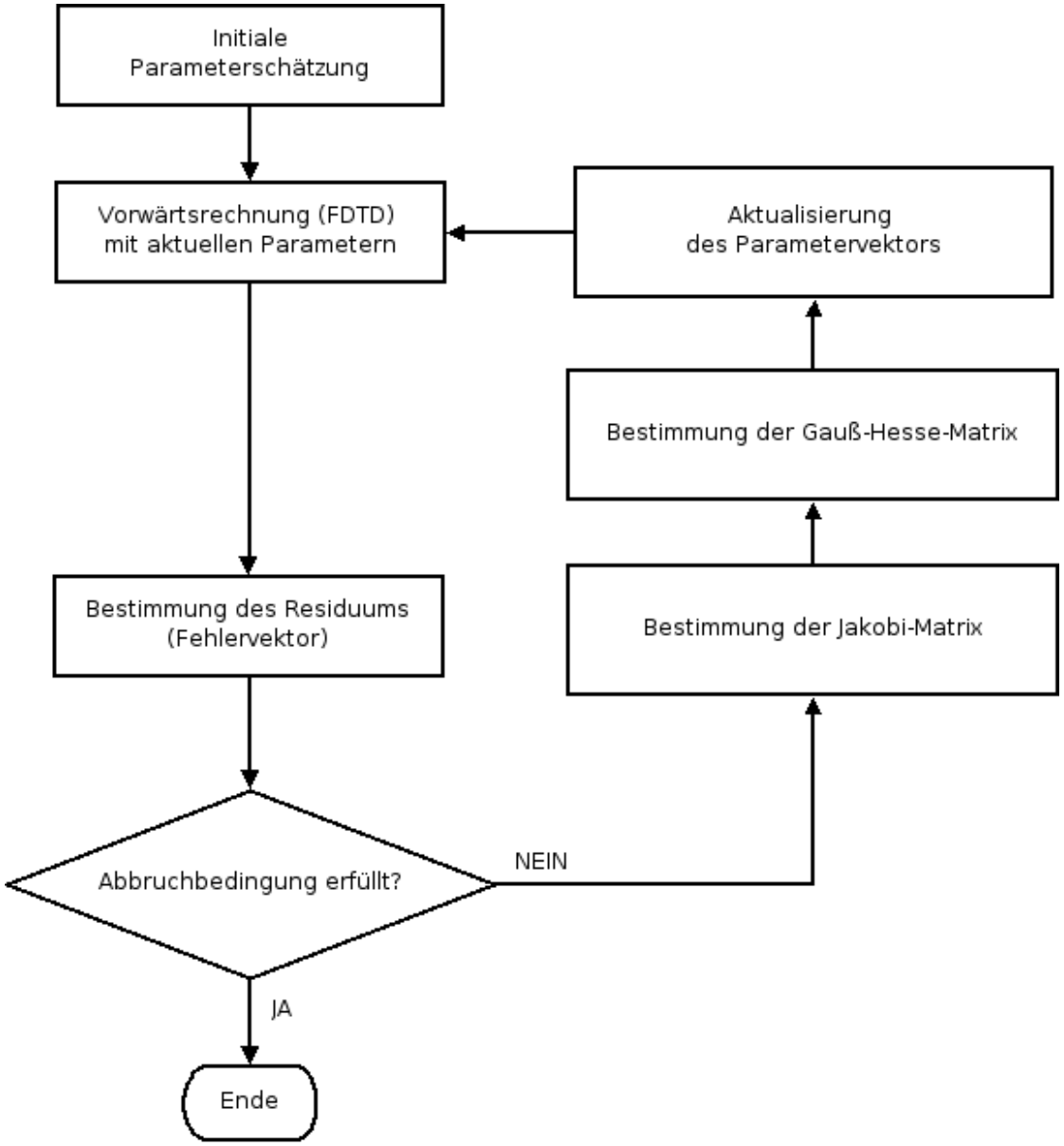

Abbildung 53: **Flussdiagramm - Optimierung mittels GN-Verfahren**

B.6. Flussdiagramm - Marquardt-Levenberg Verfahren

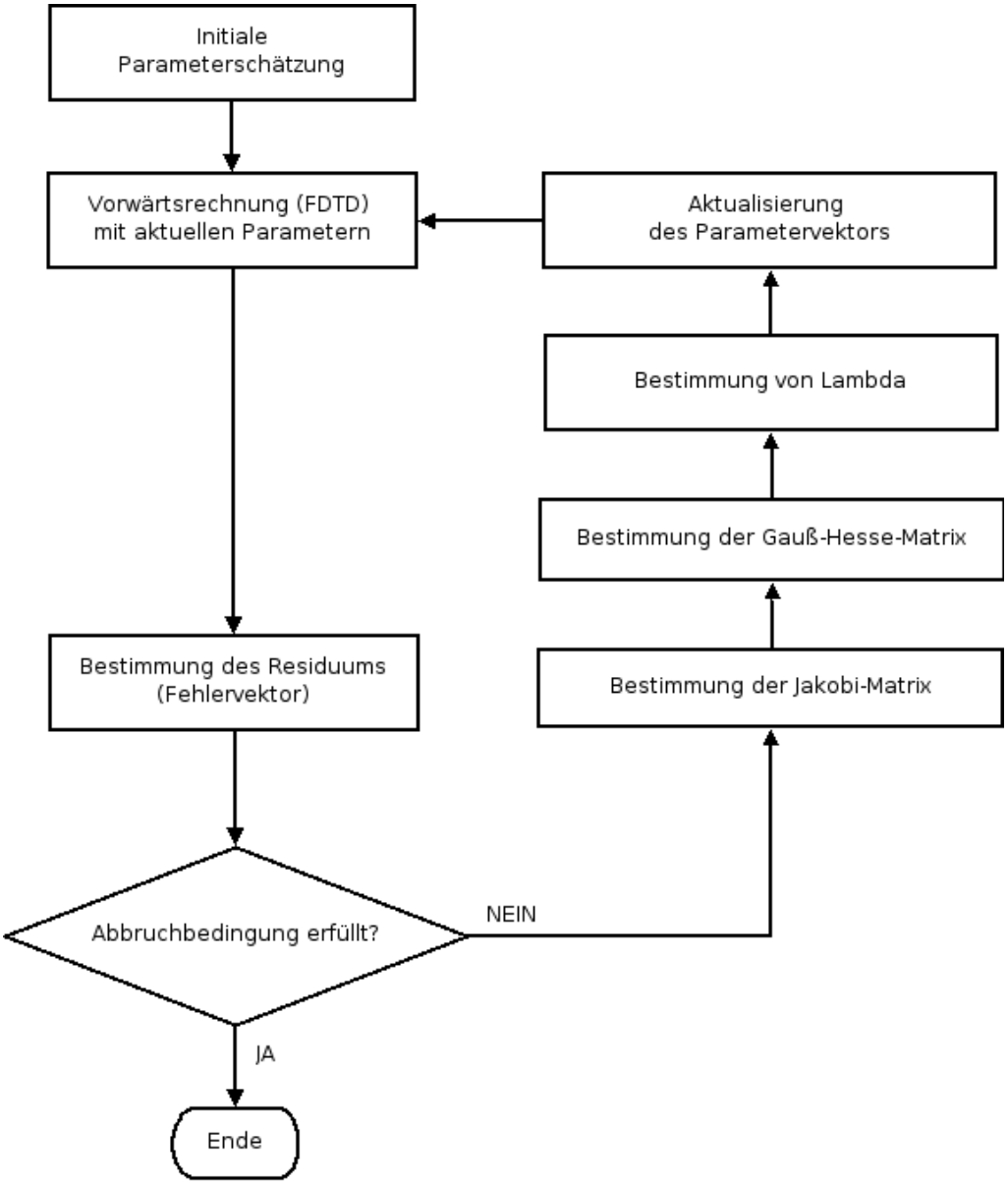

Abbildung 54: **Flussdiagramm - Optimierung mittels LM-Verfahren**

C. Listings - FDTD

C.1. Globale Definitionen

Listing 6: FDTD - Globale Definitionen

```
!#######################################
!  global module for use in each function
!#######################################
MODULE globals
        INCLUDE 'header3d.f90'

!       ------------------------------------------------------------
!       FDTD Parameter (overwriteable by function call out of matlab)
!       ------------------------------------------------------------
        INTEGER :: max_x=100, max_y=100, max_z=100      ! size of the problem space
        REAL*8  :: delta_x=1, delta_y=1, delta_z=1
        INTEGER :: time_steps=200                       ! No of time steps

        REAL*8  :: frequency     = 25*10**6             ! frequency (for sinusoidal source)
        INTEGER :: amp           = 0                    ! amplitude

        LOGICAL :: FullXYObservation    = .FALSE.       ! when the complete xy-area is to be
                                                        ! observed, an x-y-t array will be
                                                        ! returned instead of an x*y-t array
        INTEGER :: FULLXY_Observation_Z = 1             ! which x-y plane (z-coordinate)
        INTEGER :: NoOfObsPoints        = 10            ! number of observation points
                                                        ! (each timestep will be recorded)
        INTEGER :: NoOfReceivers        = 9             ! number of ReceiverPoints
                                                        ! (all componets, only the last steps)
        INTEGER :: NoOfSources          = 1

        LOGICAL :: ScatteredFieldOnly   = .FALSE.       ! seperated Field approach?!
        LOGICAL :: D2_only              = .FALSE.       ! only EZ, HX and HY are calculated
        LOGICAL :: D1_only              = .FALSE.       ! only EZ and HY are calculated
                                                        ! -> 1D expansion in x-direction
        INTEGER :: ABC_Type             = 1             ! which kind of ABC is to be used

        INTEGER :: MaxNoOfEMat  = 390
        INTEGER :: MaxNoOfHMat  = 1

!       ------------------------------
!       precalculated constant factors
!       ------------------------------
        REAL*8  :: c0                                           ! light speed in vakuum
        REAL*8  :: alpha, beta_dt, period, delay                ! incident ...
        REAL*8  :: amp_x, amp_y, amp_z, disp_x, disp_y, disp_z  ! ...field stuff
        REAL*8  :: omega                                        ! for the harmonic source
        REAL*8  :: delta_t,time                                 ! timestep width,current time

!       ------------------------------
!       predefined material constants
!       ------------------------------
        REAL*8, ALLOCATABLE     :: epsilon_r(:), sigma(:), mu_r(:)
```

```
!       ------------------------------------------------
!       precalculated factors for the update functions
!       ------------------------------------------------
        REAL*8                      :: dt_edx, dt_edy, dt_edz
        REAL*8, ALLOCATABLE         :: e_esdt(:), dtee_esdt(:), dts_esdt(:)
        REAL*8, ALLOCATABLE         :: dt_esdtdx(:), dt_esdtdy(:), dt_esdtdz(:)

        REAL*8                      :: dt_mdx, dt_mdy, dt_mdz
!       REAL*8, ALLOCATABLE         :: m_msdt(:), dtmm_msdt(:), dts_msdt(:)          ! for loosy ..
!       REAL*8, ALLOCATABLE         :: dt_msdtdx(:), dt_msdtdy(:), dt_msdtdz(:)      ! ..mag. materials

!       ------------------------------------------------
!       precalculated factors for the ABC functions
!       ------------------------------------------------
        REAL*8                      :: cdtdx_cdtdx, cdtdy_cdtdy, cdtdz_cdtdz
        REAL*8                      :: dx_cdtdx, dy_cdtdy, dz_cdtdz
        REAL*8                      :: dxcdt_2dycdtdx, dxcdt_2dzcdtdx
        REAL*8                      :: dycdt_2dxcdtdy, dycdt_2dzcdtdy
        REAL*8                      :: dzcdt_2dxcdtdz, dzcdt_2dycdtdz

!       ------------------------------------------------
!       Field Arrays (E and M field for each cell)
!       ------------------------------------------------
        REAL*8,  ALLOCATABLE        :: ES_X(:,:,:), ES_Y(:,:,:), ES_Z(:,:,:)
        REAL*8,  ALLOCATABLE        :: HS_X(:,:,:), HS_Y(:,:,:), HS_Z(:,:,:)

!       ------------------------------------------------
!       Material Array (material number for each cell)
!       ------------------------------------------------
        INTEGER*2, ALLOCATABLE      :: EMat(:,:,:) ! HMAT(:,:,:)

!       ------------------------------------------------------------------------
!       previous and pre-previous values of innerboundary cells for the ABC
!       ------------------------------------------------------------------------
        REAL*8, ALLOCATABLE         :: PRE_ESX_XZ(:,:,:), PRE_ESX_XY(:,:,:)
        REAL*8, ALLOCATABLE         :: PRE_ESY_YZ(:,:,:), PRE_ESY_XY(:,:,:)
        REAL*8, ALLOCATABLE         :: PRE_ESZ_YZ(:,:,:), PRE_ESZ_XZ(:,:,:)
        REAL*8, ALLOCATABLE         :: PREPRE_ESX_XZ(:,:,:), PREPRE_ESX_XY(:,:,:)
        REAL*8, ALLOCATABLE         :: PREPRE_ESY_YZ(:,:,:), PREPRE_ESY_XY(:,:,:)
        REAL*8, ALLOCATABLE         :: PREPRE_ESZ_YZ(:,:,:), PREPRE_ESZ_XZ(:,:,:)


!       ------------------------------------------------------------------------
!       Field values for the points of interest
!       ------------------------------------------------------------------------
        REAL*8,    ALLOCATABLE      :: ReceiverData(:,:,:)
        INTEGER*2, ALLOCATABLE      :: ReceiverPoints(:,:)

        REAL*8,    ALLOCATABLE      :: ObservationData(:,:), XYObservationData(:,:,:)
        INTEGER*2, ALLOCATABLE      :: ObservationPoints(:,:)

        INTEGER*2, ALLOCATABLE      :: Sources(:,:)

END MODULE globals
```

