Transient Simulation of Eddy Currents in Ferromagnets

Semesterarbeit von Manuel Guidon bei Herrn Prof. Dr. R. Hiptmair

Winter 2003/04

Inhaltsverzeichnis

1 Einleitung

In der vorliegenden Semesterarbeit soll ein Relaxationsverfahren in 2D implementiert und getestet werden. Das Verfahren wird auf den nichtlinearen Zusammenhang zwischen magnetischer Induktion $\mathbf{B}$ und magnetischem Feld $\mathbf{H}$ in einer ferromagnetischen Platte angewandt. Für die Diskretisierung wird eine auf Kantenelementen basierende FE Methode verwendet.

2 Beschreibung des Problems

Für die elektromagnetischen Felder in einem Ferromagneten gelten die folgenden Wirbelstromgleichungen:

$$\sigma \mathbf{E} - \mathbf{rot}\mathbf{H} = -\mathbf{j_0} \tag{1}$$

$$\mathbf{div}\mathbf{B} = 0 \tag{2}$$

Wobei $\mathbf{E}$ das elektische Feld, $\mathbf{H}$ das magnetische Feld, $\mathbf{B}$ die magnetische Induktion und $\mathbf{j_0}$ die externe Stromquelle beschreiben. σ steht für die elektrische Leitfähigkeit.
Unter Verwendung des Vektorpotentials $\mathbf{A}$ in der Coulomb-Eichung mit $\mathbf{div}\mathbf{A} = 0$ und des Skalarpotentials ϕ mit $\phi = 0$ lassen sich diese beiden Gleichungen umschreiben zu

$$\mathbf{rot}\mathbf{A} = \mathbf{B} \tag{3}$$

$$\mathbf{rot}\mathbf{H} = -\sigma \frac{d}{dt}\mathbf{A} + \mathbf{j_0}. \tag{4}$$

Leicht ferromagnetische Substanzen zeigen einen nichtlinearen Zusammenhang zwischen magnetischer Induktion $\mathbf{B}$ und magnetischem Feld $\mathbf{H}$ wie er in Abb. 1 dargestellt ist. Einsetzen der Beziehung $\mathbf{H} = \mathbf{H}(\mathbf{B})$ in Gleichung (4) liefert

$$\mathbf{rot}\mathbf{H}(\mathbf{rot}\mathbf{A}) = -\sigma \frac{d}{dt}\mathbf{A} + \mathbf{j_0}. \tag{5}$$

Ausführen einer Rotation auf beiden Seiten der Gleichung ergibt die gewünschte Formulierung in der Unbekannten $\mathbf{H}$

$$\mathbf{rot}\sigma^{-1}\mathbf{rot}\mathbf{H} = -\frac{d}{dt}\mathbf{B}(\mathbf{H}) + \mathbf{rot}(\sigma^{-1}\mathbf{j_0}). \tag{6}$$

Der Einfachheit halber wird als Rechengebiet $\Omega \subseteq \mathbb{R}^2$ eine quadratische zweidimensionale Platte angesetzt, mit den Randbedingungen

$$\mathbf{H} \times \mathbf{n} = 0 \qquad auf \quad \partial\Omega. \tag{7}$$

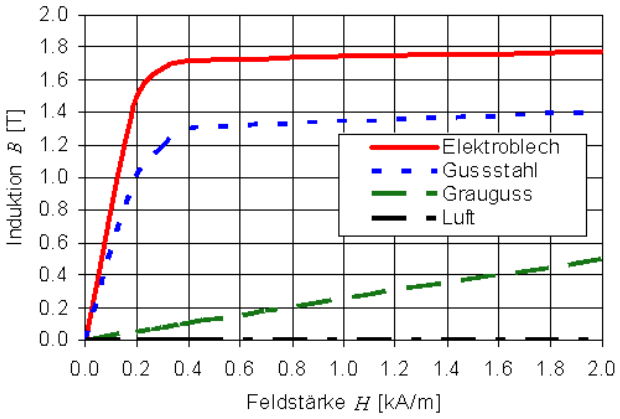

Abbildung 1: Magnetisierung verschiedener Materialien

3 Die Kantenelemente FE Methode

Da das vorliegende Problem zweidimensionaler Natur ist, müssten die auftretenden Felder in ihre skalaren Komponenten zerlegt werden, wenn als unbekannte Parameter in den finiten Elementen Knotenelemente verwendet würden. Somit gäbe es auf jedem Knoten zwei Unbekannte. Vektoren besitzen aber physikalische und mathematische Eigenschaften, die sich nicht so leicht in den Komponeneten eines Koordinatensystems ausdrücken lassen. Bei elektromagnetischen Feldern gibt es beispielsweise oft eine Randbedingung, die nur die tangentiale Komponente an den Elementgrenzen berücksichtigt. In der Regel sind solche Bedingungen sehr schwierig zu implementieren.

Eine Alternative dazu bieten die sogenannten Kantenelemente (engl. *edge elements*). Hierbei werden als unbekannte Parameter vektorielle Grössen, welche den Kanten der Elemente zugeordnet sind, verwendet. Die in dieser Arbeit verwendeten Elemente werden als „Whitney-Elemente der Ordnung 1“ bezeichnet.

3.1 Die lokalen Koordinaten

Die zu modellierende Platte soll mit Dreiecken diskretisiert werden. Für das lokale Koordinatensystem bieten sich daher Ansatzfunktionen in baryzentrischen Koordinaten an. Wie in Abb. 2 zu sehen, berechnet sich die erste baryzentrische Koordinate ξ_1 aus dem Verhältnis der Teilfläche A_{P23} zur Gesamtfläche A_{123} des Dreiecks.

Somit definieren sich die drei baryzentrischen Koordinaten im Dreieck wie folgt

$$\xi_1 = \frac{A_{p23}}{A_{123}} \tag{8}$$

$$\xi_2 = \frac{A_{1p3}}{A_{123}} \tag{9}$$

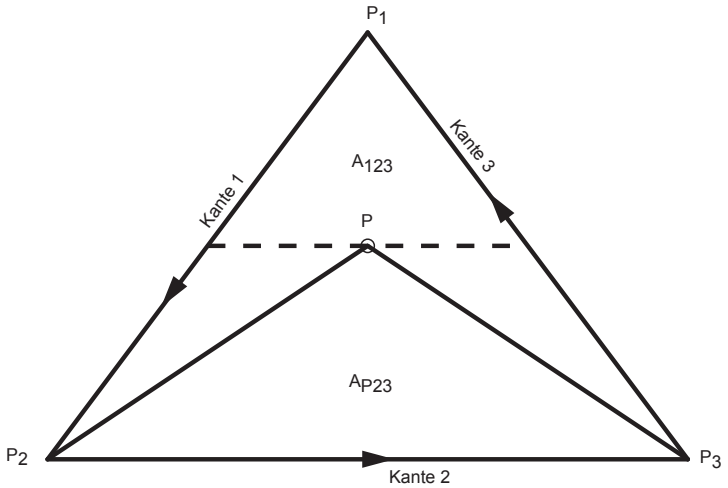

Abbildung 2: Baryzentrische Koordinate $\xi_1 = \frac{A_{P23}}{A_{123}}$ im Dreieck. $\xi_1 = const$ entlang der gesrichelten Linie

$$\xi_2 = \frac{A_{12p}}{A_{123}} \tag{10}$$

Selbstverständlich sind diese drei Koordinaten in einem zweidimensionalen Objekt linear abhängig. Deshalb gilt:

$$\xi_1 + \xi_2 + \xi_3 = 1 \tag{11}$$

3.2 Die kantenbsierten Ansatzfunktionen im Dreieckselement

Bei den hier verwendeten Formfunktionen beschreibt ein Koeffiezient den Betrag der tangentialen Vektorgrösse auf der Kante. Damit die Richtung der tangentialen Vektorgrösse eindeutig zugeordnet ist, wird eine feste Orientierung für jede Kante festgelegt (vgl Abb. 2). Durch das Vorzeichen des Koeffizienten ist somit auch die Richtung auf jeder Kante eindeutig:

$$Kante1 = P_1P_2 \tag{12}$$
$$Kante2 = P_2P_3 \tag{13}$$
$$Kante3 = P_3P_1 \tag{14}$$
$$\tag{15}$$

Für die Approximation im Element werden nun drei tangential unabhängige Vektorbasisfunktionen benötigt. Whitney führte hierzu den folgenden Anzatz ein:

$$\begin{aligned} \mathbf{N}_1 &= \xi_1 \vec{\nabla} \xi_2 - \xi_2 \vec{\nabla} \xi_1 \\ \mathbf{N}_2 &= \xi_2 \vec{\nabla} \xi_3 - \xi_3 \vec{\nabla} \xi_2 \\ \mathbf{N}_3 &= \xi_3 \vec{\nabla} \xi_1 - \xi_1 \vec{\nabla} \xi_3 \end{aligned} \tag{16}$$

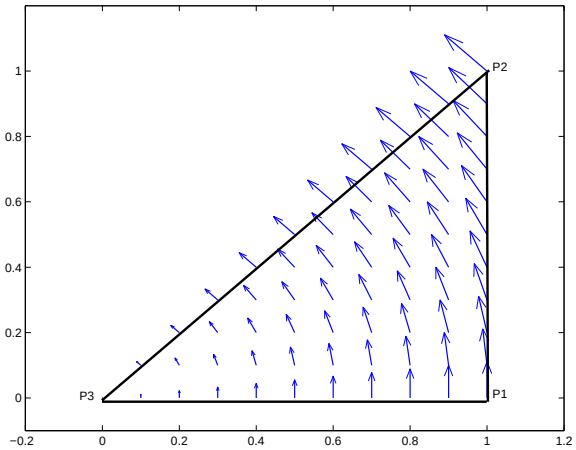

Abbildung 3: Whitney Basisfunktion $\mathbf{N}_1$

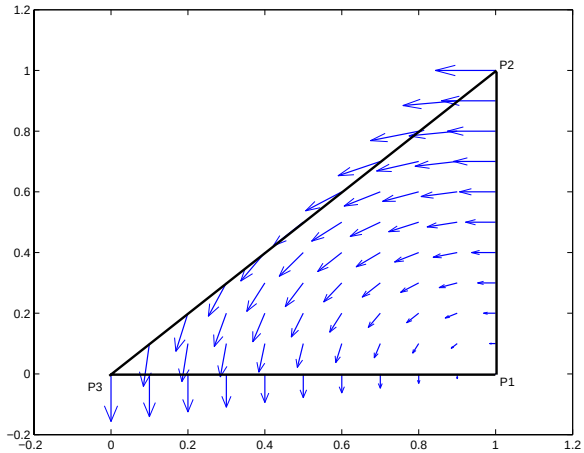

Abbildung 4: Whitney Basisfunktion $\mathbf{N}_2$

Jede dieser drei Ansatzfunktionen stellt eine Rotation um den gegenüberliegenden Dreiecks-Eckpunkt dar (vgl Abb. 3- Abb. 5)

Mittels dieser drei Basisfunktionen läesst sich nun jedes beliebige Vektorfeld $\mathbf{A}^e$ im Element e wie folgt ausdrücken:

$$\mathbf{A}^e = \sum_{i=1}^{3} A_i^e \mathbf{N}_i^e. \tag{17}$$

Hierbei beschreibt A_i^e die tangentiale Feldkomponente des Feldes $\mathbf{A}^e$ entlang der i-ten Kante. In Abb. 6 ist beispielsweise ein Feld dargestellt, bei welchem alle Komponenten den Betrag 1 haben.