C.2. MATLAB® Interface

Listing 7: FDTD - Matlab Interface

```
!############################
!  The matlab gateway routine
!############################
SUBROUTINE mexFunction(nlhs, plhs, nrhs, prhs)

        USE globals

!       MATLAB stuff
        INTEGER :: mxGetM, mxGetN, mxGetPr
        INTEGER :: mxIsNumeric, mxCreateDoubleMatrix, mxCreateNumericArray
        INTEGER :: mxClassIDFromClassName, mxCopyPtrToInteger4, mxGetDimensions
        INTEGER :: plhs(*), prhs(*)
        INTEGER :: nlhs, nrhs

!       Inputs
        INTEGER :: size1                        ! Input1 - Parameter
        INTEGER :: ExternalParams_pr
        REAL*8  :: ExternalParams(17)

        INTEGER :: epsilon_r_pr                 ! Input2 - epsilon_r array
        INTEGER :: sigma_pr                     ! Input3 - sigma array

        INTEGER :: EMat_pr                      ! Input4 - EMat array

        INTEGER :: size5                        ! Input5 - ReceiverPoints
        INTEGER :: ReceiverPoints_pr

        INTEGER :: size6                        ! Input6 - ObservationPoints
        INTEGER :: ObservationPoints_pr

        INTEGER :: size7                        ! Input7 - Sources
        INTEGER :: Sources_pr

!       Outputs
        INTEGER :: ReceiverData_pr              ! Pointer to the ReceiverPoints
        INTEGER :: ObsPointData_pr              ! Pointer to the ObservationPoints

        INTEGER :: Parameter_pr                 ! Some OutputInformation
        REAL*8  :: Param(1)

        INTEGER :: dims(3)


!---------------
!Get the inputs
!---------------

        size1                   = mxGetM(prhs(1)) * mxGetN(prhs(1))
        ExternalParams_pr       = mxGetPr(prhs(1))
        CALL mxCopyPtrToReal8(ExternalParams_pr, ExternalParams, size1)

        max_x                   = ExternalParams(1)
```

```
        max_y                   = ExternalParams(2)
        max_z                   = ExternalParams(3)
        delta_x                 = ExternalParams(4)
        delta_y                 = ExternalParams(5)
        delta_z                 = ExternalParams(6)
        time_steps              = ExternalParams(7)
        frequency               = ExternalParams(8)
        amp                     = ExternalParams(9)
        NoOfReceivers           = ExternalParams(10)
        NoOfObsPoints           = ExternalParams(11)
        NoOfSources             = ExternalParams(12)
        ABC_Type                = ExternalParams(13)
        FULLXY_Observation_Z    = ExternalParams(14)
        IF(ExternalParams(15) == 1) THEN
                ScatteredFieldOnly = .TRUE.
```

C.3. FDTD Main-Loop

Listing 8: FDTD - MainLoop

```
!##############
!Doing the FDTD
!##############
SUBROUTINE ForwardCalculation

        USE globals
        INTEGER :: time_step



        CALL SetupConstants
        CALL SetupFactors

        DO time_step = 1, time_steps    - 10
                CALL CalcField
                CALL Observe(time_step)
        END DO

        DO time_step = time_steps - 9, time_steps
                CALL CalcField
                CALL Observe(time_step)
                CALL FullObservation(10-(time_steps-time_step))
        END DO

        RETURN
END
```

C.4. Konstanten berechnen

Listing 9: FDTD - SetupConstants

```
!#####################################
!precalculate some misc constant values
!#####################################
SUBROUTINE SetupConstants

        USE globals
        REAL    :: pi

!-------------------------------------------------------------------------------
!calculate the time-step-width (limited by the cell size and the stability condition)
!-------------------------------------------------------------------------------
        c0=1./(sqrt(mu_0*epsilon_0))
        delta_t=1./SQRT((c0/delta_x)**2+(c0/delta_y)**2+(c0/delta_z)**2)

!-----------------------------------------------
!calculate some parameter for the incident field
!-----------------------------------------------

        pi      = 4.0*ATAN(1.0)
        beta_dt = beta*delta_t

        alpha   = (1./(beta_dt/4.0))**2
        period  = 2.0*beta_dt


        COSTH   = COS(pi*THINC/180.)
        SINTH   = SIN(pi*THINC/180.)
        COSPH   = COS(pi*PHINC/180.)
        SINPH   = SIN(pi*PHINC/180.)

        delay   = 0.0

        IF (disp_x.LT.0.) delay = delay-disp_x*max_x-1*delta_x
        IF (disp_y.LT.0.) delay = delay-disp_y*max_y-1*delta_y
        IF (disp_z.LT.0.) delay = delay-disp_z*max_z-1*delta_z

        amp_x   = amp * ( ETHINC*COSTH*COSPH - EPHINC*SINPH )
        amp_y   = amp * ( ETHINC*COSTH*SINPH + EPHINC*COSPH )
        amp_z   = amp * ( -ETHINC*SINTH )

        disp_x  = -COSPH*SINTH
        disp_y  = -SINPH*SINTH
        disp_z  = -COSTH

        omega = 2 * pi * frequency
```

C.5. Faktoren berechnen

Listing 10: FDTD - SetupFactors

```
!########################################
!precalculate FDTD specific constant values
!########################################
SUBROUTINE SetupFactors

        USE globals
        REAL            :: eps(MaxNoOfEMat)
        INTEGER         :: i


!-----------------------------------------------------------------------
!calculate the factors for the update functions for the electric field
!-----------------------------------------------------------------------

!       --free space constants--
        dt_edx = delta_t/(epsilon_0*delta_x)
        dt_edy = delta_t/(epsilon_0*delta_y)
        dt_edz = delta_t/(epsilon_0*delta_z)

!       --lossy dielectric constants--
        DO i=1, MaxNoOfEMat
                eps(i)=epsilon_0*epsilon_r(i)

                e_esdt(i)       = eps(i)/(eps(i)+sigma(i)*delta_t)
                dt_esdtdx(i)    = delta_t/((eps(i)+sigma(i)*delta_t)*delta_x)
                dt_esdtdy(i)    = delta_t/((eps(i)+sigma(i)*delta_t)*delta_y)
                dt_esdtdz(i)    = delta_t/((eps(i)+sigma(i)*delta_t)*delta_z)
                dts_esdt(i)     = (delta_t*sigma(i))/(eps(i)+sigma(i)*delta_t)
                dtee_esdt(i)    = (delta_t*(eps(i)-epsilon_0))/(eps(i)+sigma(i)*delta_t)
        END DO


!-----------------------------------------------------------------------
!calculate the factors for the update functions for the magnetic field
!-----------------------------------------------------------------------

!       --free space constants only (in this implementation)--
        dt_mdx = delta_t/(mu_0*delta_x)
        dt_mdy = delta_t/(mu_0*delta_y)
        dt_mdz = delta_t/(mu_0*delta_z)


!-----------------------------------------------
!calculate the factors for the ABC calculations
!-----------------------------------------------

!       FirstOrderMUR factors
        cdtdx_cdtdx = (c0 * delta_t - delta_x)/(c0 * delta_t + delta_x)
        cdtdy_cdtdy = (c0 * delta_t - delta_y)/(c0 * delta_t + delta_y)
        cdtdz_cdtdz = (c0 * delta_t - delta_z)/(c0 * delta_t + delta_z)

!       SecondOrderMUR factors
```

```
        dx_cdtdx        = (2.*delta_x)/(c0*delta_t + delta_x)
        dy_cdtdy        = (2.*delta_y)/(c0*delta_t + delta_y)
        dz_cdtdz        = (2.*delta_y)/(c0*delta_t + delta_z)
        dxcdt_2dycdtdx  = (delta_x*((c0*delta_t)**2))/(2.*(delta_y**2)*(c0*delta_t+delta_x))
        dxcdt_2dzcdtdx  = (delta_x*((c0*delta_t)**2))/(2.*(delta_z**2)*(c0*delta_t+delta_x))
        dycdt_2dxcdtdy  = (delta_y*((c0*delta_t)**2))/(2.*(delta_x**2)*(c0*delta_t+delta_y))
        dycdt_2dzcdtdy  = (delta_y*((c0*delta_t)**2))/(2.*(delta_z**2)*(c0*delta_t+delta_y))
        dzcdt_2dxcdtdz  = (delta_z*((c0*delta_t)**2))/(2.*(delta_x**2)*(c0*delta_t+delta_z))
        dzcdt_2dycdtdz  = (delta_z*((c0*delta_t)**2))/(2.*(delta_y**2)*(c0*delta_t+delta_z))

        RETURN
END
```