3.3 Die Variationsformulierung des Problems

Zur Herleitung einer Variationsformulierung geht man von Gleichung (6) aus und integriet sie über den ganzen Raum $\Omega \subseteq \mathbb{R}^2$:

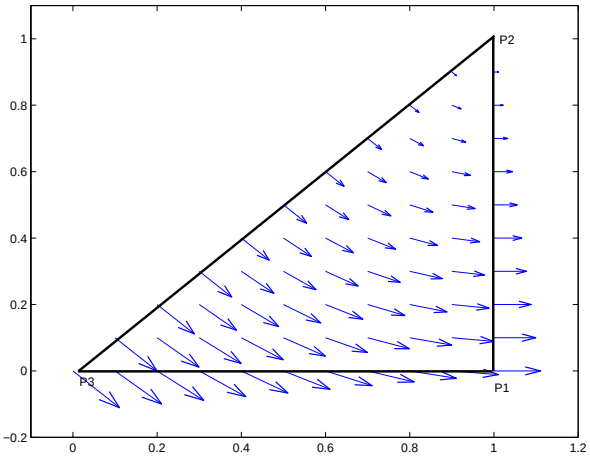

Abbildung 5: Whitney Basisfunktion $\mathbf{N}_3$

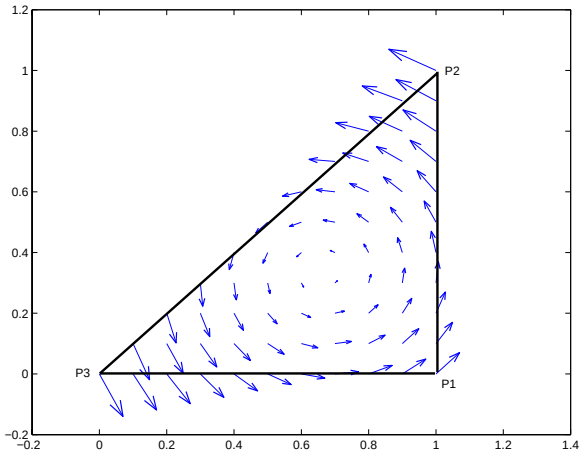

Abbildung 6: Beispiel für ein Vektorfeld

$$\int_\Omega \mathbf{rot}\sigma^{-1}\mathbf{rotH}d\mathbf{x} = -\int_\Omega \frac{d}{dt}\mathbf{B}(\mathbf{H}) + \int_\Omega \mathbf{rot}\sigma^{-1}\mathbf{j_0} \tag{18}$$

Schliesslich multipliziert man diese Gleichung mit der Testfunktion $\mathbf{N}$ und vereinfacht sie unter Verwendung der vektoriellen Green'schen Sätze

$$\int_\Omega \mathbf{rot}\sigma^{-1}\mathbf{rotH}\cdot\mathbf{N}d\mathbf{x} = -\int_\Omega \frac{d}{dt}\mathbf{B}(\mathbf{H})\cdot\mathbf{N}d\mathbf{x} + \int_\Omega \mathbf{rot}\sigma^{-1}\mathbf{j_0}\cdot\mathbf{N}d\mathbf{x}. \tag{19}$$

Die linke Seite lässt sich umschreiben zu

$$\int_\Omega \sigma^{-1}\mathbf{rotH}\cdot\mathbf{rotN}d\mathbf{x} + \int_{\partial\Omega} \mathbf{n}\times(\sigma^{-1}\mathbf{rotH})\cdot\mathbf{N}d\mathbf{x}. \tag{20}$$

Analog liefert die rechte Seite:

$$-\int_{\Omega} \frac{d}{dt}\mathbf{B}(\mathbf{H}) \cdot \mathbf{N} d\mathbf{x} + \int_{\Omega} \sigma^{-1}\mathbf{j_0}\mathbf{rot}\mathbf{N} d\mathbf{x} - \int_{\partial\Omega} \mathbf{n} \times \mathbf{N}\sigma^{-1}\mathbf{j_0} d\mathbf{x}. \tag{21}$$

Aufgrund der Randbedingung (7) vereinfacht sich die Gleichung weiter und man erhält für die Variationsformulierung folgenden Ausdruck:

$$\int_{\Omega} \sigma^{-1}\mathbf{rotH} \cdot \mathbf{rotN} d\mathbf{x} = -\int_{\Omega} \frac{d}{dt}\mathbf{B}(\mathbf{H}) \cdot \mathbf{N} d\mathbf{x} + \int_{\Omega} \sigma^{-1}\mathbf{j_0}\mathbf{rot}\mathbf{N} d\mathbf{x}. \tag{22}$$

3.4 Berechnung der Elementmatrizen

In diesem Abschnitt werden die lokalen Elementmatrizen für die Gleichung (22) hergeleitet.
Da es sich bei den baryzentrischen Koordinaten um lineare Funktionen in x und y handelt, kann folgender Ansatz gemacht werden:

$$\xi_1 = \frac{a_1 + b_1 x + b_1 y}{2A} \tag{23}$$

$$\xi_2 = \frac{a_2 + b_2 x + c_2 y}{2A} \tag{24}$$

$$\xi_3 = \frac{a_3 + b_3 x + c_3 y}{2A}. \tag{25}$$

Hierbei bezeichnet A die Fläche des Dreieck-Elements. Sie wird über eine Determinante berechnet.

$$A = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \tag{26}$$

Ebenso können die Koeffizienten a_i, b_i und c_i mittels Determinanten ausgedrückt werden.

$$a_1 = \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} \qquad b_1 = -\begin{vmatrix} 1 & y_2 \\ 1 & y_3 \end{vmatrix} \qquad c_1 = \begin{vmatrix} 1 & x_2 \\ 1 & x_3 \end{vmatrix} \tag{27}$$

$$a_2 = \begin{vmatrix} x_3 & y_3 \\ x_1 & y_1 \end{vmatrix} \qquad b_2 = -\begin{vmatrix} 1 & y_3 \\ 1 & y_1 \end{vmatrix} \qquad c_2 = \begin{vmatrix} 1 & x_3 \\ 1 & x_1 \end{vmatrix} \tag{28}$$

$$a_3 = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} \qquad b_3 = -\begin{vmatrix} 1 & y_1 \\ 1 & y_2 \end{vmatrix} \qquad c_3 = \begin{vmatrix} 1 & x_1 \\ 1 & x_2 \end{vmatrix} \tag{29}$$

Für die Berechnung von Integralen über das Produkt von baryzentrischen Koordinaten auf einer Dreicksfläche A existiert eine nützliche Identität

$$\int_{A} (\xi_1)^k (\xi_2)^l (\xi_3)^m dA = 2A \frac{k!l!m!}{(k+l+m+2)!}. \tag{30}$$

Mit Hilfe obiger Formeln lassen sich nun die Element-Matrizen des Variationsproblems (22) berechnen. Hierzu werden die einzelnen Integrale separat betrachtet.

$$\mathbb{A}_{ij}^e := \int_A (\vec{\nabla} \times \mathbf{N}_i^e)(\vec{\nabla} \times \mathbf{N}_j^e) dA \tag{31}$$

$$\mathbb{B}_{ij}^e := \int_A \mathbf{N}_i^e \mathbf{N}_j^e dA \tag{32}$$

$$\mathbb{C}_i^e := \int_A \mathbf{j_0} \mathbf{rot} \mathbf{N}_i^e dA \tag{33}$$

3.4.1 Berechnung der Matrix $\mathbb{A}$

Aufgrund der Linearität der baryzentrischen Koordinaten sind die Einträge $\mathbb{A}_{ij}$ konstant über dem Dreieck. Mit Hilfe der folgenden drei Identitäten lassen sich die Matrixeinträge einfach berechnen.

$$\vec{\nabla} \times \mathbf{N}_1 = \frac{1}{2A^2} \begin{pmatrix} 0 \\ 0 \\ b_1c_2 - b_2c_1 \end{pmatrix} \tag{34}$$

$$\vec{\nabla} \times \mathbf{N}_2 = \frac{1}{2A^2} \begin{pmatrix} 0 \\ 0 \\ b_2c_3 - b_3c_2 \end{pmatrix} \tag{35}$$

$$\vec{\nabla} \times \mathbf{N}_1 = \frac{1}{2A^2} \begin{pmatrix} 0 \\ 0 \\ b_3c_1 - b_1c_3 \end{pmatrix} \tag{36}$$

$$\tag{37}$$

Somit erhält man für die Matrix $\mathbb{A}$:

$$\mathbb{A} = \frac{1}{4A^3} \begin{pmatrix} (b_1c_2 - b_2c_1)^2 & (b_1c_2 - b_2c_1)(b_2c_3 - b_3c_2) & (b_1c_2 - b_2c_1)(b_3c_1 - b_1c_3) \\ (b_1c_2 - b_2c_1)(b_2c_3 - b_3c_2) & (b_2c_3 - b_3c_2)^2 & (b_2c_3 - b_3c_2)(b_3c_1 - b_1c_3) \\ (b_1c_2 - b_2c_1)(b_3c_1 - b_1c_3) & (b_2c_3 - b_3c_2)(b_3c_1 - b_1c_3) & (b_3c_1 - b_1c_3)^2 \end{pmatrix}. \tag{38}$$

3.4.2 Berechnung der Matrix $\mathbb{B}$

Unter Verwendung von (30) ergibt sich beispielsweise für den Matrixeintrag $\mathbb{B}_{11}$:

$$\begin{aligned} \mathbb{B}_{11} &= \int_A (\xi_1 \vec{\nabla} \xi_2 - \xi_2 \vec{\nabla} \xi_1)^2 dA && (39) \\ &= \frac{1}{4A^2} \int_A (\xi_1^2(b_2^2 + c_2^2) - 2\xi_1\xi_2(b_1b_2 + c_1c_2) + \xi_2^2(b_1^2 + c_1^2)) dA && (40) \end{aligned}$$

$$= \frac{1}{2A4!}(2(b_2^2+c_2^2)-2b_1b_2+c_1c_2)+2(b_1^2+c_1^2)) \tag{41}$$

$$= \frac{1}{24A}(b_2^2+c_2^2-b_1b_2-c_1c_2+b_1^2+c_1^2) \tag{42}$$

Durch analoge Berechnungen erhält man

$$\mathbb{B}_{12} = \frac{1}{48A}(b_2b_3+c_2c_3-(b_2^2+c_2^2)-2(b_1b_3+c_1c_3)+b_1b_2+c_1c_2) \tag{43}$$

$$\mathbb{B}_{13} = \frac{1}{48A}(b_1b_2+c_1c_2-(b_1^2+c_1^2)-2(b_2b_3+c_2c_3)+b_1b_3+c_1c_3) \tag{44}$$

$$\mathbb{B}_{21} = \mathbb{B}_{12} \tag{45}$$

$$\mathbb{B}_{22} = \frac{1}{24A}(b_3^2+c_3^2-b_2b_3-c_2c_3+b_2^2+c_2^2) \tag{46}$$

$$\mathbb{B}_{23} = \frac{1}{48A}(b_1b_3+c_1c_3-(b_3^2+c_3^2)-2(b_1b_2+c_1c_2)+b_2b_3+c_2c_3) \tag{47}$$

$$\mathbb{B}_{31} = \mathbb{B}_{13} \tag{48}$$

$$\mathbb{B}_{32} = \mathbb{B}_{23} \tag{49}$$

$$\mathbb{B}_{33} = \frac{1}{24A}(b_1^2+c_1^2-b_1b_3-c_1c_3+b_3^2+c_3^2) \tag{50}$$

3.4.3 Berechnung des Vektors $\mathbb{C}$

Analog den vorhergehenden Berechnungen gelten für die Komponenten des Vektors $\mathbb{C}$ die folgenden Gleichungen:

$$\mathbb{C}_1 = \frac{1}{2A^2}\begin{pmatrix} 0 \\ 0 \\ b_1c_2-b_2c_1 \end{pmatrix} \cdot \int_A \mathbf{j_0} d\mathbf{x} \tag{51}$$

$$\mathbb{C}_2 = \frac{1}{2A^2}\begin{pmatrix} 0 \\ 0 \\ b_2c_3-b_3c_2 \end{pmatrix} \cdot \int_A \mathbf{j_0} d\mathbf{x} \tag{52}$$

$$\mathbb{C}_3 = \frac{1}{2A^2}\begin{pmatrix} 0 \\ 0 \\ b_3c_1-b_1c_3 \end{pmatrix} \cdot \int_A \mathbf{j_0} d\mathbf{x} \tag{53}$$

Die Integrale auf der rechten Seite werden durch eine Mittelpunktsregel im Dreieck numerisch ausgewertet.

$$\int_A \mathbf{f} d\mathbf{x} = \frac{1}{3}A\sum_{j=1}^{3}\mathbf{f}(\mathbf{m}_j) \tag{54}$$

wobei $\mathbf{m}_j$ den Mittelpunkt der Kante j im Dreieck bezeichnet.