C.6. Felder aktualisieren

Listing 11: FDTD - CalcField

```
!######################################################################
!Calculation of the electric and magnetic field at the next time-step
!######################################################################
SUBROUTINE CalcField
        USE globals

        IF(D1_only == .TRUE.) THEN
                CALL Calc_EZ_Components
        ELSE IF (D2_only == .TRUE.) THEN
                CALL Calc_EZ_Components
        ELSE
                CALL Calc_EX_Components
                CALL Calc_EY_Components
                CALL Calc_EZ_Components
        END IF

        CALL ABC_Correction

        IF(ScatteredFieldOnly == FALSE) THEN
                CALL InjectSources
        END IF
        time=time+delta_t/2
        IF(D1_only == .TRUE.) THEN
                CALL Calc_HY_Components
        ELSE IF (D2_only == .TRUE.) THEN
                CALL Calc_HX_Components
                CALL Calc_HY_Components
        ELSE
                CALL Calc_HX_Components
                CALL Calc_HY_Components
                CALL Calc_HZ_Components
        END IF
        time=time+delta_t/2
        RETURN
END
```

C.6.1. E_x Update

Listing 12: FDTD - CalcEX

```
!#################################################
!Calculation of the x-component of the electric field
!#################################################
SUBROUTINE Calc_EX_Components

        USE globals
        INTEGER :: i,j,k,Mat



        DO k=2, max_z-1
                DO j=2, max_y-1
                        DO i=1, max_x-1
                                Mat = EMAT(i,j,k)
!                               Free Space
                                IF(Mat==0) THEN
                                        ES_X(i,j,k) = ES_X(i,j,k) &
&                                                     + dt_edy * ( HS_Z(i,j,k) - HS_Z(i,j-1,k) ) &
&                                                     - dt_edz * ( HS_Y(i,j,k) - HS_Y(i,j,k-1) )
!                               Perfect Conductor
                                ELSE IF(Mat==1)THEN
                                        ES_X(i,j,k) = - ES_X(i,j,k)
!                               Lossy Material
                                ELSE
                                        ES_X(i,j,k) = e_esdt(Mat)*ES_X(i,j,k) &
&                                                     - dtee_esdt(Mat)*DEI_X(i,j,k) &
&                                                     - dts_esdt(Mat) *EI_X(i,j,k) &
&                                                     + dt_esdtdy(Mat)*(HS_Z(i,j,k)-HS_Z(i,j-1,k))&
&                                                     - dt_esdtdz(Mat)*(HS_Y(i,j,k)-HS_Y(i,j,k-1))
                                END IF
                        END DO
                END DO
        END DO

        RETURN
END
```

C.6.2. E_y Update

Listing 13: FDTD - CalcEY

```
!#################################################
!Calculation of the y-component of the electric field
!#################################################
SUBROUTINE Calc_EY_Components

        USE globals
        INTEGER :: i,j,k,Mat
```

```
        DO k=2, max_z-1
                DO j=1, max_y-1
                        DO i=2, max_x-1
                                Mat = EMAT(i,j,k)
!                               Free Space
                                IF(Mat==0) THEN
                                        ES_Y(i,j,k) = ES_Y(i,j,k)&
&                                                     + dt_edz * ( HS_X(i,j,k) - HS_X(i,j,k-1) ) &
&                                                     - dt_edx * ( HS_Z(i,j,k) - HS_Z(i-1,j,k) )
!                               Perfect Conductor
                                ELSE IF(Mat==1)THEN
                                        ES_Y(i,j,k) = - ES_Y(i,j,k)
!                               Lossy Material
                                ELSE
                                        ES_Y(i,j,k) = e_esdt(Mat)*ES_Y(i,j,k) &
&                                                     - dtee_esdt(Mat)*DEI_Y(i,j,k) &
&                                                     - dts_esdt(Mat) *EI_Y(i,j,k) &
&                                                     + dt_esdtdz(Mat)*(HS_X(i,j,k)-HS_X(i,j,k-1))&
&                                                     - dt_esdtdx(Mat)*(HS_Z(i,j,k)-HS_Z(i-1,j,k))
                                END IF
                        END DO
                END DO
        END DO

        RETURN
END
```

C.6.3. E_z Update

Listing 14: FDTD - CalcEZ

```
!#################################################
!Calculation of the z-component of the electric field
!#################################################
SUBROUTINE Calc_EZ_Components

        USE globals
        INTEGER :: i,j,k,Mat


        DO k=1, max_z-1
                DO j=2, max_y-1
                        DO i=2, max_x-1
                                Mat = EMAT(i,j,k)
!                               Free Space
                                IF(Mat==0) THEN
                                        ES_Z(i,j,k) = ES_Z(i,j,k) &
&                                                     + dt_edx * (HS_Y(i,j,k) - HS_Y(i-1,j,k)) &
&                                                     - dt_edy * (HS_X(i,j,k) - HS_X(i,j-1,k))
!                               Perfect Conductor
                                ELSE IF(Mat==1)THEN
                                        ES_Z(i,j,k) = - ES_Z(i,j,k)
!                               Lossy Material
                                ELSE
```

```
                                        ES_Z(i,j,k) = e_esdt(Mat)*ES_Z(i,j,k) &
&                                                   - dtee_esdt(Mat) * DEI_Y(i,j,k) &
&                                                   - dts_esdt(Mat)  * EI_Y(i,j,k) &
&                                                   + dt_esdtdx(Mat) * ( HS_Y(i,j,k) - HS_Y(i-1,j,k) ) &
&                                                   - dt_esdtdy(Mat) * ( HS_X(i,j,k) - HS_X(i,j-1,k) )
                                END IF
                        END DO
                END DO
        END DO

        RETURN
END
```

C.6.4. H_x Update

Listing 15: FDTD - CalcHX

```
!#####################################################
!Calculation of the x-component of the magnetic field
!#####################################################
SUBROUTINE Calc_HX_Components

        USE globals
        INTEGER :: i,j,k,Mat


        DO k=1, max_z-1
                DO j=1, max_y-1
                        DO i=2, max_x-1
                                HS_X(i,j,k) = HS_X(i,j,k) &
&                                           - dt_mdy * (ES_Z(i,j+1,k) - ES_Z(i,j,k)) &
&                                           + dt_mdz * (ES_Y(i,j,k+1) - ES_Y(i,j,k))
                        END DO
                END DO
        END DO

        RETURN
END
```

C.6.5. H_y Update

Listing 16: FDTD - CalcHY

```
!#####################################################
!Calculation of the y-component of the magnetic field
!#####################################################
SUBROUTINE Calc_HY_Components

        USE globals
        INTEGER :: i,j,k,Mat
```

```
        DO k=1, max_z-1
                DO j=2, max_y-1
                        DO i=1, max_x-1
                                HS_Y(i,j,k) = HS_Y(i,j,k)&
&                                           - dt_mdz * (ES_X(i,j,k+1) - ES_X(i,j,k)) &
&                                           + dt_mdx * (ES_Z(i+1,j,k) - ES_Z(i,j,k))
                        END DO
                END DO
        END DO

        RETURN
END
```

C.6.6. H_z Update

Listing 17: FDTD - CalcHZ

```
!#####################################################
!Calculation of the z-component of the magnetic field
!#####################################################
SUBROUTINE Calc_HZ_Components

        USE globals
        INTEGER :: i,j,k,Mat


        DO k=2, max_z-1
                DO j=1, max_y-1
                        DO i=1, max_x-1
                                HS_Z(i,j,k) = HS_Z(i,j,k)&
&                                           - dt_mdx * (ES_Y(i+1,j,k) - ES_Y(i,j,k)) &
&                                           + dt_mdy * (ES_X(i,j+1,k) - ES_X(i,j,k))
                        END DO
                END DO
        END DO

        RETURN
END
```

C.7. Quellen injezieren

Listing 18: FDTD - SourceInjection

```
!###################
!Insert Sources here
!###################
SUBROUTINE InjectSources

        USE globals
        REAL            :: pulse
        INTEGER         :: Type_Pos=1,X_Pos=2,Y_Pos=3,Z_Pos=4
        INTEGER         :: EX_AmpPos=5, EY_AmpPos=6, EZ_AmpPos=7
        INTEGER         :: I,J,K



!       ------------------------------
!       impose the current source value
!       ------------------------------
        DO source = 1, NoOfSources
                I=Sources(source,X_Pos)
                J=Sources(source,Y_Pos)
                K=Sources(source,Z_Pos)

                IF (Sources(source,Type_Pos) == 1) THEN
                        ES_X(I,J,K) = &
                                Sources(source,EX_AmpPos) * exp(-alpha * ((time - beta_dt)**2))
                        ES_Y(I,J,K) = &
                                Sources(source,EY_AmpPos) * exp(-alpha * ((time - beta_dt)**2))
                        ES_Z(I,J,K) = &
                                Sources(source,EZ_AmpPos) * exp(-alpha * ((time - beta_dt)**2))

                ELSE IF (Sources(source,Type_Pos) == 2) THEN
                        ES_X(I,J,K) = &
                                Sources(source,EX_AmpPos) * sin(omega*time)
                        ES_Y(I,J,K) = &
                                Sources(source,EY_AmpPos) * sin(omega*time)
                        ES_Z(I,J,K) = &
                                Sources(source,EZ_AmpPos) * sin(omega*time)
                END IF

        END DO

        RETURN
END
```

C.8. Absorbierende Randbedingung

Listing 19: FDTD - ABC

```
!############################
!Calculate the boundary values
!############################
SUBROUTINE ABC_Correction

        USE globals

        IF(ABC_Type == 1) THEN
                CALL FirstOrderMUR
        ELSE IF (ABC_Type == 2) THEN
                CALL SecondOrderMur
        END IF

        RETURN
END
```