3.5 Lokales und globales Gleichungssystem

Für das Variationsproblem (22) wird gemäss (17) folgender Ansatz für das magnetische Feld $\mathbf{H}^e$ gemacht:

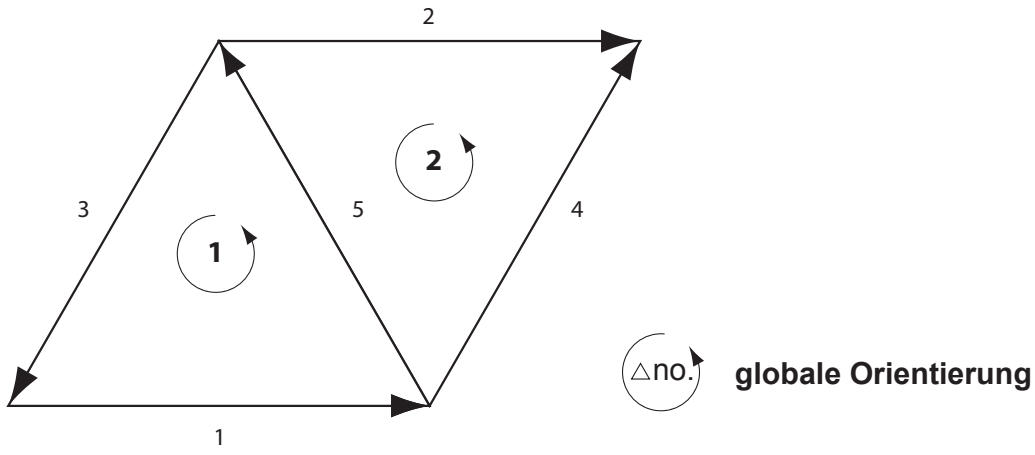

Abbildung 7: lokale vs. globale Kantenorientierung

$$\mathbf{H}^e = \sum_{i=1}^{3} H_i^e \mathbf{N}_i^e. \tag{55}$$

Damit muss für jedes Element die lokale Matrixgleichung

$$\sigma^{-1}\mathbb{A}^e\mathbf{H}^e = -\frac{d}{dt}\left(\mathbb{B}^e(\mathbf{B}(\mathbf{H}))\right) + \sigma^{-1}\mathbb{C}^e \tag{56}$$

gelöst werden. Wobei $\mathbf{H}^e$ den Vektor der unbekannten Parameter beschreibt. Bei der Einarbeitung des Elementgleichungssystems in das Gesamtgleichungssystem ist darauf zu achten, dass die globale Kantenrichtung des Dreiecksgitters respektiert wird. Stimmen lokale und globale Orientierung nicht überein, so muss bei den den Matrixeinträgen der lokalen Matrizen das Vorzeichen der entsprechenden Zeilen und Spalten geändert werden. Die lokale Matrix des Dreiecks 2 in Abb. 7 müsste beispielsweise folgendermassen abgeändert werden:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad \Longrightarrow \quad \begin{pmatrix} a_{11} & a_{12} & -a_{13} \\ a_{21} & a_{22} & -a_{23} \\ -a_{31} & -a_{32} & a_{33} \end{pmatrix}. \tag{57}$$

Wobei die lokale Kante 1 der globalen Kante 2, die lokale Kante 2 der globalen Kante 5 und die lokale Kante 3 der globalen Kante 4 entspricht.

4 Zeitdiskretisierung

Nach der Assemblierung des Gesamtgleichungssystems, ist das folgende Problem zu lösen:

$$\sigma^{-1}\mathbb{A}\mathbf{H} = -\frac{d}{dt}\left(\mathbb{B}(\mathbf{B}(\mathbf{H}))\right) + \sigma^{-1}\mathbb{C}. \tag{58}$$

Hierbei entspricht $\mathbf{H}$ dem Kantenvektor aller unbekannten Parameter. Für einen einfachen linearen Zusammenhang zwischen magnetischer Induktion und magnetischem Feld $\mathbf{B}(\mathbf{H})$ kann die Zeitdiskretisierung mittels eines impliziten Euler Verfahrens gemacht werden. Dieser Ansatz liefert eine Gleichung, welche für jeden Zeitschritt τ nach $\mathbf{H}_{i+1}$, wo $\mathbf{H}_i \approx \mathbf{H}(i \cdot \tau)$, aufgelöst werden kann.

$$\sigma^{-1}\mathbb{A}\mathbf{H}_i = \sigma^{-1}\mathbb{C} - \frac{1}{\tau}(\mathbb{B}(\mathbf{B}(\mathbf{H}_i)) - \mathbb{B}(\mathbf{B}(\mathbf{H}_{i-1}))) \tag{59}$$

Wie bereits oben erwähnt ist der Zusammenhang $\mathbf{B}(\mathbf{H})$ keinesfalls einfacher Natur, sondern nichtlinear. In dieser Semesterarbeit wurden deshalb zwei andere Verfahren verwendet. Zum einen wurde ein Iterationsverfahren implementiert und zum andern ein bestehendes Relaxationsverfahren für skalare Probleme auf mehrere Dimensionen weiterentwickelt.

4.1 Das Fixpunkt-Iterationsverfahren

Im Folgenden soll ein Fixpunkt-Iterationsverfahren vorgestellt werden, welches den nichtlinearen Zusammenhang zwischen magnetischem Feld und magnetischer Induktion miteinbezieht.

4.1.1 Skalares Modell Problem

Man betrachte die skalare Modell Gleichung

$$\frac{\partial}{\partial t}u + au = f \tag{60}$$

wobei $u = u(t)$, $a = const$ und $f = f(u)$. Bezeichnet τ den Zeitschritt und $u_i \approx u(i \cdot \tau)$, dann ergibt sich mit der impliziten Euler Methode folgendes Zeitschritt Schema:

$$u_i + \tau a u_i = u_{i-1} + \tau f(u_i). \tag{61}$$

In Hinblick auf ein Fixpunkt-Iterationsverfahren lässt sich dieser Ausdruck umformulieren zu

$$u_i = (1 + \tau a)^{-1}(u_{i-1} + \tau f(u_i)) =: g(u_i). \tag{62}$$

Um die Lösung u_i dieser Gleichung zu finden, beginnt man bei einem „sinnvollen“ Startpunkt $u_i^{(0)}$ und berechnet sukzessive für $j = 0, 1, 2, ...$

$$u_i^{(j+1)} = g(u_i^{(j)}) = (1 + \tau a)^{-1}(u_{i-1} + \tau f(u_i^{(j)})). \tag{63}$$

Diese Iteration wird solange ausgeführt, bis $\left\| u_i^{(j+1)} - u_i^{(j)} \right\|$ genügend klein wird.

4.1.2 Anwendung des Fixpunkt-Iterationsverfahren auf den Ferromagnetismus

Durch Umformen kann Gleichung (63) direkt auf Problem (59) übertragen werden.

$$\sigma^{-1}\mathbb{A}\mathbf{H}_i = \sigma^{-1}\mathbb{C} - \frac{1}{\tau}[\mathbb{B}(\mathbf{B}(\mathbf{H}_i)) - \mathbb{B}(\mathbf{B}(\mathbf{H}_{i-1}))] \tag{64}$$

$$\tau\sigma^{-1}\mathbb{A}\mathbf{H}_i = \tau\sigma^{-1}\mathbb{C} - [\mathbb{B}(\mathbf{B}(\mathbf{H}_i)) - \mathbb{B}(\mathbf{B}(\mathbf{H}_{i-1}))] \tag{65}$$

Damit erhält man einen analogen Ausdruck zu (62)

$$\mathbf{H}_i = (\tau\sigma^{-1}\mathbb{A})^{-1}(\tau\sigma^{-1}\mathbb{C} - [\mathbb{B}(\mathbf{B}(\mathbf{H}_i)) - \mathbb{B}(\mathbf{B}(\mathbf{H}_{i-1}))] =: \mathbf{g}(\mathbf{H}_i) \tag{66}$$

Für die Lösung dieser Gleichung beginnt man analog zum skalaren Fall mit einem passenden Startvektor $\mathbf{H}_i^0$ und berechnet für $j = 0, 1, 2, ...$

$$\mathbf{H}_i^{(j+1)} = \mathbf{g}(\mathbf{H}_i^{(j)}) = (\tau\sigma^{-1}\mathbb{A})^{-1}[\tau\sigma^{-1}\mathbb{C} - \mathbb{B}(\mathbf{B}(\mathbf{H}_i^{(j)})) - (\mathbb{B}(\mathbf{B}(\mathbf{H}_{i-1}))] \tag{67}$$

Dieses Verfahren wurde mit Matlab implementiert und dient als Grundlage zum Vergleich mit dem im nächsten Abschnitt diskutierten Relaxationsverfahren. Beispiele und Resultate finden sich am Ende dieser Arbeit.

4.2 Das Relaxationsverfahren

Das Relaxationsverfahren basiert auf einer Methode, welche von Jäger und Kacur für eindimensionale, nichtlineare Probleme entwickelt wurde. Zunächst soll ein skalares Modell Problem betrachtet werden und dann dessen Erweiterung auf den hier vorliegenden zweidimensionalen Fall besprochen werden.

4.2.1 Skalares Modell Problem: Variante 1

Man betrachte das folgende Problem

$$\begin{aligned} \frac{d}{dt}u - \triangle\beta(u) &= f(\mathbf{x}, t) \qquad \mathbf{x} \in \Omega, \qquad 0 < t < T, \\ u &= 0 \qquad auf \quad \partial\Omega\times]0, T[. \end{aligned} \tag{68}$$

Hierbei sei $\beta : \mathbb{R} \mapsto \mathbb{R}$ Lipschitz-stetig und monoton steigend.
(68) kann umgeschrieben werden zu

$$\begin{aligned} \frac{d}{dt}\beta^{-1}(\Theta) - \triangle\Theta &= f(\mathbf{x}, t) \qquad \mathbf{x} \in \Omega, \qquad 0 < t < T, \\ \Theta &= 0 \qquad auf \quad \partial\Omega\times]0, T[. \end{aligned} \tag{69}$$

Mit einem impliziten Euler Schema wird daraus

$$\beta^{-1}(\Theta_i) - \beta^{-1}(\Theta_{i-1}) - \tau\triangle\Theta = \tau f_i, \tag{70}$$

wobei $u_i = \beta^{-1}(\Theta_i)$ eine Approximation von $u(\mathbf{x}, t)$ für $t = t_i = i\tau$ mit τ als Zeitschritt. Die linke Seite obiger Gleichung lässt sich umformen zu

$$\begin{aligned} \beta^{-1}(\Theta_i) - \beta^{-1}(\Theta_{i-1}) &= \frac{\beta^{-1}(\Theta_i - \beta^{-1}(\Theta_{i-1})}{\Theta_i - \Theta_{i-1}} \cdot (\Theta_i - \Theta_{i-1}) && (71) \\ &= \lambda_i(\Theta_i - \Theta_{i-1}) = \lambda_i(\Theta_i - \beta(u_{i-1})), && (72) \end{aligned}$$

wo

$$\lambda_i \approx (\beta'(\beta^{-1}(\xi_i)))^{-1}, \qquad \xi_i \in]\Theta_{i-1}, \Theta_i[. \tag{73}$$

An dieser Stelle nimmt man nun eine Regularisierung vor. Mit $0 < \alpha < 1$ (nahe bei 1):

$$\lambda_i = min\left\{K, \frac{\beta^{-1}(\beta(u_{i-1}) + \alpha(\Theta_i - \beta(u_{i-1}))) - u_{i-1}}{\Theta_i - \beta(u_i - 1)}\right\}. \tag{74}$$

Zusätzliche Regularisierung: Anstelle von β wird β_n verwendet mit $\beta_n' \geq \gamma_n > 0$. Ausserdem wird eine obere Schranke K für λ_i gesetzt. Somit erhält man als Updateformel

$$u_i = u_{i-1} + \lambda_i(\Theta_i - \beta(u_{i-1})). \tag{75}$$

Die Funktion Θ_i wird bestimmt als Lösung des linearen elliptischen Randwertproblems

$$\lambda_i(\Theta_i - \beta(u_{i-1})) - \tau\triangle\Theta_i = \tau f_i. \tag{76}$$

Das Lösungsschema sieht dann wie folgt aus: In jedem Zeitschritt τ werden die beiden Funktionen Θ_i und λ_i durch eine Fixpunkt Iteration gelöst, welche (75) und (76) miteinbezieht.

4.2.2 Skalares Modell Problem: Variante 2

In Hinblick auf die ferromagnetische Anwendung des Relaxationsverfahrens betrachte man eine weitere Modell Gleichung:

$$\begin{aligned} \frac{d}{dt}b(\theta) - \triangle\theta &= f(\mathbf{x}, t) \qquad \mathbf{x} \in \Omega, \quad 0 < t < T, && (77) \\ \theta &= 0 \quad auf \quad \partial\Omega\times]0, T[. \end{aligned}$$

Für die Funktion $b(s)$ wird angenommen, dass sie als Ableitung eines strikt konvexen Potentials $\Phi(s)$ darstellbar ist, d.h.

$$b(s) = \Phi'(s), \qquad \Phi : \mathbb{R} \mapsto \mathbb{R}. \tag{78}$$

Für die Relaxationsmethode nimmt man nun eine Regularisierung vor und ersetzt

$$\Phi \to \Phi_\epsilon \in C^2(\mathbb{R}), \qquad \epsilon > 0. \tag{79}$$

Für das regularisierte Potential soll gelten: $\lim\limits_{\epsilon \to 0} \Phi_\epsilon = \Phi$.