C.8.1. First Order Mur

Listing 20: FDTD - FirstOrderMur

```
!###################
!first order Mur ABC
!###################
SUBROUTINE FirstOrderMUR
        USE globals
        INTEGER                     :: i,j
        INTEGER, PARAMETER          :: FIRST=1,SECOND=2,PRELAST=3,LAST=4

        DO j=1, max_y
                DO i=1, max_x
                        ES_X(i,j,1)      = PRE_ESX_XY(i,j,SECOND) &
&                                        + cdtdz_cdtdz * ( ES_X(i,j,2) &
&                                        - PRE_ESX_XY(i,j,FIRST) )
                        ES_X(i,j,max_z) = PRE_ESX_XY(i,j,PRELAST) &
&                                        + cdtdz_cdtdz * ( ES_X(i,j,max_z-1) &
&                                        - PRE_ESX_XY(i,j,LAST) )

                        ES_Y(i,j,1)      = PRE_ESY_XY(i,j,SECOND) &
&                                        + cdtdz_cdtdz * ( ES_Y(i,j,2) &
&                                        - PRE_ESY_XY(i,j,FIRST) )
                        ES_Y(i,j,max_z) = PRE_ESY_XY(i,j,PRELAST) &
&                                        + cdtdz_cdtdz * ( ES_Y(i,j,max_z-1) &
&                                        - PRE_ESY_XY(i,j,LAST) )
                        PRE_ESX_XY(i,j,FIRST)    = ES_X(i,j,1)
                        PRE_ESX_XY(i,j,SECOND)   = ES_X(i,j,2)
                        PRE_ESX_XY(i,j,PRELAST) = ES_X(i,j,max_z-1)
                        PRE_ESX_XY(i,j,LAST)     = ES_X(i,j,max_z)
                        PRE_ESY_XY(i,j,FIRST)    = ES_Y(i,j,1)
                        PRE_ESY_XY(i,j,SECOND)   = ES_Y(i,j,2)
```

```
                        PRE_ESY_XY(i,j,PRELAST) = ES_Y(i,j,max_z-1)
                        PRE_ESY_XY(i,j,LAST)    = ES_Y(i,j,max_z)
                END DO
        END DO
        DO k=1, max_z
                DO i=1, max_x
                        ES_X(i,1,k)      = PRE_ESX_XZ(i,SECOND,k) &
&                                        + cdtdy_cdtdy * ( ES_X(i,2,k) &
&                                        - PRE_ESX_XZ(i,FIRST,k) )
                        ES_X(i,max_y,k) = PRE_ESX_XZ(i,PRELAST,k) &
&                                        + cdtdy_cdtdy * ( ES_X(i,max_y-1,k) &
&                                        - PRE_ESX_XZ(i,LAST,k) )
                        ES_Z(i,1,k)      = PRE_ESZ_XZ(i,SECOND,k) &
&                                        + cdtdy_cdtdy * ( ES_Z(i,2,k) &
&                                        - PRE_ESZ_XZ(i,FIRST,k) )
                        ES_Z(i,max_y,k) = PRE_ESZ_XZ(i,PRELAST,k) &
&                                        + cdtdy_cdtdy * ( ES_Z(i,max_y-1,k) &
&                                        - PRE_ESZ_XZ(i,LAST,k) )

                        PRE_ESX_XZ(i,FIRST,k)    = ES_X(i,1,k)
                        PRE_ESX_XZ(i,SECOND,k)   = ES_X(i,2,k)
                        PRE_ESX_XZ(i,PRELAST,k) = ES_X(i,max_y-1,k)
                        PRE_ESX_XZ(i,LAST,k)     = ES_X(i,max_y,k)
                        PRE_ESZ_XZ(i,FIRST,k)    = ES_Z(i,1,k)
                        PRE_ESZ_XZ(i,SECOND,k)   = ES_Z(i,2,k)
                        PRE_ESZ_XZ(i,PRELAST,k) = ES_Z(i,max_y-1,k)
                        PRE_ESZ_XZ(i,LAST,k)     = ES_Z(i,max_y,k)
                END DO
        END DO
        DO k=1, max_z
                DO j=1, max_y
                        ES_Y(1,j,k)      = PRE_ESY_YZ(SECOND,j,k) &
&                                        + cdtdx_cdtdx * ( ES_Y(2,j,k) &
&                                        - PRE_ESY_YZ(FIRST,j,k) )
                        ES_Y(max_x,j,k) = PRE_ESY_YZ(PRELAST,j,k) &
&                                        + cdtdx_cdtdx * ( ES_Y(max_x-1,j,k) &
&                                        - PRE_ESY_YZ(LAST,j,k) )
                        ES_Z(1,j,k)      = PRE_ESZ_YZ(SECOND,j,k) &
&                                        + cdtdx_cdtdx * ( ES_Z(2,j,k) &
&                                        - PRE_ESZ_YZ(FIRST,j,k) )
                        ES_Z(max_x,j,k) = PRE_ESZ_YZ(PRELAST,j,k) &
                                         + cdtdx_cdtdx * ( ES_Z(max_x-1,j,k) &
&                                        - PRE_ESZ_YZ(LAST,j,k) )

                        PRE_ESY_YZ(FIRST,j,k)    = ES_Y(1,j,k)
                        PRE_ESY_YZ(SECOND,j,k)   = ES_Y(2,j,k)
                        PRE_ESY_YZ(PRELAST,j,k) = ES_Y(max_x-1,j,k)
                        PRE_ESY_YZ(LAST,j,k)     = ES_Y(max_x,j,k)
                        PRE_ESZ_YZ(FIRST,j,k)    = ES_Z(1,j,k)
                        PRE_ESZ_YZ(SECOND,j,k)   = ES_Z(2,j,k)
                        PRE_ESZ_YZ(PRELAST,j,k) = ES_Z(max_x-1,j,k)
                        PRE_ESZ_YZ(LAST,j,k)     = ES_Z(max_x,j,k)
                END DO
        END DO
        RETURN
END
```