Sei $\tau > 0$ wiederum der Zeitschritt. Wir nehmen an, dass

$$c_0 \leq \Phi''(s) \leq \tau^{-d} \qquad f\ddot{u}r \quad ein \quad 0 < d < \frac{1}{4}. \tag{80}$$

Ausserdem soll es eine konvexe Funktion $g : \mathbb{R} \mapsto \mathbb{R}$ so dass

$$g(|s_1 - s_2|) \leq (b_\epsilon(s_1) - b_\epsilon(s_2))(s_1 - s_2) \qquad \forall s_1, s_2 \in \mathbb{R}. \tag{81}$$

Schliesslich gelte noch die folgende Annahme:

$$\exists c_1, c_2, \delta > 0 : \qquad |b_\epsilon(s)|^{1+\delta} \leq c_1 \left(b(s)s - \int\limits_0^1 b(ts)sdt \right) + c_1 \qquad \forall s \in \mathbb{R}. \tag{82}$$

Das implizite Zeitschritt Schema um $\Theta_i \approx \Theta(i \cdot \tau)$ lautet in diesem Fall

$$\lambda_i(\widetilde{\Theta_i} - \Theta_{i-1}) - \tau \triangle \widetilde{\Theta}_i = \tau f \quad in \quad \Omega, \quad mit \quad \widetilde{\Theta}_i = 0 \quad auf \quad \partial\Omega, \tag{83}$$

wobei die „Konvergenz Bedingung“

$$\left| \lambda_i - \gamma \int\limits_0^1 b'_\epsilon(\Theta_{i-1} + t\gamma(\Theta_i - \Theta_{i-1}))dt \right| < \tau \tag{84}$$

erfüllt werden muss mit $\gamma \in]0, 1[$, $\gamma \approx 1$. Die Näherung für den aktuellen Zeitschritt erhält man dann durch auflösen von

$$b(\Theta_i) = b(\Theta_{i-1}) + \lambda_i(\widetilde{\Theta}_i - \Theta_{i-1}) \tag{85}$$

nach Θ_i.

4.2.3 Anwendung des Relaxationsverfahrens auf den Ferromagnetismus

Ziel ist es nun, eine vektorwertige Variante des vorangehend vorgestellten Relaxations-Schema zu finden. Im speziellen Fall eines Ferromagneten, sieht das gesuchte Potential wie folgt aus:

$$\Phi(\mathbf{H}) = \frac{1}{2}\mu_0(|\mathbf{H}| + B_0\mu_0^{-1})^2 \qquad \mathbf{H} \in \mathbb{R}^2. \tag{86}$$

Hierbei ist μ_0 die Permeabilität des Vakuums und B_0 eine beliebige Konstante. Daraus ergibt sich der nichtlineare Zusammenhang

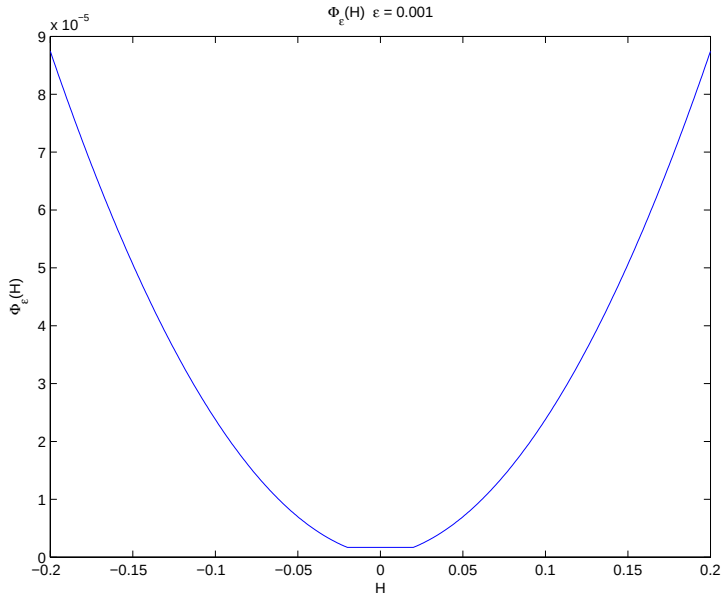

Abbildung 8: Regularisiertes Potential $\Phi_\epsilon(H)$

$$\mathbf{B}(\mathbf{H}) = \mathbf{grad}\Phi(\mathbf{H}) = \mu_0\mathbf{H} + B_0\frac{\mathbf{H}}{|\mathbf{H}|}. \tag{87}$$

Das Potential Φ kann auf eine natürliche Weise regularisiert werden (vgl. Abb. 8)

$$\Phi \to \Phi_\epsilon(\mathbf{H}) := \frac{1}{2}\mu_0(\sqrt{|\mathbf{H}|^2+\epsilon^2} + B_0\mu_0^{-1})^2, \qquad \epsilon > 0. \tag{88}$$

Damit ergibt sich (vgl. Abb. 9)

$$\begin{aligned} \mathbf{B}_\epsilon(\mathbf{H}) &= \mu_0(\sqrt{|\mathbf{H}|^2+\epsilon^2} + B_0\mu_0^{-1}) \cdot \frac{\mathbf{H}}{\sqrt{|\mathbf{H}|^2+\epsilon^2}} \\ &= \mu_0\mathbf{H} + B_0 \cdot \frac{\mathbf{H}}{\sqrt{|\mathbf{H}|^2+\epsilon^2}}, \qquad (89) \\ D\mathbf{B}_\epsilon(\mathbf{H}) &= \mu_0 Id + B_0\left\{\frac{1}{\sqrt{|\mathbf{H}|^2+\epsilon^2}} \cdot Id - \frac{\mathbf{H}\cdot\mathbf{H}^T}{\sqrt{|\mathbf{H}|^2+\epsilon^2}^3}\right\}. \qquad (90) \end{aligned}$$

Für die Eigenwerte der Ableitung erhält man

$$\lambda_{max}(D\mathbf{B}_\epsilon(\mathbf{H})) = \mu_0 + B_0 \cdot \frac{1}{\sqrt{|\mathbf{H}|^2+\epsilon^2}} \leq \mu_0 + B_0\epsilon^{-1}, \tag{91}$$

$$\lambda_{min}(D\mathbf{B}_\epsilon(\mathbf{H})) = \mu_0 + B_0 \cdot \frac{1 - |\mathbf{H}|^2/(|\mathbf{H}|^2+\epsilon^2)}{\sqrt{|\mathbf{H}|^2+\epsilon^2}} \geq \mu_0. \tag{92}$$

In Analogie zu (80) soll nun gelten:

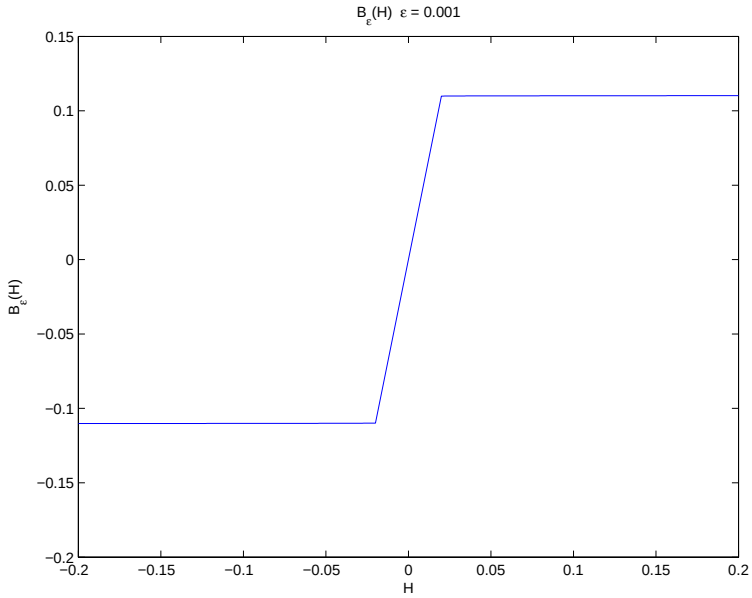

Abbildung 9: Regularisierter Zusammenhang $\mathbf{B}(\mathbf{H})$

$$\lambda_{max}(D\mathbf{B}_\epsilon(\mathbf{H})) \leq \tau^{-1} \quad \Leftrightarrow \quad \epsilon \geq \frac{B_0}{\tau^{-1} - \mu_0}. \tag{93}$$

Für eine Verallgemeinerung von (81) ergibt sich:

$$\begin{aligned}(\mathbf{B}_\epsilon(\mathbf{H}_1) - \mathbf{B}_\epsilon(\mathbf{H}_2)) \cdot (\mathbf{H}_1 - \mathbf{H}_2) &= \int_0^1 D\mathbf{B}_\epsilon(\mathbf{H}_2 + t(\mathbf{H}_1 - \mathbf{H}_2))(\mathbf{H}_1 - \mathbf{H}_2)dt \cdot (\mathbf{H}_1 - \mathbf{H}_2) \\ &= \int_0^1 (\mathbf{H}_1 - \mathbf{H}_2)^T D\mathbf{B}_\epsilon(\mathbf{H}_2 + t(\mathbf{H}_1 - \mathbf{H}_2))(\mathbf{H}_1 - \mathbf{H}_2)dt \\ &\geq \mu_0|\mathbf{H}_1 - \mathbf{H}_2|^2. \end{aligned} \tag{94}$$

Wegen

$$\mathbf{B}(\mathbf{H}) = \begin{cases} \mu_0\mathbf{H} + B_0\frac{\mathbf{H}}{|\mathbf{H}|} & f\ddot{u}r \quad \mathbf{H} \neq 0 \\ 0 & f\ddot{u}r \quad \mathbf{H} = 0 \end{cases} \tag{95}$$

gilt

$$\mathbf{B}(\mathbf{H}) \cdot \mathbf{H} = \mu_0|\mathbf{H}|^2 + B_0|\mathbf{H}|, \qquad \int_0^1 \mathbf{B}(t\mathbf{H}) \cdot \mathbf{H}dt = \frac{1}{2}\mu_0|\mathbf{H}|^2 + B_0|\mathbf{H}|. \tag{96}$$

Daher

$$|\mathbf{H}_\epsilon(\mathbf{H})|^2 = \mu_0^2|\mathbf{H}|^2 + 2B_0\mu_0 \cdot \frac{|\mathbf{H}|^2}{\sqrt{|\mathbf{H}|^2 + \epsilon^2}} + B_0^2 \cdot \frac{|\mathbf{H}|^2}{|\mathbf{H}|^2 + \epsilon^2} \geq \mu_0^2|H|^2, \tag{97}$$

was (82) entspricht.

Schliesslich erhält man das folgende Zeitschritt-Verfahren, welches $\mathbf{H}_i$ aus $\mathbf{H}_{i-1}$ berechnet:

$$\mathbb{L}_i(\mathbf{H}_i - \mathbf{H}_{i-1}) + \tau \mathbf{rot}\sigma^{-1}\mathbf{rot}\mathbf{H}_i = \mathbf{rot}(\sigma^{-1}\mathbf{j_0}(i\tau)) \qquad in \quad \Omega, \tag{98}$$

$$\left\| \mathbb{L}_i - \gamma \int_0^1 D\mathbf{B}_\epsilon(\mathbf{H}_{i-1} + t\gamma(\mathbf{H}_i - \mathbf{H}_{i-1}))dt \right\| < \tau. \tag{99}$$

Dis ist ein nichtlineares Gleichungssystem für $\mathbf{H}_i$ und $\mathbb{L}_i$, welches nur durch eine Fixpunkt Iteration gelöst werden kann. Das Iterationsschema lautet wie folgt:
Setze $\mathbb{L}_i^{(0)} := D\mathbf{B}_\epsilon(\mathbf{H}_{i-1})$

$$\begin{aligned}
\widetilde{\mathbf{H}}_i \quad &: \quad \mathbb{L}_i^{(j)}(\widetilde{\mathbf{H}}_i - \mathbf{H}_{i-1}) + \tau \mathbf{rot}\sigma^{-1}\mathbf{rot}\widetilde{\mathbf{H}}_i = \mathbf{rot}(\sigma^{-1}\mathbf{j_0}(i\tau)) \qquad in \quad \Omega, \\
\widetilde{\mathbb{L}}_i \quad &: \quad \widetilde{\mathbb{L}}_i = \gamma \int_0^1 D\mathbf{B}_\epsilon(\mathbf{H}_{i-1} + t\gamma(\widetilde{\mathbf{H}}_i - \mathbf{H}_{i-1}))dt, \\
\mathbb{L}_i^{j+1} \quad &:= min\left\{1, \frac{K}{\lambda_{max}(\widetilde{\mathbb{L}}_i)}\right\} \cdot \widetilde{\mathbb{L}}_i, \qquad für \quad ein \quad K > 0.
\end{aligned}$$

Diese Iteration wird solange widerholt, bis

$$\left\| \mathbb{L}_i^{(j+1)} - \mathbb{L}_i^{(i)} \right\| < \frac{1}{2}\tau. \tag{100}$$

Die auftauchenden Integrale $\int_0^1 ...dt$ können mittels einer Trapezregel angenähert werden. Als Updateformel dient die Gleichung

$$\mathbf{B}(\mathbf{H}_i) = \mathbf{B}(\mathbf{H_{i-1}}) + \mathbb{L}_\mathbf{i}(\widetilde{\mathbf{H}}_\mathbf{i} - \mathbf{H_{i-1}}), \tag{101}$$

welche mit (95) nach $\mathbf{H}_i$ aufgelöst werden kann.