C.8.2. Second Order Mur

Listing 21: FDTD - SecondOrderMur

```
!####################
!second order Mur ABC
!####################
SUBROUTINE SecondOrderMUR

        CALL MUR_XY
        CALL MUR_XZ
        CALL MUR_YZ

        RETURN
END


!#############################
!second order Mur ABC XY-plane
!#############################
SUBROUTINE MUR_XY

        USE globals

        INTEGER                 :: i,j,k,left_x,right_x,left_y,right_y
        INTEGER, PARAMETER      :: FIRST=1,SECOND=2,PRELAST=3,LAST=4


!       X-Component
!       ###########

!       ---------------------------------------
!       Calculate edges with first order MUR
!       ---------------------------------------

        left_y=2
        right_y=max_y-1
        DO i=2, max_x-2
!               z=1
                ES_X(i,left_y,1)        = PRE_ESX_XY(i,left_y,SECOND) &
&                                       + cdtdz_cdtdz * ( ES_X(i,left_y,2)  &
&                                       - PRE_ESX_XY(i,left_y,FIRST) )
                ES_X(i,right_y,1)       = PRE_ESX_XY(i,right_y,SECOND) &
&                                       + cdtdz_cdtdz * ( ES_X(i,right_y,2)  &
&                                       - PRE_ESX_XY(i,right_y,FIRST) )
!               z=max_z
                ES_X(i,left_y,max_z)    = PRE_ESX_XY(i,left_y,PRELAST) &
&                                       + cdtdz_cdtdz * ( ES_X(i,left_y,max_z-1)  &
&                                       - PRE_ESX_XY(i,left_y,LAST) )
                ES_X(i,right_y,max_z)   = PRE_ESX_XY(i,right_y,PRELAST) &
&                                       + cdtdz_cdtdz * ( ES_X(i,right_y,max_z-1)  &
&                                       - PRE_ESX_XY(i,right_y,LAST) )
        END DO

        left_x=1
        right_x=max_x-1
```

```
        DO j=2, max_y-1
!               z=1
                ES_X(left_x,j,1)        = PRE_ESX_XY(left_x,j,SECOND) &
&                                       + cdtdz_cdtdz * ( ES_X(left_x,j,2)  &
&                                       - PRE_ESX_XY(left_x,j,FIRST) )
                ES_X(right_x,j,1)       = PRE_ESX_XY(right_x,j,SECOND) &
&                                       + cdtdz_cdtdz * ( ES_X(right_x,j,2)  &
&                                       - PRE_ESX_XY(right_x,j,FIRST) )
!               z=max_z
                ES_X(left_x,j,max_z)    = PRE_ESX_XY(left_x,j,PRELAST) &
&                                       + cdtdz_cdtdz * ( ES_X(left_x,j,max_z-1)  &
&                                       - PRE_ESX_XY(left_x,j,LAST) )
                ES_X(right_x,j,max_z)   = PRE_ESX_XY(right_x,j,PRELAST) &
&                                       + cdtdz_cdtdz * ( ES_X(right_x,j,max_z-1)  &
&                                       - PRE_ESX_XY(right_x,j,LAST) )
        END DO


!       ------------------------------------------------------------
!       Calculate areas between the edges with second order MUR
!       ------------------------------------------------------------

        DO j=3, max_y-2
                DO i=2, max_x-2
!                       area at z=1
                        ES_X(i,j,1) =   - PREPRE_ESX_XY(i,j,SECOND) &
&                                       + cdtdz_cdtdz   * ( ES_X(i,j,2) &
&                                                         + PREPRE_ESX_XY(i,j,FIRST) )&
&                                       + dz_cdtdz      * ( PRE_ESX_XY(i,j,FIRST)  &
&                                                         + PRE_ESX_XY(i,j,SECOND) )&
&                                       + dzcdt_2dxcdtdz* ( PRE_ESX_XY(i+1,j,FIRST)      &
&                                                         - 2.*PRE_ESX_XY(i,j,FIRST) &
&                                                         + PRE_ESX_XY(i-1,j,FIRST) &
&                                                         + PRE_ESX_XY(i+1,j,SECOND) &
&                                                         - 2.*PRE_ESX_XY(i,j,SECOND) &
&                                                         + PRE_ESX_XY(i-1,j,SECOND) ) &
&                                       + dzcdt_2dycdtdz* ( PRE_ESX_XY(i,j+1,FIRST)      &
&                                                         - 2.*PRE_ESX_XY(i,j,FIRST) &
&                                                         + PRE_ESX_XY(i,j-1,FIRST) &
&                                                         + PRE_ESX_XY(i,j+1,SECOND) &
&                                                         - 2.*PRE_ESX_XY(i,j,SECOND) &
&                                                         + PRE_ESX_XY(i,j-1,SECOND) )
!                       area at z=max_z
                        ES_X(i,j,max_z) = - PREPRE_ESX_XY(i,j,PRELAST) &
&                                       + cdtdz_cdtdz   * ( ES_X(i,j,max_z-1) &
&                                                         + PREPRE_ESX_XY(i,j,LAST) )&
&                                       + dz_cdtdz      * ( PRE_ESX_XY(i,j,LAST) &
&                                                         + PRE_ESX_XY(i,j,PRELAST) )&
&                                       + dzcdt_2dxcdtdz* ( PRE_ESX_XY(i+1,j,LAST)       &
&                                                         - 2.*PRE_ESX_XY(i,j,LAST) &
&                                                         + PRE_ESX_XY(i-1,j,LAST) &
&                                                         + PRE_ESX_XY(i+1,j,PRELAST) &
&                                                         - 2.*PRE_ESX_XY(i,j,PRELAST) &
&                                                         + PRE_ESX_XY(i-1,j,PRELAST) ) &
&                                       + dzcdt_2dycdtdz* ( PRE_ESX_XY(i,j+1,LAST)       &
&                                                         - 2.*PRE_ESX_XY(i,j,LAST) &
&                                                         + PRE_ESX_XY(i,j-1,LAST) &
```

```
&                                                                    + PRE_ESX_XY(i,j+1,PRELAST) &
&                                                                    - 2.*PRE_ESX_XY(i,j,PRELAST) &
&                                                                    + PRE_ESX_XY(i,j-1,PRELAST) )
            END DO
        END DO


!       ---------------------
!       Save previous values
!       ---------------------
        DO j=2,max_y-1
            DO i=1,max_x-1

                PREPRE_ESX_XY(i,j,FIRST)        = PRE_ESX_XY(i,j,FIRST)
                PREPRE_ESX_XY(i,j,SECOND)       = PRE_ESX_XY(i,j,SECOND)
                PREPRE_ESX_XY(i,j,PRELAST)      = PRE_ESX_XY(i,j,PRELAST)
                PREPRE_ESX_XY(i,j,LAST)         = PRE_ESX_XY(i,j,LAST)

                PRE_ESX_XY(i,j,FIRST)           = ES_X(i,j,1)
                PRE_ESX_XY(i,j,SECOND)          = ES_X(i,j,2)
                PRE_ESX_XY(i,j,PRELAST)         = ES_X(i,j,max_z-1)
                PRE_ESX_XY(i,j,LAST)            = ES_X(i,j,max_z)
            END DO
        END DO


!       Y-Component
!       ###########

!       ------------------------------------
!       Calculate edges with first order MUR
!       ------------------------------------

        left_y=1
        right_y=max_y-1
        DO i=3, max_x-2
!           z=1
            ES_Y(i,left_y,1)        = PRE_ESY_XY(i,left_y,SECOND) &
&                                   + cdtdz_cdtdz * ( ES_Y(i,left_y,2) &
&                                   - PRE_ESY_XY(i,left_y,FIRST) )
            ES_Y(i,right_y,1)       = PRE_ESY_XY(i,right_y,SECOND) &
&                                   + cdtdz_cdtdz * ( ES_Y(i,right_y,2) &
&                                   - PRE_ESY_XY(i,right_y,FIRST) )
!           z=max_z
            ES_Y(i,left_y,max_z)    = PRE_ESY_XY(i,left_y,PRELAST) &
&                                   + cdtdz_cdtdz * ( ES_Y(i,left_y,max_z-1) &
&                                   - PRE_ESY_XY(i,left_y,LAST) )
            ES_Y(i,right_y,max_z)   = PRE_ESY_XY(i,right_y,PRELAST) &
&                                   + cdtdz_cdtdz * ( ES_Y(i,right_y,max_z-1) &
&                                   - PRE_ESY_XY(i,right_y,LAST) )
        END DO

        left_x=2
        right_x=max_x-1
        DO j=1, max_y-1
!           z=1
            ES_Y(left_x,j,1)        = PRE_ESY_XY(left_x,j,SECOND) &
```

```
&                                       + cdtdz_cdtdz * ( ES_Y(left_x,j,2) &
&                                       - PRE_ESY_XY(left_x,j,FIRST) )
                ES_Y(right_x,j,1)       = PRE_ESY_XY(right_x,j,SECOND) &
&                                       + cdtdz_cdtdz * ( ES_Y(right_x,j,2) &
&                                       - PRE_ESY_XY(right_x,j,FIRST) )
!               z=max_z
                ES_Y(left_x,j,max_z)    = PRE_ESY_XY(left_x,j,PRELAST) &
&                                       + cdtdz_cdtdz * ( ES_Y(left_x,j,max_z-1) &
&                                       - PRE_ESY_XY(left_x,j,LAST) )
                ES_Y(right_x,j,max_z)   = PRE_ESY_XY(right_x,j,PRELAST) &
&                                       + cdtdz_cdtdz * ( ES_Y(right_x,j,max_z-1) &
&                                       - PRE_ESY_XY(right_x,j,LAST) )
        END DO


!       ------------------------------------------------------
!       Calculate areas between the edges with second order MUR
!       ------------------------------------------------------

        DO j=2, max_y-2
                DO i=3, max_x-2
!                       area at z=1
                        ES_Y(i,j,1) =   - PREPRE_ESY_XY(i,j,SECOND) &
&                                       + cdtdz_cdtdz   * ( ES_Y(i,j,2) &
&                                                         + PREPRE_ESY_XY(i,j,FIRST) )&
&                                       + dz_cdtdz      * ( PRE_ESY_XY(i,j,FIRST) &
&                                                         + PRE_ESY_XY(i,j,SECOND) )&
&                                       + dzcdt_2dxcdtdz* ( PRE_ESY_XY(i+1,j,FIRST)     &
&                                                         - 2.*PRE_ESY_XY(i,j,FIRST) &
&                                                         + PRE_ESY_XY(i-1,j,FIRST) &
&                                                         + PRE_ESY_XY(i+1,j,SECOND) &
&                                                         - 2.*PRE_ESY_XY(i,j,SECOND) &
&                                                         + PRE_ESY_XY(i-1,j,SECOND) ) &
&                                       + dzcdt_2dycdtdz* ( PRE_ESY_XY(i,j+1,FIRST)     &
&                                                         - 2.*PRE_ESY_XY(i,j,FIRST) &
&                                                         + PRE_ESY_XY(i,j-1,FIRST) &
&                                                         + PRE_ESY_XY(i,j+1,SECOND) &
&                                                         - 2.*PRE_ESY_XY(i,j,SECOND) &
&                                                         + PRE_ESY_XY(i,j-1,SECOND) )
!                       area at z=max_z
                        ES_Y(i,j,max_z) = - PREPRE_ESY_XY(i,j,PRELAST) &
&                                       + cdtdz_cdtdz   * ( ES_Y(i,j,max_z-1) &
&                                                         + PREPRE_ESY_XY(i,j,LAST) )&
&                                       + dz_cdtdz      * ( PRE_ESY_XY(i,j,LAST) &
&                                                         + PRE_ESY_XY(i,j,PRELAST) )&
&                                       + dzcdt_2dxcdtdz* ( PRE_ESY_XY(i+1,j,LAST)      &
&                                                         - 2.*PRE_ESY_XY(i,j,LAST) &
&                                                         + PRE_ESY_XY(i-1,j,LAST) &
&                                                         + PRE_ESY_XY(i+1,j,PRELAST) &
&                                                         - 2.*PRE_ESY_XY(i,j,PRELAST) &
&                                                         + PRE_ESY_XY(i-1,j,PRELAST) ) &
&                                       + dzcdt_2dycdtdz* ( PRE_ESY_XY(i,j+1,LAST)      &
&                                                         - 2.*PRE_ESY_XY(i,j,LAST) &
&                                                         + PRE_ESY_XY(i,j-1,LAST) &
&                                                         + PRE_ESY_XY(i,j+1,PRELAST) &
&                                                         - 2.*PRE_ESY_XY(i,j,PRELAST) &
&                                                         + PRE_ESY_XY(i,j-1,PRELAST) )
```

```
                END DO
        END DO


!       ---------------------
!       Save previous values
!       ---------------------
        DO j=1,max_y-1
                DO i=2,max_x-1

                        PREPRE_ESY_XY(i,j,FIRST)        = PRE_ESY_XY(i,j,FIRST)
                        PREPRE_ESY_XY(i,j,SECOND)       = PRE_ESY_XY(i,j,SECOND)
                        PREPRE_ESY_XY(i,j,PRELAST)      = PRE_ESY_XY(i,j,PRELAST)
                        PREPRE_ESY_XY(i,j,LAST)         = PRE_ESY_XY(i,j,LAST)

                        PRE_ESY_XY(i,j,FIRST)           = ES_Y(i,j,1)
                        PRE_ESY_XY(i,j,SECOND)          = ES_Y(i,j,2)
                        PRE_ESY_XY(i,j,PRELAST)         = ES_Y(i,j,max_z-1)