4.2.4 Die Matrixgleichung

Die Matrixgleichung von Gleichung (98) sieht wegen der Matrix $\mathbb{L}_i$ etwas anders aus als Gleichung (58). Insbesondere kann nicht mehr $\mathbb{B}^e$ als lokale Elementmatrix verwendet werden. Die neue Matrix $\mathbb{D}^e$ lässt sich wie folgt berechen:

$$\mathbb{D}_{ij} := \int_A \mathbb{L}(\mathbf{N}_i^e)\mathbf{N}_j^e dA \tag{102}$$

wobei $\mathbb{L} = \begin{pmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{pmatrix}$ eine beliebige Matrix ist. Konkret ergibt sich für die einzelnen Einträge:

$$
\begin{aligned}
\mathbb{D}_{11} &= \frac{l_{11}}{192A}(2b_2^2 - 2b_1b_2 + 2b_1^2) + \frac{l_{12}}{192A}(2b_2c_2 - b_1c_2 - b_2c_1 + 2b_1c_1) \\
&+ \frac{l_{21}}{192A}(2b_2c_2 - b_2c_1 - b_1c_2 + 2b_1c_1) + \frac{l_{22}}{192A}(2c_2^2 - 2c_1c_2 + 2c_1^2)
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{D}_{12} &= \frac{l_{11}}{192A}(b_2b_3 - b_2^2 - 2b_1b_3 + b_1b_2) + \frac{l_{12}}{192A}(b_3c_2 - b_2c_2 - 2b_3c_1 + b_2c_1) \\
&+ \frac{l_{21}}{192A}(b_2c_3 - b_2c_2 - 2b_1c_3 + b_1c_2) + \frac{l_{22}}{192A}(c_2c_3 - c_2^2 - 2c_1c_3 + c_1c_2)
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{D}_{13} &= \frac{l_{11}}{192A}(b_1b_2 - b_1^2 - 2b_2b_3 + b_1b_3) + \frac{l_{12}}{192A}(b_1c_2 - b_1c_1 - 2b_3c_2 + b_3c_1) \\
&+ \frac{l_{21}}{192A}(b_2c_1 - b_1c_1 - 2b_2c_3 + b_1c_3) + \frac{l_{22}}{192A}(c_1c_2 - c_1^2 - 2c_2c_3 + c_1c_3)
\end{aligned}
$$

$$
\mathbb{D}_{21} = \mathbb{D}_{12}
$$

$$
\begin{aligned}
\mathbb{D}_{22} &= \frac{l_{11}}{192A}(2b_3^2 - 2b_2b_3 + 2b_2^2) + \frac{l_{12}}{192A}(2b_3c_3 - b_2c_3 - b_3c_2 + 2b_2c_2) \\
&+ \frac{l_{21}}{192A}(2b_3c_3 - b_3c_2 - b_2c_3 + 2b_2c_2) + \frac{l_{22}}{192A}(2c_3^2 - 2c_2c_3 + 2c_2^2)
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{D}_{23} &= \frac{l_{11}}{192A}(b_1b_3 - b_3^2 - 2b_1b_2 + b_2b_3) + \frac{l_{12}}{192A}(b_1c_3 - b_3c_3 - 2b_1c_2 + b_3c_2) \\
&+ \frac{l_{21}}{192A}(b_3c_1 - b_3c_3 - 2b_2c_1 + b_2c_3) + \frac{l_{22}}{192A}(c_1c_3 - c_3^2 - 2c_1c_2 + c_2c_3)
\end{aligned}
$$

$$
\mathbb{D}_{31} = \mathbb{D}_{13}
$$

$$
\mathbb{D}_{32} = \mathbb{D}_{23}
$$

$$
\begin{aligned}
\mathbb{D}_{33} &= \frac{l_{11}}{192A}(2b_1^2 - 2b_1b_3 + 2b_3^2) + \frac{l_{12}}{192A}(2b_1c_1 - b_3c_1 - b_1c_3 + 2b_3c_3) \\
&+ \frac{l_{21}}{192A}(2b_1c_1 - b_1c_3 - b_3c_1 + 2b_3c_3) + \frac{l_{22}}{192A}(2c_1^2 - 2c_1c_3 + 2c_3^2)
\end{aligned}
$$

Als Matrixversion von Gleichung (98) ergibt sich schliesslich:

$$
\mathbb{D}(\mathbf{H}_i - \mathbf{H}_{i-1}) + \tau\sigma^{-1}\mathbb{A}\mathbf{H}_i = \tau\sigma^{-1}\mathbb{C}. \tag{103}
$$

5 Resultate

5.1 Fixpunkt Iteration

5.1.1 Anzahl Iterationen bis zur Konvergenz

Im Folgenden werden einige Resultate für den „Erregerstrom“ $\mathbf{j}_0 = 1000y^2 \sin(\frac{2\Pi t}{T})$,wo T der Endzeit entspricht, besprochen. Als Anfangsbedingung wurde die Lösung des stationären Problems genommen. Es wurden für verschiedene Zeitschritte τ und Ortsdiskretisierung dx Testläufe durchgeführt, um die Anzahl der Iterationen bis zur Konvergenz zu messen. Die ersten vier Abbildungen zeigen den Absolutbetrag des Feldes $\mathbf{H}$ zu vier verschiedenen Zeitpunkten.

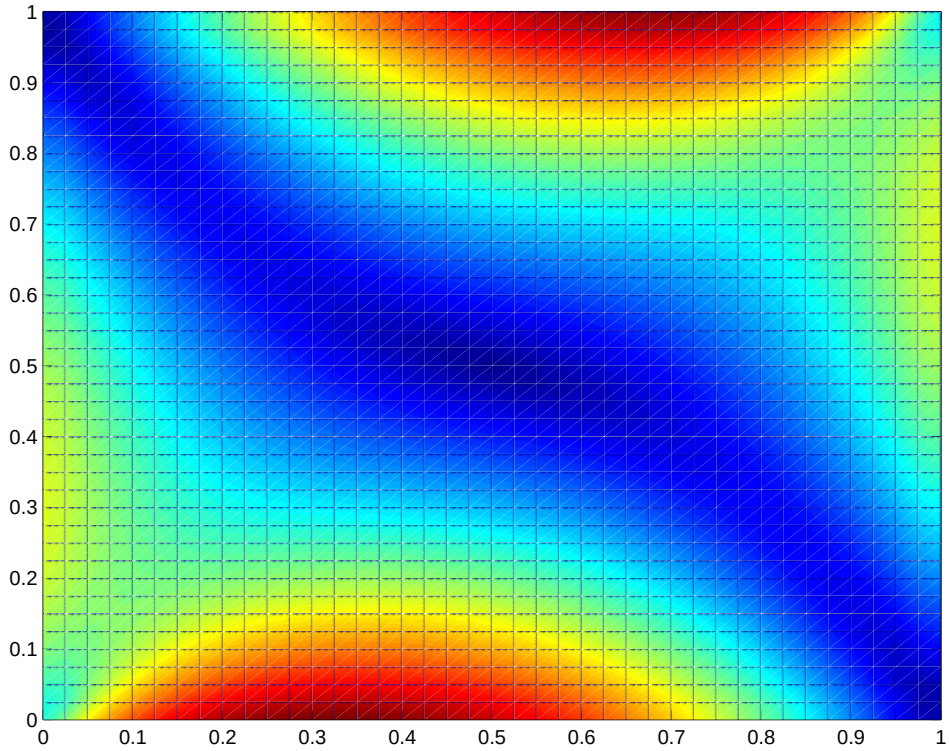

Abbildung 10: $|\mathbf{H}|$ bei $T = 0$

Aus den Resultaten können folgende Schlüsse gezogen werden:

- Der Zeitschritt τ kann nicht beliebig gross gewählt werden. Wird er zu gross, dann konvergiert das Iterationsschema nicht mehr. Es scheint, dass diese Tatsache in direktem Zusammenhang mit der Feinheit der Ortsdiskretisierung steht. Für $dx = 0.1$ muss $\tau \leq 0.005$ sein, damit es zur Konvergenz kommt. Für $dx = 0.05$ konvergiert das Verfahren nur für $\tau \leq 0.01$ und für $dx = 0.02$, gilt $\tau \leq 0.1$.

- Bei einer feineren Ortsdiskretisierung steigt die Anzahl der benötigten Iterationen bis zur Konvergenz.

- Dass bei der ersten und der letzten Iteration jeweils am wenigsten Iterationen notwendig sind, hängt mit der Wahl des Erregerstrom als sinusförmig zusammen. Die Testläufe liefen jeweils genau über eine Periode T. Dies erklärt auch die gewisse Symmetrie bezüglich der x-Achsen im Punkt $\frac{T}{2}$

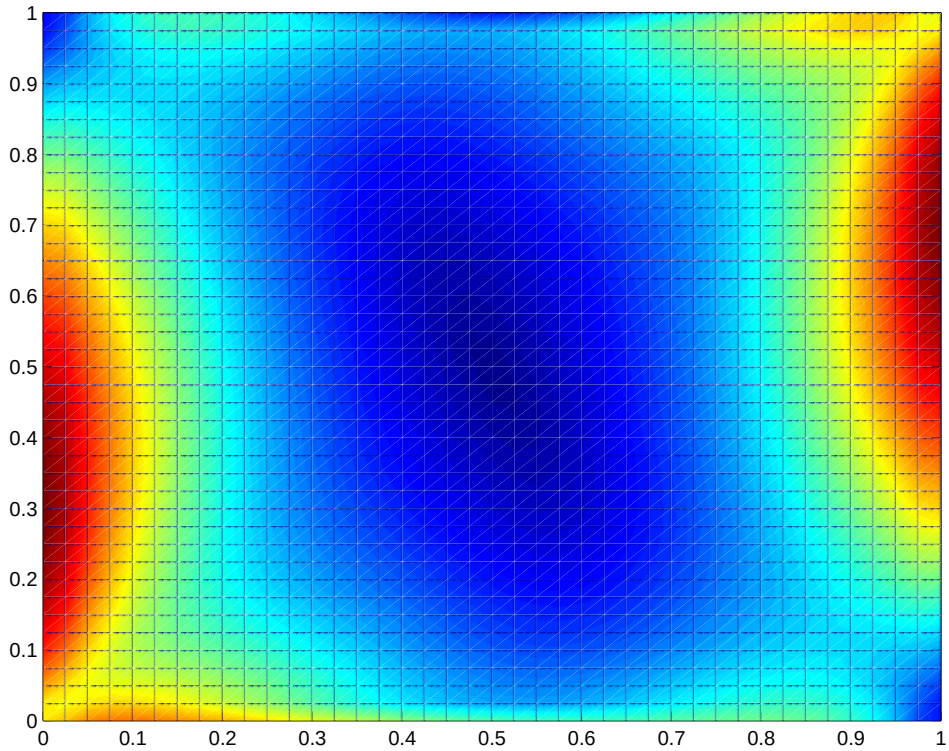

Abbildung 11: $|\mathbf{H}|$ nach $\frac{1}{4}T$

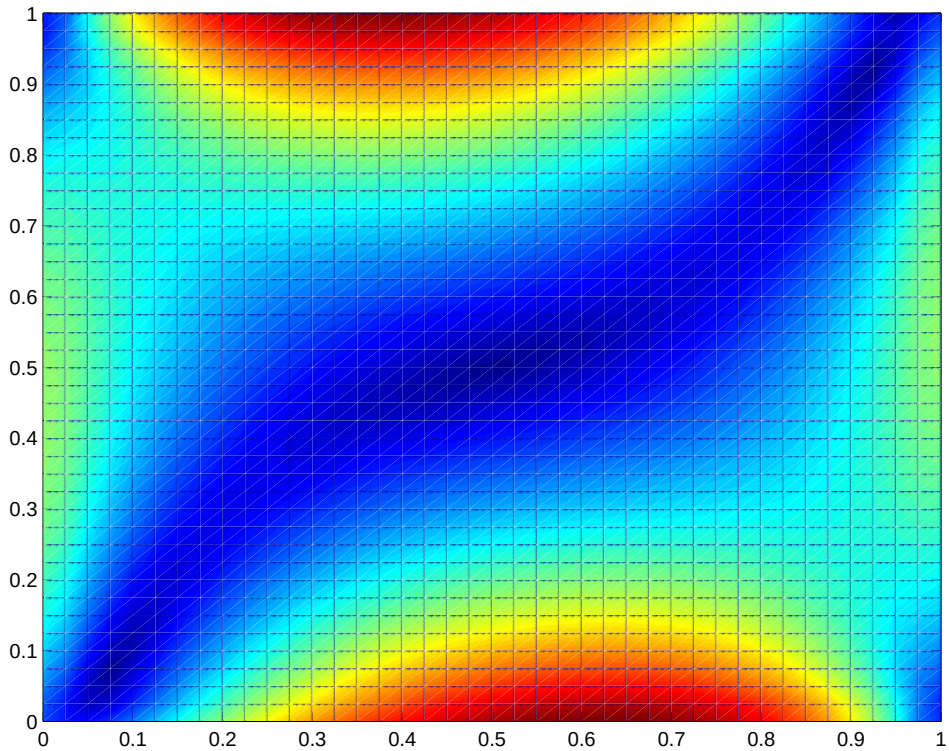

Abbildung 12: $|\mathbf{H}|$ nach $\frac{1}{2}T$

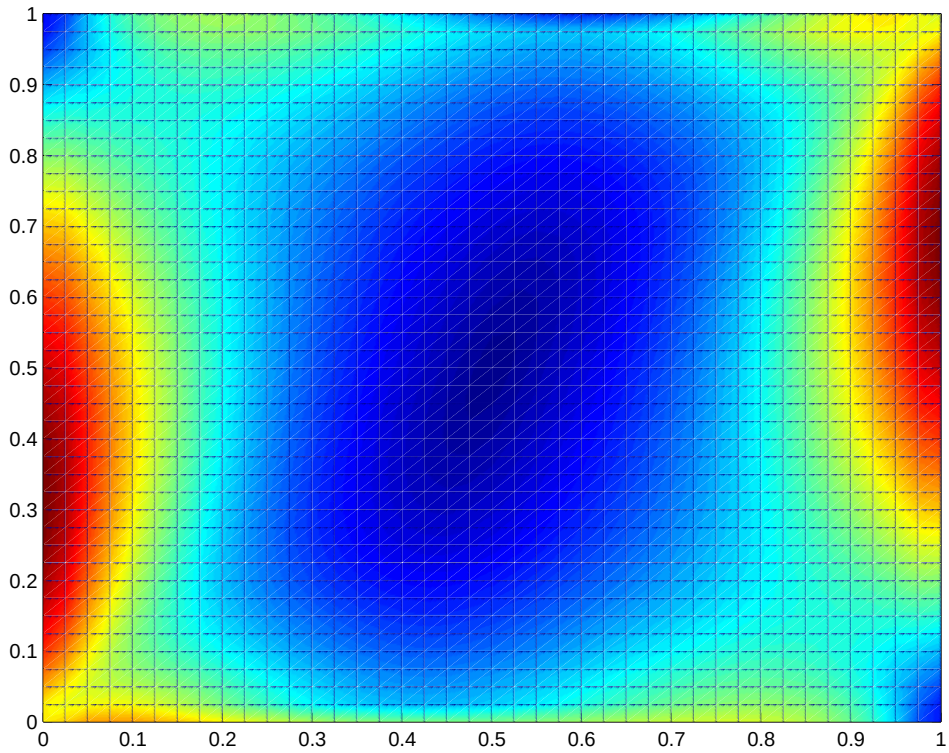

Abbildung 13: $|\mathbf{H}|$ nach $\frac{3}{4}T$

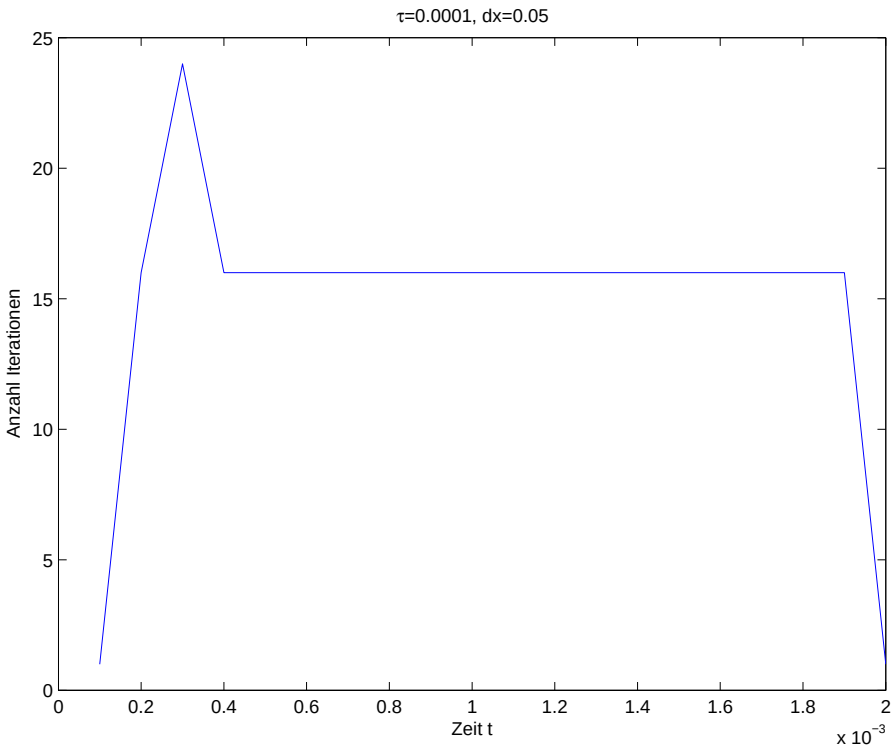

Abbildung 14: Anzahl Iterationen $\tau = 0.0001$ $dx = 0.05$

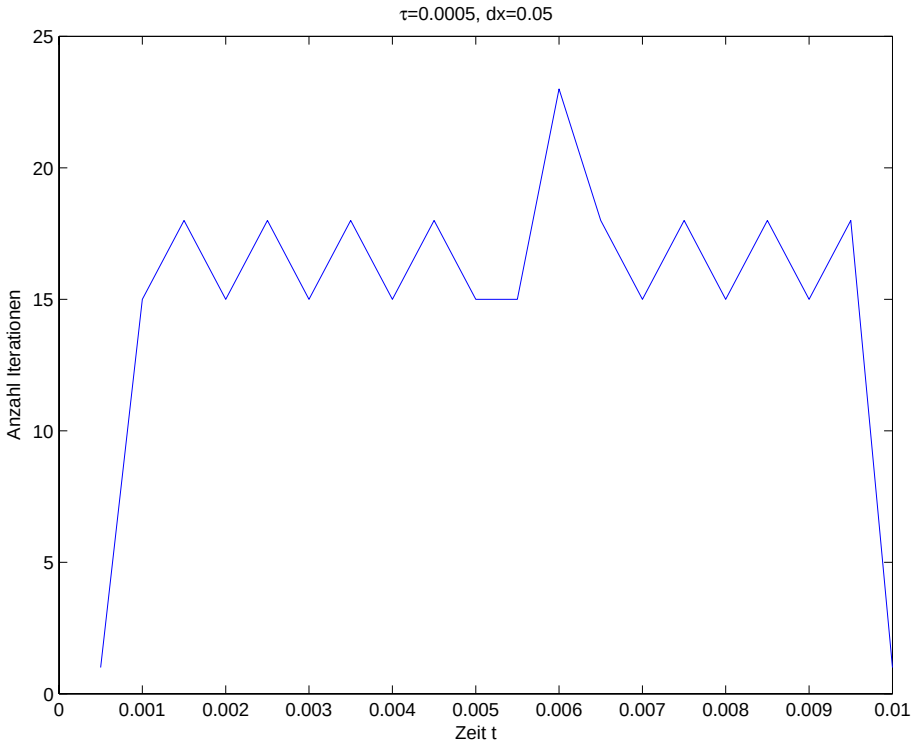

Abbildung 15: Anzahl Iterationen $\tau = 0.0005$ $dx = 0.05$

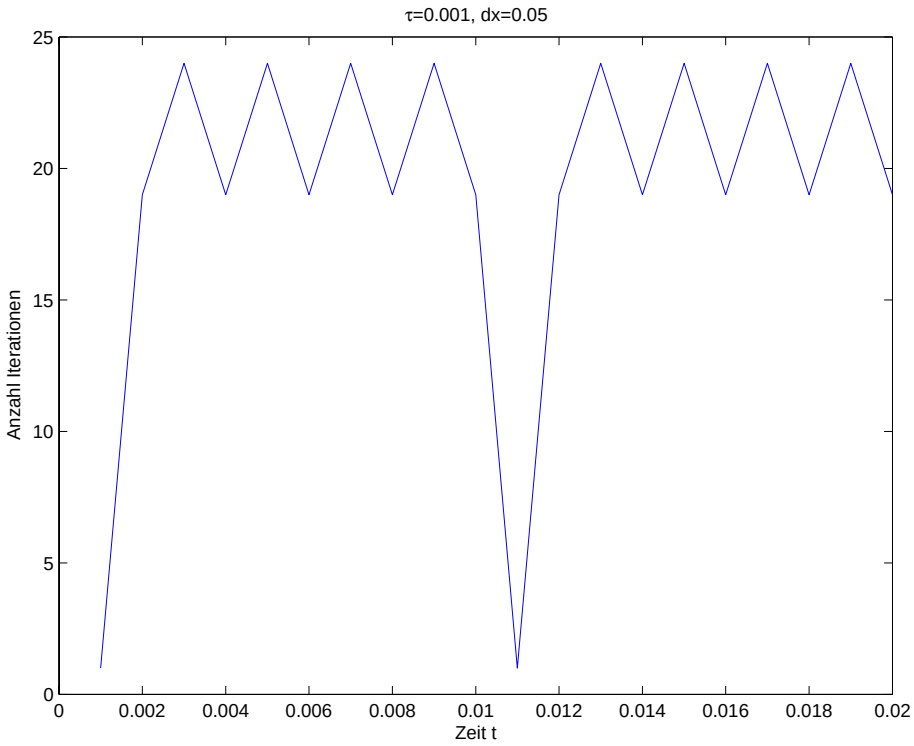

Abbildung 16: Anzahl Iterationen $\tau = 0.001$ $dx = 0.05$

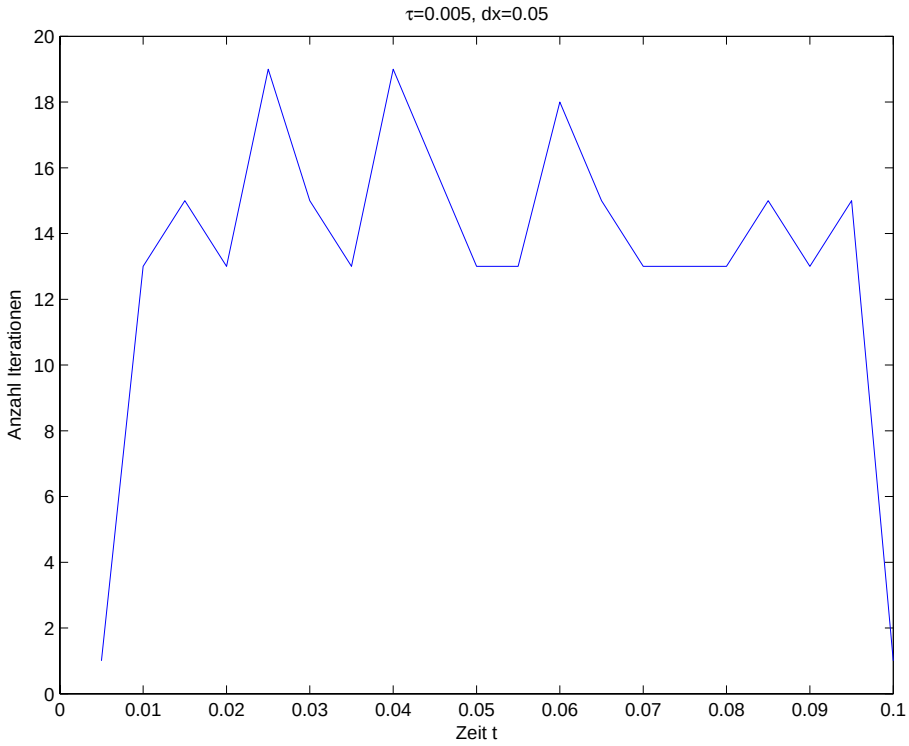

Abbildung 17: Anzahl Iterationen $\tau = 0.005$ $dx = 0.05$

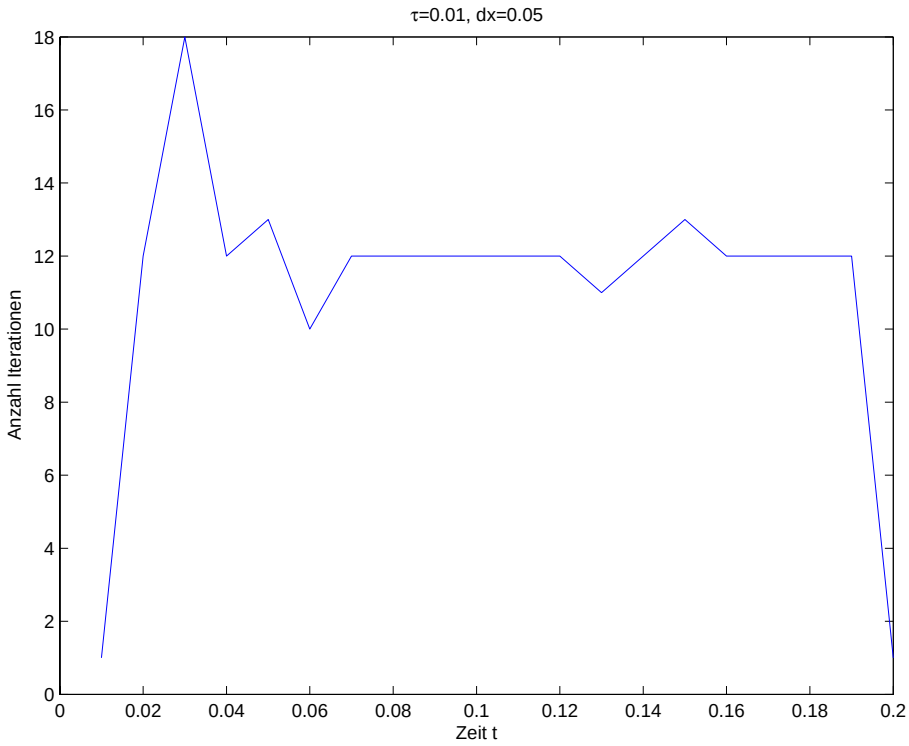

Abbildung 18: Anzahl Iterationen $\tau = 0.01$ $dx = 0.05$

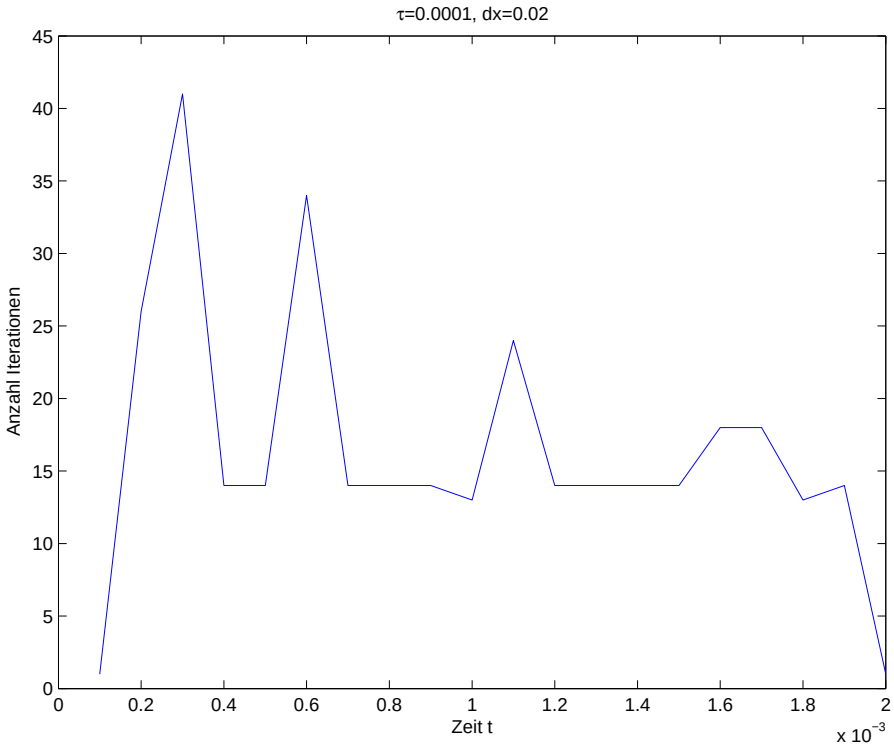

Abbildung 19: Anzahl Iterationen $\tau = 0.0001$ $dx = 0.02$

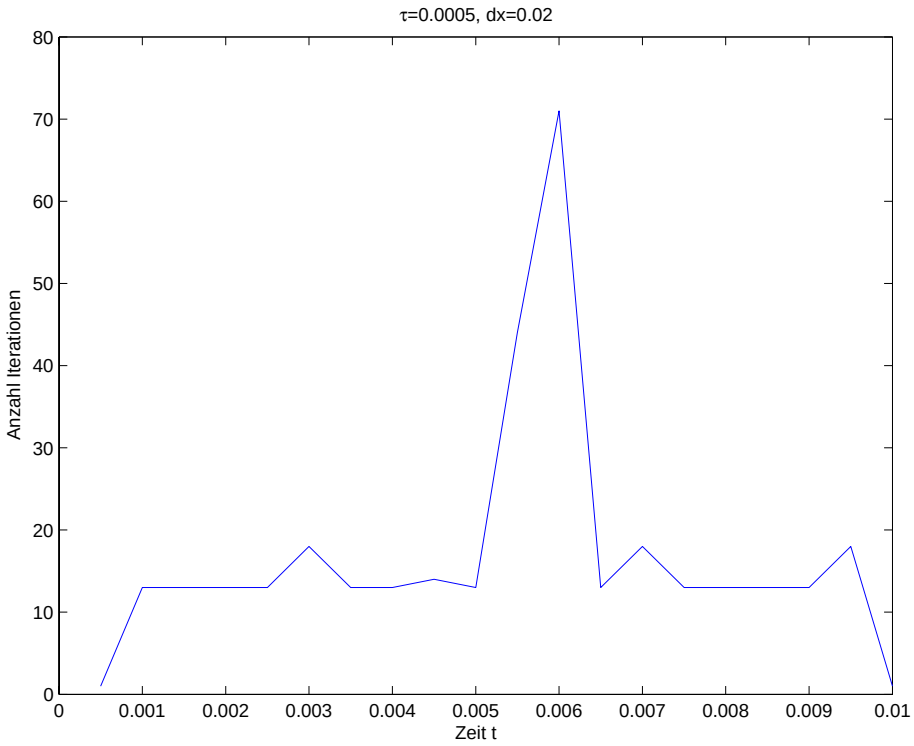