                        PRE_ESY_XY(i,j,LAST)            = ES_Y(i,j,max_z)
                END DO
        END DO


        RETURN
END


!##############################
!second order Mur ABC XZ-plane
!##############################
SUBROUTINE MUR_XZ

        USE globals

        INTEGER                 :: i,j,k,left_x,right_x,left_z,right_z
        INTEGER, PARAMETER      :: FIRST=1,SECOND=2,PRELAST=3,LAST=4


!       X-Component
!       ###########

!       ---------------------------------------
!       Calculate edges with first order MUR
!       ---------------------------------------

        left_z=2
        right_z=max_z-1
        DO i=2, max_x-2
!               y=1
                ES_X(i,1,left_z)        = PRE_ESX_XZ(i,SECOND,left_z) &
&                                       + cdtdy_cdtdy * ( ES_X(i,2,left_z) &
&                                       - PRE_ESX_XZ(i,FIRST,left_z) )
                ES_X(i,1,right_z)       = PRE_ESX_XZ(i,SECOND,right_z) &
&                                       + cdtdy_cdtdy * ( ES_X(i,2,right_z) &
&                                       - PRE_ESX_XZ(i,FIRST,right_z) )
!               y=max_y
                ES_X(i,max_y,left_z)    = PRE_ESX_XZ(i,PRELAST,left_z) &
```

```
&                                       + cdtdy_cdtdy * ( ES_X(i,max_y-1,left_z) &
&                                       - PRE_ESX_XZ(i,LAST,left_z) )
                ES_X(i,max_y,right_z)   = PRE_ESX_XZ(i,PRELAST,right_z) &
&                                       + cdtdy_cdtdy * ( ES_X(i,max_y-1,right_z) &
&                                       - PRE_ESX_XZ(i,LAST,right_z) )
        END DO

        left_x=1
        right_x=max_x-1
        DO k=2, max_z-1
!               y=1
                ES_X(left_x,1,k)        = PRE_ESX_XZ(left_x,SECOND,k) &
&                                       + cdtdy_cdtdy * ( ES_X(left_x,2,k) &
&                                       - PRE_ESX_XZ(left_x,FIRST,k) )
                ES_X(right_x,1,k)       = PRE_ESX_XZ(right_x,SECOND,k) &
&                                       + cdtdy_cdtdy * ( ES_X(right_x,2,k) &
&                                       - PRE_ESX_XZ(right_x,FIRST,k) )
!               y=max_y
                ES_X(left_x,max_y,k)    = PRE_ESX_XZ(left_x,PRELAST,k) &
&                                       + cdtdy_cdtdy * ( ES_X(left_x,max_y-1,k) &
&                                       - PRE_ESX_XZ(left_x,LAST,k) )
                ES_X(right_x,max_y,k)   = PRE_ESX_XZ(right_x,PRELAST,k) &
&                                       + cdtdy_cdtdy * ( ES_X(right_x,max_y-1,k) &
&                                       - PRE_ESX_XZ(right_x,LAST,k) )
        END DO


!       ------------------------------------------------------
!       Calculate areas between the edges with second order MUR
!       ------------------------------------------------------

        DO k=3, max_z-2
                DO i=2, max_x-2
!                       area at y=1
                        ES_X(i,1,k) =   - PREPRE_ESX_XZ(i,SECOND,k) &
&                                       + cdtdy_cdtdy   * ( ES_X(i,2,k) &
&                                                         + PREPRE_ESX_XZ(i,FIRST,k) )&
&                                       + dy_cdtdy      * ( PRE_ESX_XZ(i,FIRST,k) &
&                                                         + PRE_ESX_XZ(i,SECOND,k) )&
&                                       + dycdt_2dxcdtdy* ( PRE_ESX_XZ(i+1,FIRST,k)     &
&                                                         - 2.*PRE_ESX_XZ(i,FIRST,k) &
&                                                         + PRE_ESX_XZ(i-1,FIRST,k) &
&                                                         + PRE_ESX_XZ(i+1,SECOND,k) &
&                                                         - 2.*PRE_ESX_XZ(i,SECOND,k) &
&                                                         + PRE_ESX_XZ(i-1,SECOND,k) ) &
&                                       + dycdt_2dzcdtdy* ( PRE_ESX_XZ(i,FIRST,k+1)     &
&                                                         - 2.*PRE_ESX_XZ(i,FIRST,k) &
&                                                         + PRE_ESX_XZ(i,FIRST,k-1) &
&                                                         + PRE_ESX_XZ(i,SECOND,k+1) &
&                                                         - 2.*PRE_ESX_XZ(i,SECOND,k) &
&                                                         + PRE_ESX_XZ(i,SECOND,k-1) )
!                       area at y=max_y
                        ES_X(i,max_y,k) = - PREPRE_ESX_XZ(i,PRELAST,k) &
&                                       + cdtdy_cdtdy   * ( ES_X(i,max_y-1,k) &
&                                                         + PREPRE_ESX_XZ(i,LAST,k) )&
&                                       + dy_cdtdy      * ( PRE_ESX_XZ(i,LAST,k) &
&                                                         + PRE_ESX_XZ(i,PRELAST,k) )&
```

```
&                                   + dycdt_2dxcdtdy* ( PRE_ESX_XZ(i+1,LAST,k)      &
&                                                     - 2.*PRE_ESX_XZ(i,LAST,k) &
&                                                     + PRE_ESX_XZ(i-1,LAST,k) &
&                                                     + PRE_ESX_XZ(i+1,PRELAST,k) &
&                                                     - 2.*PRE_ESX_XZ(i,PRELAST,k) &
&                                                     + PRE_ESX_XZ(i-1,PRELAST,k) ) &
&                                   + dycdt_2dzcdtdy* ( PRE_ESX_XZ(i,LAST,k+1)      &
&                                                     - 2.*PRE_ESX_XZ(i,LAST,k) &
&                                                     + PRE_ESX_XZ(i,LAST,k-1) &
&                                                     + PRE_ESX_XZ(i,PRELAST,k+1) &
&                                                     - 2.*PRE_ESX_XZ(i,PRELAST,k) &
&                                                     + PRE_ESX_XZ(i,PRELAST,k-1) )
            END DO
        END DO

!       ---------------------
!       Save previous values
!       ---------------------
        DO k=2,max_z-1
            DO i=1,max_x-1

                PREPRE_ESX_XZ(i,FIRST,k)      = PRE_ESX_XZ(i,FIRST,k)
                PREPRE_ESX_XZ(i,SECOND,k)     = PRE_ESX_XZ(i,SECOND,k)
                PREPRE_ESX_XZ(i,PRELAST,k)    = PRE_ESX_XZ(i,PRELAST,k)
                PREPRE_ESX_XZ(i,LAST,k)       = PRE_ESX_XZ(i,LAST,k)

                PRE_ESX_XZ(i,FIRST,k)         = ES_X(i,1,k)
                PRE_ESX_XZ(i,SECOND,k)        = ES_X(i,2,k)
                PRE_ESX_XZ(i,PRELAST,k)       = ES_X(i,max_y-1,k)
                PRE_ESX_XZ(i,LAST,k)          = ES_X(i,max_y,k)
            END DO
        END DO



!       Z-Component
!       ###########

!       ---------------------------------
!       Calculate edges with first order MUR
!       ---------------------------------

        left_z=1
        right_z=max_z-1
        DO i=3, max_x-2
!           y=1
            ES_Z(i,1,left_z)        = PRE_ESZ_XZ(i,SECOND,left_z) &
&                                   + cdtdy_cdtdy * ( ES_Z(i,2,left_z) &
&                                   - PRE_ESZ_XZ(i,FIRST,left_z) )
            ES_Z(i,1,right_z)       = PRE_ESZ_XZ(i,SECOND,right_z) &
&                                   + cdtdy_cdtdy * ( ES_Z(i,2,right_z) &
&                                   - PRE_ESZ_XZ(i,FIRST,right_z) )
!           y=max_y
            ES_Z(i,max_y,left_z)    = PRE_ESZ_XZ(i,PRELAST,left_z) &
&                                   + cdtdy_cdtdy * ( ES_Z(i,max_y-1,left_z) &
&                                   - PRE_ESZ_XZ(i,LAST,left_z) )
            ES_Z(i,max_y,right_z)   = PRE_ESZ_XZ(i,PRELAST,right_z) &
```

```
&                                        + cdtdy_cdtdy * ( ES_Z(i,max_y-1,right_z) &
&                                        - PRE_ESZ_XZ(i,LAST,right_z) )
        END DO


        left_x=2
        right_x=max_x-1
        DO k=1, max_z-1
!               y=1
                ES_Z(left_x,1,k)         = PRE_ESZ_XZ(left_x,SECOND,k) &
&                                        + cdtdy_cdtdy * ( ES_Z(left_x,2,k) &
&                                        - PRE_ESZ_XZ(left_x,FIRST,k) )
                ES_Z(right_x,1,k)        = PRE_ESZ_XZ(right_x,SECOND,k) &
&                                        + cdtdy_cdtdy * ( ES_Z(right_x,2,k) &
&                                        - PRE_ESZ_XZ(right_x,FIRST,k) )
!               y=max_y
                ES_Z(left_x,max_y,k)     = PRE_ESZ_XZ(left_x,PRELAST,k) &
&                                        + cdtdy_cdtdy * ( ES_Z(left_x,max_y-1,k) &
&                                        - PRE_ESZ_XZ(left_x,LAST,k) )
                ES_Z(right_x,max_y,k)    = PRE_ESZ_XZ(right_x,PRELAST,k) &
&                                        + cdtdy_cdtdy * ( ES_Z(right_x,max_y-1,k) &
&                                        - PRE_ESZ_XZ(right_x,LAST,k) )
        END DO


!       ------------------------------------------------------
!       Calculate areas between the edges with second order MUR
!       ------------------------------------------------------

        DO k=2, max_z-2
                DO i=3, max_x-2
!                       area at y=1
                        ES_Z(i,1,k) =    - PREPRE_ESZ_XZ(i,SECOND,k) &
&                                        + cdtdy_cdtdy    * ( ES_Z(i,2,k) &
&                                                           + PREPRE_ESZ_XZ(i,FIRST,k) )&
&                                        + dy_cdtdy       * ( PRE_ESZ_XZ(i,FIRST,k) &
&                                                           + PRE_ESZ_XZ(i,SECOND,k) )&
&                                        + dycdt_2dxcdtdy* ( PRE_ESZ_XZ(i+1,FIRST,k)      &
&                                                           - 2.*PRE_ESZ_XZ(i,FIRST,k) &
&                                                           + PRE_ESZ_XZ(i-1,FIRST,k) &
&                                                           + PRE_ESZ_XZ(i+1,SECOND,k) &
&                                                           - 2.*PRE_ESZ_XZ(i,SECOND,k) &
&                                                           + PRE_ESZ_XZ(i-1,SECOND,k) ) &
&                                        + dycdt_2dzcdtdy* ( PRE_ESZ_XZ(i,FIRST,k+1)      &
&                                                           - 2.*PRE_ESZ_XZ(i,FIRST,k) &
&                                                           + PRE_ESZ_XZ(i,FIRST,k-1) &
&                                                           + PRE_ESZ_XZ(i,SECOND,k+1) &
&                                                           - 2.*PRE_ESZ_XZ(i,SECOND,k) &
&                                                           + PRE_ESZ_XZ(i,SECOND,k-1) )
!                       area at y=max_y
                        ES_Z(i,max_y,k) = - PREPRE_ESZ_XZ(i,PRELAST,k) &
&                                        + cdtdy_cdtdy    * ( ES_Z(i,max_y-1,k) &
&                                                           + PREPRE_ESZ_XZ(i,LAST,k) )&
&                                        + dy_cdtdy       * ( PRE_ESZ_XZ(i,LAST,k) &
&                                                           + PRE_ESZ_XZ(i,PRELAST,k) )&
&                                        + dycdt_2dxcdtdy* ( PRE_ESZ_XZ(i+1,LAST,k)       &
&                                                           - 2.*PRE_ESZ_XZ(i,LAST,k) &
```

```
&                                                                    + PRE_ESZ_XZ(i-1,LAST,k) &
&                                                                    + PRE_ESZ_XZ(i+1,PRELAST,k) &
&                                                                    - 2.*PRE_ESZ_XZ(i,PRELAST,k) &
&                                                                    + PRE_ESZ_XZ(i-1,PRELAST,k) ) &
&                                                 + dycdt_2dzcdtdy* ( PRE_ESZ_XZ(i,LAST,k+1)       &
&                                                                    - 2.*PRE_ESZ_XZ(i,LAST,k) &
&                                                                    + PRE_ESZ_XZ(i,LAST,k-1) &
&                                                                    + PRE_ESZ_XZ(i,PRELAST,k+1) &
&                                                                    - 2.*PRE_ESZ_XZ(i,PRELAST,k) &
&                                                                    + PRE_ESZ_XZ(i,PRELAST,k-1) )
            END DO
        END DO

!       ---------------------
!       Save previous values
!       ---------------------
        DO k=1,max_z-1
            DO i=2,max_x-1
                PREPRE_ESZ_XZ(i,FIRST,k)     = PRE_ESZ_XZ(i,FIRST,k)
                PREPRE_ESZ_XZ(i,SECOND,k)    = PRE_ESZ_XZ(i,SECOND,k)
                PREPRE_ESZ_XZ(i,PRELAST,k)   = PRE_ESZ_XZ(i,PRELAST,k)
                PREPRE_ESZ_XZ(i,LAST,k)      = PRE_ESZ_XZ(i,LAST,k)

                PRE_ESZ_XZ(i,FIRST,k)        = ES_Z(i,1,k)
                PRE_ESZ_XZ(i,SECOND,k)       = ES_Z(i,2,k)
                PRE_ESZ_XZ(i,PRELAST,k)      = ES_Z(i,max_y-1,k)
                PRE_ESZ_XZ(i,LAST,k)         = ES_Z(i,max_y,k)
            END DO
        END DO


        RETURN
END


!##############################
!second order Mur ABC YZ-plane