Abbildung 20: Anzahl Iterationen $\tau = 0.0005$ $dx = 0.02$

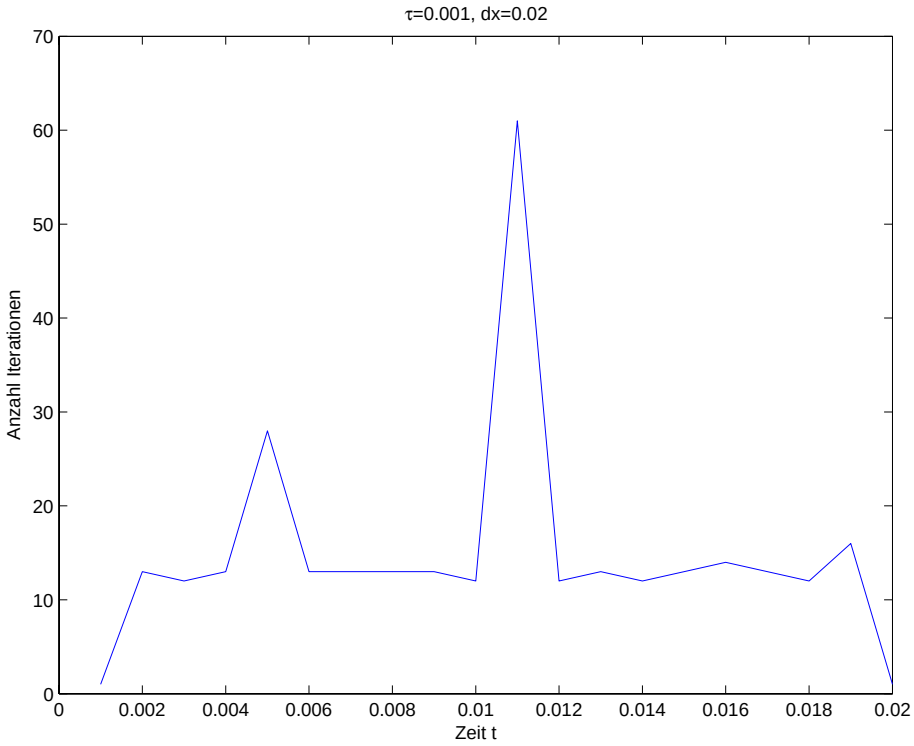

Abbildung 21: Anzahl Iterationen $\tau = 0.001$ $dx = 0.02$

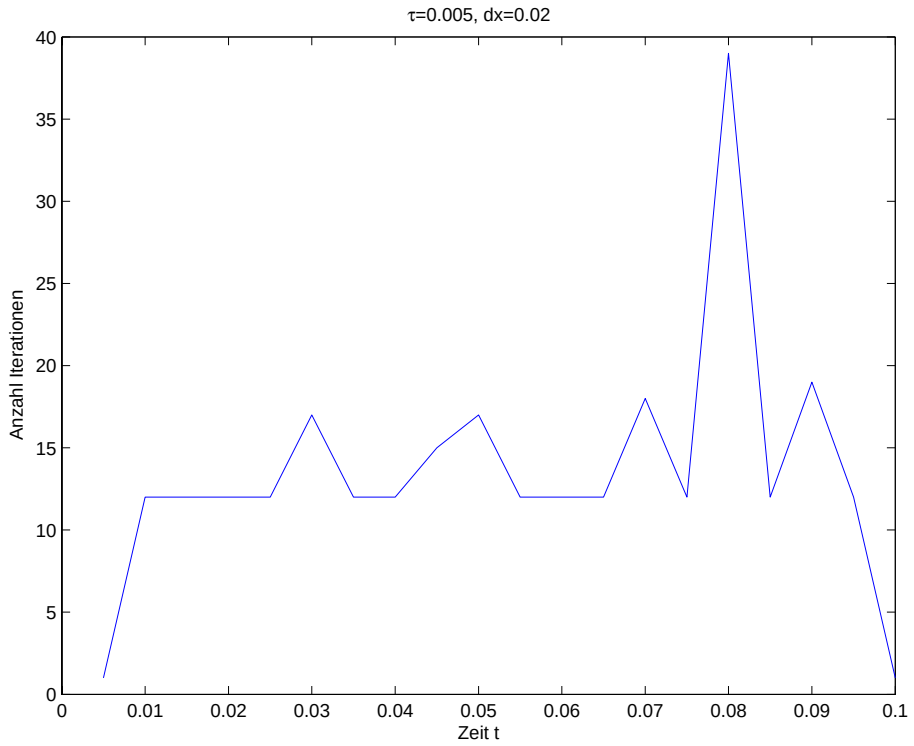

Abbildung 22: Anzahl Iterationen $\tau = 0.005$ $dx = 0.02$

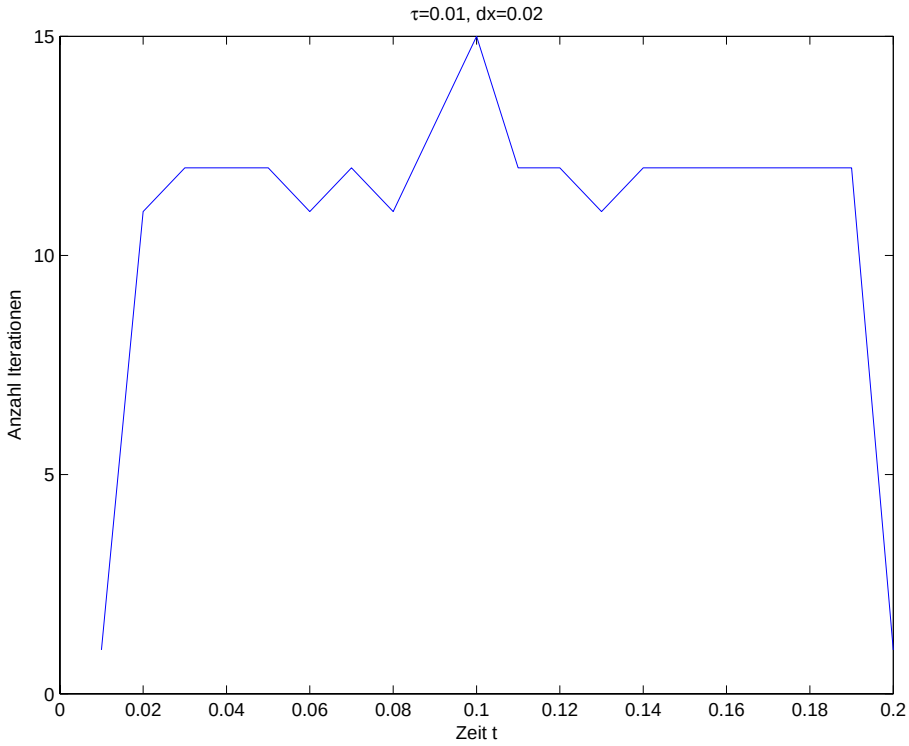

Abbildung 23: Anzahl Iterationen $\tau = 0.01$ $dx = 0.02$

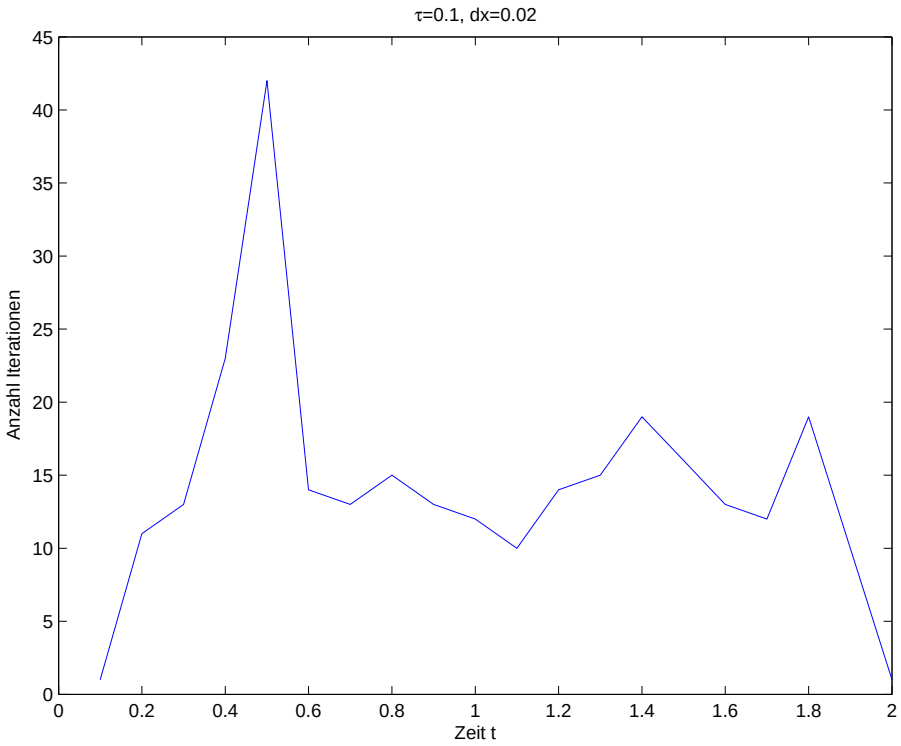

Abbildung 24: Anzahl Iterationen $\tau = 0.1$ $dx = 0.02$

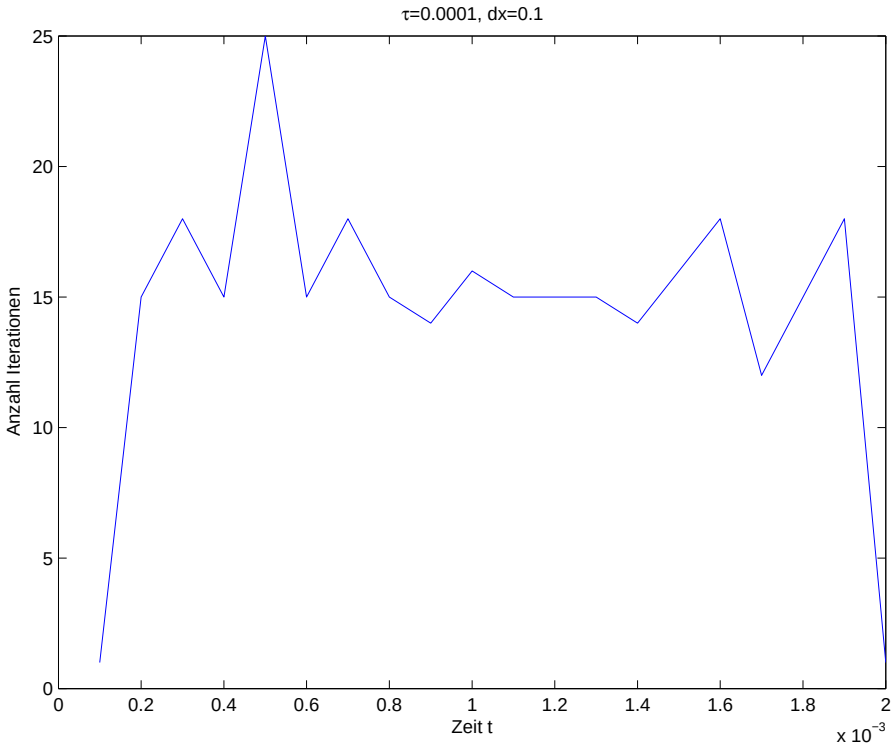

Abbildung 25: Anzahl Iterationen $\tau = 0.0001$ $dx = 0.1$

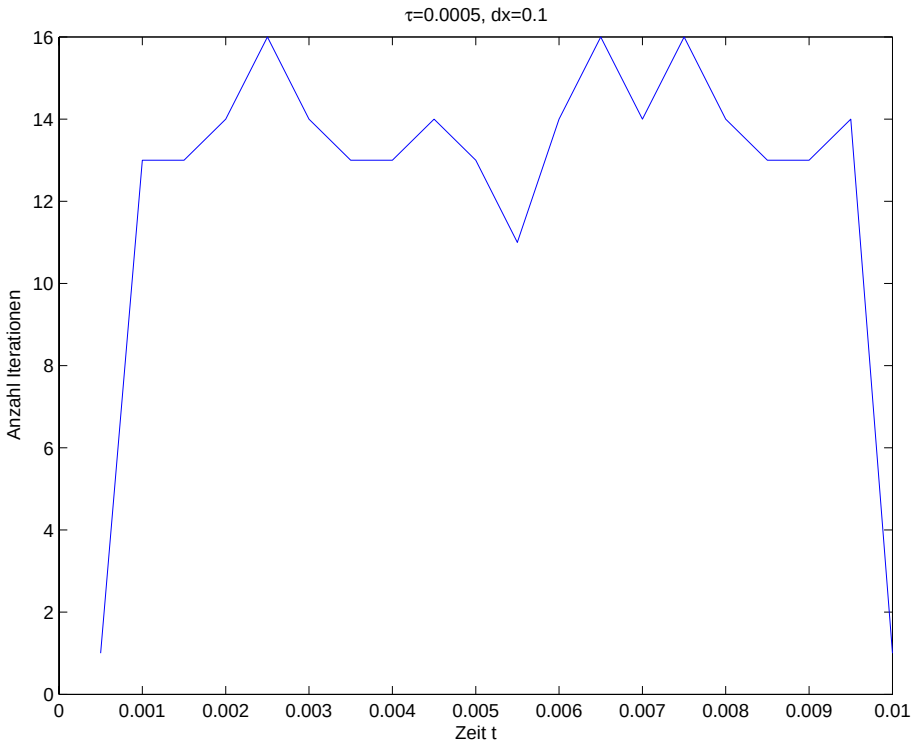

Abbildung 26: Anzahl Iterationen $\tau = 0.0005$ $dx = 0.1$

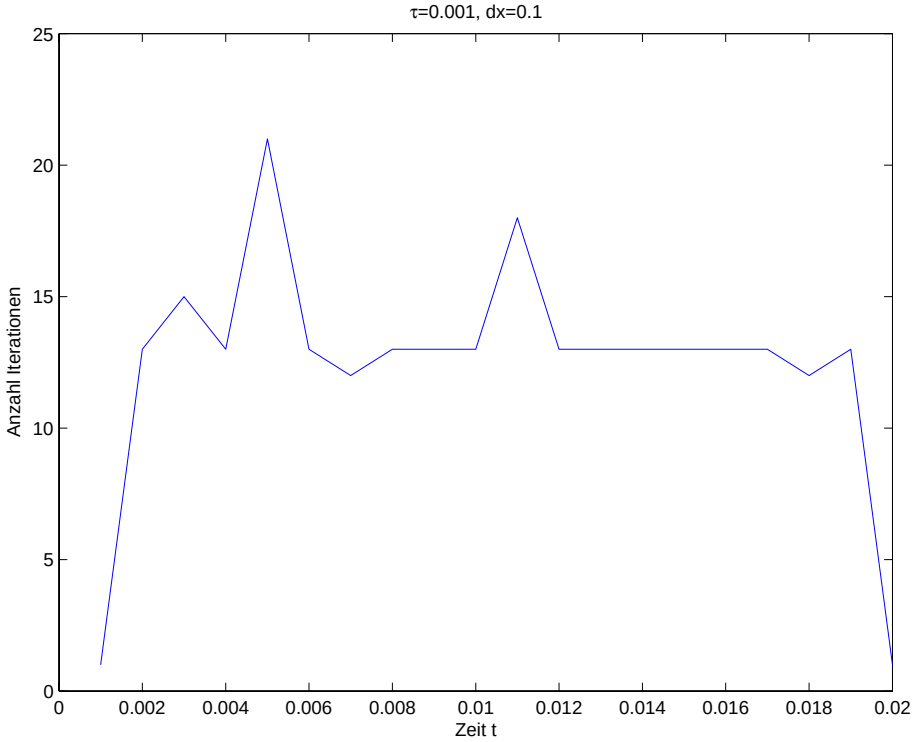

Abbildung 27: Anzahl Iterationen $\tau = 0.001$ $dx = 0.1$

5.1.2 Konvergenztests

Um die Konvergenz des Fixpunktiterationsverfahrens zu testen wurde für verschiedene Zeit- und Ortsdiskretisierungen die magnetische Energie