!##############################
SUBROUTINE MUR_YZ

        USE globals

        INTEGER                 :: i,j,k,left_y,right_y,left_z,right_z
        INTEGER, PARAMETER      :: FIRST=1,SECOND=2,PRELAST=3,LAST=4



!       WRITE(*,*) cdtdz_cdtdz, dz_cdtdz, dzcdt_2dxcdtdz, dzcdt_2dycdtdz



!       Y-Component
!       ###########

!       ---------------------------------
!       Calculate edges with first order MUR
!       ---------------------------------
```

```
        left_z=2
        right_z=max_z-1
        DO j=2, max_y-2
!               x=1
                ES_Y(1,j,left_z)        = PRE_ESY_YZ(SECOND,j,left_z) &
&                                       + cdtdx_cdtdx * ( ES_Y(2,j,left_z) &
&                                       - PRE_ESY_YZ(FIRST,j,left_z) )
                ES_Y(1,j,right_z)       = PRE_ESY_YZ(SECOND,j,right_z) &
&                                       + cdtdx_cdtdx * ( ES_Y(2,j,right_z) &
&                                       - PRE_ESY_YZ(FIRST,j,right_z) )
!               x=max_x
                ES_Y(max_x,j,left_z)    = PRE_ESY_YZ(PRELAST,j,left_z) &
&                                       + cdtdx_cdtdx * ( ES_Y(max_x-1,j,left_z) &
&                                       - PRE_ESY_YZ(LAST,j,left_z) )
                ES_Y(max_x,j,right_z)   = PRE_ESY_YZ(PRELAST,j,right_z) &
&                                       + cdtdx_cdtdx * ( ES_Y(max_x-1,j,right_z) &
&                                       - PRE_ESY_YZ(LAST,j,right_z) )
        END DO

        left_y=1
        right_y=max_y-1
        DO k=2, max_z-1
!               x=1
                ES_Y(1,left_y,k)        = PRE_ESY_YZ(SECOND,left_y,k) &
&                                       + cdtdx_cdtdx * ( ES_Y(2,left_y,k) &
&                                       - PRE_ESY_YZ(FIRST,left_y,k) )
                ES_Y(1,right_y,k)       = PRE_ESY_YZ(SECOND,right_y,k) &
&                                       + cdtdx_cdtdx * ( ES_Y(2,right_y,k) &
&                                       - PRE_ESY_YZ(FIRST,right_y,k) )
!               x=max_x
                ES_Y(max_x,left_y,k)    = PRE_ESY_YZ(PRELAST,left_y,k) &
&                                       + cdtdx_cdtdx * ( ES_Y(max_x-1,left_y,k) &
&                                       - PRE_ESY_YZ(LAST,left_y,k) )
                ES_Y(max_x,right_y,k)   = PRE_ESY_YZ(PRELAST,right_y,k) &
&                                       + cdtdx_cdtdx * ( ES_Y(max_x-1,right_y,k) &
&                                       - PRE_ESY_YZ(LAST,right_y,k) )
        END DO


!       ------------------------------------------------------------
!       Calculate areas between the edges with second order MUR
!       ------------------------------------------------------------

        DO k=3, max_z-2
                DO j=2, max_y-2
!                       area at x=1
                        ES_Y(1,j,k) =   - PREPRE_ESY_YZ(SECOND,j,k) &
&                                       + cdtdx_cdtdx    * ( ES_Y(2,j,k) &
&                                                          + PREPRE_ESY_YZ(FIRST,j,k) )&
&                                       + dx_cdtdx       * ( PRE_ESY_YZ(FIRST,j,k) &
&                                                          + PRE_ESY_YZ(SECOND,j,k) )&
&                                       + dxcdt_2dycdtdx* ( PRE_ESY_YZ(FIRST,j+1,k)      &
&                                                          - 2.*PRE_ESY_YZ(FIRST,j,k) &
&                                                          + PRE_ESY_YZ(FIRST,j-1,k) &
&                                                          + PRE_ESY_YZ(SECOND,j+1,k) &
&                                                          - 2.*PRE_ESY_YZ(SECOND,j,k) &
```

```
&                                                                + PRE_ESY_YZ(SECOND,j-1,k) ) &
&                                    + dxcdt_2dzcdtdx* ( PRE_ESY_YZ(FIRST,j,k+1)       &
&                                                                - 2.*PRE_ESY_YZ(FIRST,j,k) &
&                                                                + PRE_ESY_YZ(FIRST,j,k-1) &
&                                                                + PRE_ESY_YZ(SECOND,j,k+1) &
&                                                                - 2.*PRE_ESY_YZ(SECOND,j,k) &
&                                                                + PRE_ESY_YZ(SECOND,j,k-1) )
!                          area at x=max_x
                           ES_Y(max_x,j,k) = - PREPRE_ESY_YZ(PRELAST,j,k)&
&                                    + cdtdx_cdtdx    * ( ES_Y(max_x-1,j,k) &
&                                                                + PREPRE_ESY_YZ(LAST,j,k) )&
&                                    + dx_cdtdx       * ( PRE_ESY_YZ(LAST,j,k) &
&                                                                + PRE_ESY_YZ(PRELAST,j,k) )&
&                                    + dxcdt_2dycdtdx* ( PRE_ESY_YZ(LAST,j+1,k)        &
&                                                                - 2.*PRE_ESY_YZ(LAST,j,k) &
&                                                                + PRE_ESY_YZ(LAST,j-1,k) &
&                                                                + PRE_ESY_YZ(PRELAST,j+1,k) &
&                                                                - 2.*PRE_ESY_YZ(PRELAST,j,k) &
&                                                                + PRE_ESY_YZ(PRELAST,j-1,k) ) &
&                                    + dxcdt_2dzcdtdx* ( PRE_ESY_YZ(LAST,j,k+1)        &
&                                                                - 2.*PRE_ESY_YZ(LAST,j,k) &
&                                                                + PRE_ESY_YZ(LAST,j,k-1) &
&                                                                + PRE_ESY_YZ(PRELAST,j,k+1) &
&                                                                - 2.*PRE_ESY_YZ(PRELAST,j,k) &
&                                                                + PRE_ESY_YZ(PRELAST,j,k-1) )
                  END DO
            END DO

!           ---------------------
!           Save previous values
!           ---------------------
            DO k=2,max_z-1
                  DO j=1,max_y-1

                        PREPRE_ESY_YZ(FIRST,j,k)        = PRE_ESY_YZ(FIRST,j,k)
                        PREPRE_ESY_YZ(SECOND,j,k)       = PRE_ESY_YZ(SECOND,j,k)
                        PREPRE_ESY_YZ(PRELAST,j,k)      = PRE_ESY_YZ(PRELAST,j,k)
                        PREPRE_ESY_YZ(LAST,j,k)         = PRE_ESY_YZ(LAST,j,k)

                        PRE_ESY_YZ(FIRST,j,k)           = ES_Y(1,j,k)
                        PRE_ESY_YZ(SECOND,j,k)          = ES_Y(2,j,k)
                        PRE_ESY_YZ(PRELAST,j,k)         = ES_Y(max_x-1,j,k)
                        PRE_ESY_YZ(LAST,j,k)            = ES_Y(max_x,j,k)
                  END DO
            END DO


!           Z-Component
!           ###########


!           ------------------------------------
!           Calculate edges with first order MUR
!           ------------------------------------

            left_z=1
            right_z=max_z-1
```

```
      DO j=3, max_y-2
!           x=1
            ES_Z(1,j,left_z)      = PRE_ESZ_YZ(SECOND,j,left_z) &
&                                 + cdtdx_cdtdx * ( ES_Z(2,j,left_z) &
&                                 - PRE_ESZ_YZ(FIRST,j,left_z) )
            ES_Z(1,j,right_z)     = PRE_ESZ_YZ(SECOND,j,right_z) &
&                                 + cdtdx_cdtdx * ( ES_Z(2,j,right_z)&
&                                 - PRE_ESZ_YZ(FIRST,j,right_z) )
!           x=max_x
            ES_Z(max_x,j,left_z)  = PRE_ESZ_YZ(PRELAST,j,left_z) &
&                                 + cdtdx_cdtdx * ( ES_Z(max_x-1,j,left_z) &
&                                 - PRE_ESZ_YZ(LAST,j,left_z) )
            ES_Z(max_x,j,right_z) = PRE_ESZ_YZ(PRELAST,j,right_z) &
&                                 + cdtdx_cdtdx * ( ES_Z(max_x-1,j,right_z) &
&                                 - PRE_ESZ_YZ(LAST,j,right_z) )
      END DO

      left_y=2
      right_y=max_y-1
      DO k=1, max_z-1
!           x=1
            ES_Z(1,left_y,k)      = PRE_ESZ_YZ(SECOND,left_y,k) &
&                                 + cdtdx_cdtdx * ( ES_Z(2,left_y,k) &
&                                 - PRE_ESZ_YZ(FIRST,left_y,k) )
            ES_Z(1,right_y,k)     = PRE_ESZ_YZ(SECOND,right_y,k) &
&                                 + cdtdx_cdtdx * ( ES_Z(2,right_y,k)  &
&                                 - PRE_ESZ_YZ(FIRST,right_y,k)  )
!           x=max_x
            ES_Z(max_x,left_y,k)  = PRE_ESZ_YZ(PRELAST,left_y,k) &
&                                 + cdtdx_cdtdx * ( ES_Z(max_x-1,left_y,k) &
&                                 - PRE_ESZ_YZ(LAST,left_y,k) )
            ES_Z(max_x,right_y,k) = PRE_ESZ_YZ(PRELAST,right_y,k) &
&                                 + cdtdx_cdtdx * ( ES_Z(max_x-1,right_y,k)  &
&                                 - PRE_ESZ_YZ(LAST,right_y,k)  )
      END DO


!     ------------------------------------------------------
!     Calculate areas between the edges with second order MUR
!     ------------------------------------------------------

      DO k=2, max_z-2
            DO j=3, max_y-2
!                 area at x=1
                  ES_Z(1,j,k) =   - PREPRE_ESZ_YZ(SECOND,j,k) &
                                  + cdtdx_cdtdx    * ( ES_Z(2,j,k) &
&                                                     + PREPRE_ESZ_YZ(FIRST,j,k) )&
&                                 + dx_cdtdx       * ( PRE_ESZ_YZ(FIRST,j,k) &
&                                                     + PRE_ESZ_YZ(SECOND,j,k) )&
&                                 + dxcdt_2dycdtdx* ( PRE_ESZ_YZ(FIRST,j+1,k)      &
&                                                     - 2.*PRE_ESZ_YZ(FIRST,j,k) &
&                                                     + PRE_ESZ_YZ(FIRST,j-1,k) &
&                                                     + PRE_ESZ_YZ(SECOND,j+1,k) &
&                                                     - 2.*PRE_ESZ_YZ(SECOND,j,k) &
&                                                     + PRE_ESZ_YZ(SECOND,j-1,k) ) &
&                                 + dxcdt_2dzcdtdx* ( PRE_ESZ_YZ(FIRST,j,k+1)      &
&                                                     - 2.*PRE_ESZ_YZ(FIRST,j,k) &
```

```
&                                                                + PRE_ESZ_YZ(FIRST,j,k-1) &
&                                                                + PRE_ESZ_YZ(SECOND,j,k+1) &
&                                                                - 2.*PRE_ESZ_YZ(SECOND,j,k) &
&                                                                + PRE_ESZ_YZ(SECOND,j,k-1) )
!                       area at x=max_x
                        ES_Z(max_x,j,k) = - PREPRE_ESZ_YZ(PRELAST,j,k) &
&                                         + cdtdx_cdtdx    * ( ES_Z(max_x-1,j,k) &
&                                                            + PREPRE_ESZ_YZ(LAST,j,k) )&
&                                         + dx_cdtdx       * ( PRE_ESZ_YZ(LAST,j,k) &
&                                                            + PRE_ESZ_YZ(PRELAST,j,k) )&
&                                         + dxcdt_2dycdtdx* ( PRE_ESZ_YZ(LAST,j+1,k)       &
&                                                            - 2.*PRE_ESZ_YZ(LAST,j,k) &
&                                                            + PRE_ESZ_YZ(LAST,j-1,k) &
&                                                            + PRE_ESZ_YZ(PRELAST,j+1,k) &
&                                                            - 2.*PRE_ESZ_YZ(PRELAST,j,k) &
&                                                            + PRE_ESZ_YZ(PRELAST,j-1,k) ) &
&                                         + dxcdt_2dzcdtdx* ( PRE_ESZ_YZ(LAST,j,k+1)       &
&                                                            - 2.*PRE_ESZ_YZ(LAST,j,k) &
&                                                            + PRE_ESZ_YZ(LAST,j,k-1) &
&                                                            + PRE_ESZ_YZ(PRELAST,j,k+1) &
&                                                            - 2.*PRE_ESZ_YZ(PRELAST,j,k) &
&                                                            + PRE_ESZ_YZ(PRELAST,j,k-1) )
                END DO
        END DO

!       ----------------------
!       Save previous values
!       ----------------------
        DO k=1,max_z-1
                DO j=2,max_y-1

                        PREPRE_ESZ_YZ(FIRST,j,k)        = PRE_ESZ_YZ(FIRST,j,k)
                        PREPRE_ESZ_YZ(SECOND,j,k)       = PRE_ESZ_YZ(SECOND,j,k)
                        PREPRE_ESZ_YZ(PRELAST,j,k)      = PRE_ESZ_YZ(PRELAST,j,k)
                        PREPRE_ESZ_YZ(LAST,j,k)         = PRE_ESZ_YZ(LAST,j,k)

                        PRE_ESZ_YZ(FIRST,j,k)           = ES_Z(1,j,k)
                        PRE_ESZ_YZ(SECOND,j,k)          = ES_Z(2,j,k)
                        PRE_ESZ_YZ(PRELAST,j,k)         = ES_Z(max_x-1,j,k)
                        PRE_ESZ_YZ(LAST,j,k)            = ES_Z(max_x,j,k)
                END DO
        END DO


        RETURN
END
```

C.9. Empfänger aufzeichnen

Listing 22: FDTD - FullObservation

```
!#############################################################
!Choose and save the field componets for the last two time steps
!#############################################################
SUBROUTINE FullObservation(step)

        USE globals
        INTEGER, PARAMETER      :: I=1, J=2, K=3
        INTEGER                 :: ObsPoint
        INTEGER                 :: step

        DO ObsPoint=1, NoOfReceivers
                ReceiverData(ObsPoint,1,step) = ES_X(   ReceiverPoints(ObsPoint,I),&
&                                                       ReceiverPoints(ObsPoint,J),&
&                                                       ReceiverPoints(ObsPoint,K))
                ReceiverData(ObsPoint,2,step) = ES_Y(   ReceiverPoints(ObsPoint,I),&
&                                                       ReceiverPoints(ObsPoint,J),&
&                                                       ReceiverPoints(ObsPoint,K))
                ReceiverData(ObsPoint,3,step) = ES_Z(   ReceiverPoints(ObsPoint,I),&
&                                                       ReceiverPoints(ObsPoint,J),&
&                                                       ReceiverPoints(ObsPoint,K))
                ReceiverData(ObsPoint,4,step) = HS_X(   ReceiverPoints(ObsPoint,I),&
&                                                       ReceiverPoints(ObsPoint,J),&
&                                                       ReceiverPoints(ObsPoint,K))
                ReceiverData(ObsPoint,5,step) = HS_Y(   ReceiverPoints(ObsPoint,I),&
&                                                       ReceiverPoints(ObsPoint,J),&
&                                                       ReceiverPoints(ObsPoint,K))
                ReceiverData(ObsPoint,6,step) = HS_Z(   ReceiverPoints(ObsPoint,I),&
&                                                       ReceiverPoints(ObsPoint,J),&
&                                                       ReceiverPoints(ObsPoint,K))
        END DO

        RETURN
END
```

C.10. Beobachtungspunkte aufzeichnen

Listing 23: FDTD - Observe

```
!#################################################
!Choose and Save Field Componets after each timestep
!#################################################
SUBROUTINE Observe(time_step)

        USE globals
        INTEGER, PARAMETER      :: FieldType=1, I=2, J=3, K=4
        INTEGER                 :: time_step
        INTEGER                 :: ObsPoint, x_pos, y_pos, z_pos
```

```
        z_pos=FULLXY_Observation_Z;

        IF (FullXYObservation == .TRUE.) THEN
                DO y_pos = 1,max_y
                        DO x_pos = 1,max_x
                                XYObservationData(x_pos,y_pos,time_step) = ES_Z(x_pos,y_pos,z_pos)
                        END DO
                END DO
        ELSE
                DO ObsPoint=1, NoOfObsPoints
                        SELECT CASE(ObservationPoints(ObsPoint,FieldType))
                                CASE(1)
                                        ObservationData(ObsPoint,time_step) = &
                                                ES_X(   ObservationPoints(ObsPoint,I),&
&                                                       ObservationPoints(ObsPoint,J),&
&                                                       ObservationPoints(ObsPoint,K))
                                CASE(2)
                                        ObservationData(ObsPoint,time_step) = &
                                                ES_Y(   ObservationPoints(ObsPoint,I),&
&                                                       ObservationPoints(ObsPoint,J),&
&                                                       ObservationPoints(ObsPoint,K))
                                CASE(3)
                                        ObservationData(ObsPoint,time_step) = &
                                                ES_Z(   ObservationPoints(ObsPoint,I),&
&                                                       ObservationPoints(ObsPoint,J),&
&                                                       ObservationPoints(ObsPoint,K))
                                CASE(4)
                                        ObservationData(ObsPoint,time_step) = &
                                                HS_X(   ObservationPoints(ObsPoint,I),&
&                                                       ObservationPoints(ObsPoint,J),&
&                                                       ObservationPoints(ObsPoint,K))
                                CASE(5)
                                        ObservationData(ObsPoint,time_step) = &
                                                HS_Y(   ObservationPoints(ObsPoint,I),&
&                                                       ObservationPoints(ObsPoint,J),&
&                                                       ObservationPoints(ObsPoint,K))
                                CASE(6)
                                        ObservationData(ObsPoint,time_step) = &
                                                HS_Z(   ObservationPoints(ObsPoint,I),&
&                                                       ObservationPoints(ObsPoint,J),&
&                                                       ObservationPoints(ObsPoint,K))
                        END SELECT
                END DO
        END IF

        RETURN
END
```

D. Verwendete Software

Gentoo Linux - Betriebssystem Alle Arbeiten im Rahmen dieser Diplomarbeit wurden unter dem freien Betriebssystem Linux durchgeführt. Als Distribution kam gentoo Linux™ [33] zum Einsatz. Als Desktopumgebung wurde GNOME™ [34] verwendet.

MATLAB® - Arithmetikprogramm MATLAB® diente als Umgebung um die Optimierung, sowie die Steuerung des FDTD-Programms zu implementieren. Für alle Versuche wurden entsprechende Funktionen in der MATLAB® eigenen Programmiersprache entworfen. Für die Visualisierung der Ergebnisse fanden ebenfalls Funktionen dieser Umgebung Verwendung.

Intel® Fortran - FORTRAN Compiler Die FDTD Implementierung wurde in FORTRAN realisiert. Als Compiler wurde die Version 9.1 des Intel® FORTRAN Compilers verwendet.

gedit - FORTRAN SourceCode Editor Die FORTRAN Quellfiles wurden mit Hilfe des Texteditors *gedit* entworfen. Dieser ist Teil der GNOME™ Desktopumgebung.

GIMP - Bildbearbeitung Soweit es nötig war, wurden die Bilder mittels des freien Bildbearbeitungsprogramms *GIMP* bearbeitet[35].

Dia - Diagrammerstellung Zur Erstellung der Flussdiagramme kam das GNOME™ Projekt *Dia* zum Einsatz.

OpenOffice - Zeichnungen Zeichnungen wurden mit Hilfe des *Draw* Modules der Office-Umgebung *OpenOffice.org*[36] erstellt.

Latex - Testsatzsystem Das Setzen des schrifftlichen Teils der Arbeit wurde vom Textsatzsystem LaTeX übernommen.

[33] http://www.gentoo.org
[34] http://www.gnome.org
[35] http://www.gimp.org
[36] http://www.openoffice.org

WineFish - Latex IDE Mit *Winefish*[37] wurden die verschiedenen LaTeX-Files als Projekt verwaltet und editiert. Der integrierte Postprozessor übernahm auch das komilieren der LaTeX-Files und das Erstellen des PDF-Dokumentes.

E. Inhalt der CD-ROM

- LaTeX-Projekt des Schriftstückes
- PDF-Version des Schriftstückes
- Quellcode der FDTD in FORTRAN
- MATLAB® Scripte für sämtliche Versuche inklusive Erstellung der Bilder
- Verwendete Literatur soweit verfügbar
- Intel® Fortran Compiler

[37] http://winefish.berlios.de

Technische Universität Dresden
Fakultät Elektrotechnik
Elektrotechnisches Institut

Thesen zur Diplomarbeit

Thema:	Untersuchung und Implementierung numerischer Optimierungsmethoden zur Lösung inverser elektromagnetischer Streuprobleme
Tag der Einreichung:	14.08.2007
Verfasser:	Christian Stüllein

Das inverse Streuproblem, zur Rekonstruktion der dielektrischen Parameter eines Streukörpers, kann durch die Kombination von Vorwärtsrechnung und nichtlinearer Optimierung näherungsweise gelöst werden. Die vorliegende Arbeit befasst sich mit der FDTD-Methode (Finite Differenzen im Zeitbereich) als Vorwärtsrechnung und mit ausgewählten Optimierungsalgorithmen, um die Prinzipien dieses Ansatzes zu demonstrieren.

1. Die theoretischen Grundlagen der FDTD wurden anhand verfügbarer Literatur behandelt und die für die Implementierung benötigten Aktualisierungsgleichungen hergeleitet.
2. Die Notwendigkeit absorbierender Randbedingungen wurde begründet.
3. Die absorbierenden Randbedingungen nach Mur sind für die vorliegende Arbeit ausreichend genau.
4. Die FDTD Implementierung ist flexibel und kann einfach um weitere Funktionalitäten erweitert werden.
5. Die Ausführungsgeschwindigkeit der Vorwärtsrechung kann durch geeignete Maßnahmen beschleunigt werden.

6. Die FDTD ist zur Berechnung sich dynamisch ändernder elektromagnetischer Felder geeignet.

7. Die deterministischen Optimierungsverfahren finden nicht immer die globalen Minimalstellen des Fehlers. Die Rekonstruktion funktioniert daher nicht für jeden Startpunkt.

8. Mit zunehmender Komplexität des Streukörpers steigt der Rechenaufwand zur Bestimmung der Jacobimatrix.

9. Es ist prinzipiell möglich, durch Optimierungsverfahren das inverse Streuproblem zu lösen.

10. Die Klasse der heuristischen Optimierungsverfahren ist interessant für weitere Untersuchungen.

Heidelberg, 09.08.2007 Christian Stüllein

[Fan04]

[Tup97]

[Mey05]

[KL93]

[Qui01]

[Jan05]

[Str02]

[Dec02]

[ZKGY05]

[Yee66]

[Fen]

[Abe04]

[Ple05]

[Els96]

[Gon04]

[Sch00]

[EN04]