$$\int_{\Omega} \mathbf{H}^2 dx \tag{104}$$

über der ferromagnetischen Platte berechnet. (vgl. Abb.29-Abb.31). Als „Erregerstrom“ wurde wiederum $\mathbf{j}_0 = 1000y^2 \sin(\frac{2\Pi t}{T})$,wo T der Endzeit entspricht, gewählt. Als Anfangsbedingung wurde die Lösung des stationären Problems genommen.

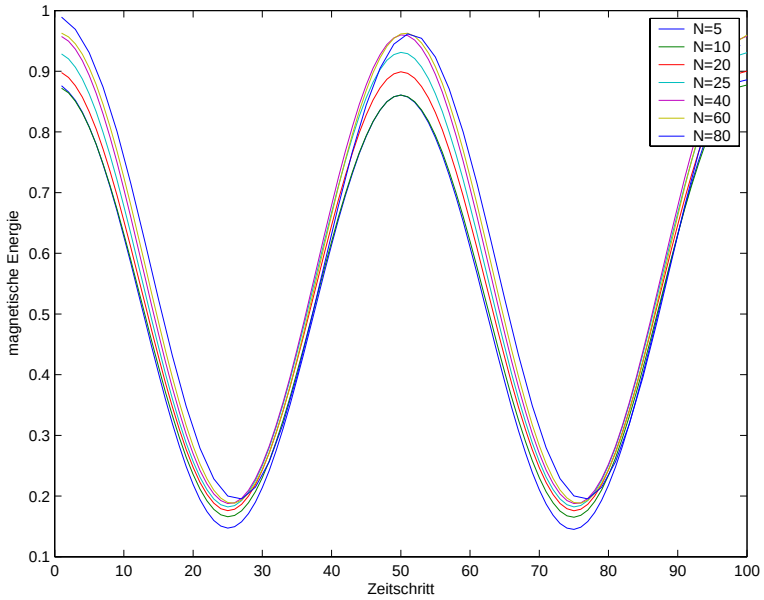

Abbildung 29: Magnetische Energie bei verschiedenen Ortsdiskretisierungen für $\tau = 0.0001$ (normiert auf 1)

Aus den Resultaten können folgende Schlüsse gezogen werden:

- Die magnetische Energie scheint bei besser werdender Ortsdiskretisierung gegen einen bestimmten Wert zu konvergieren. Dies lässt darauf schliessen, dass das Verfahren stabil ist und konvergiert.
- Konvergenz tritt erwartungsgemäss schneller ein, wenn die Zeitdiskretisierung kleiner ist.
- Für $\tau = 0.01$ konvergiert das Verfahren nur für $N \geq 20$.

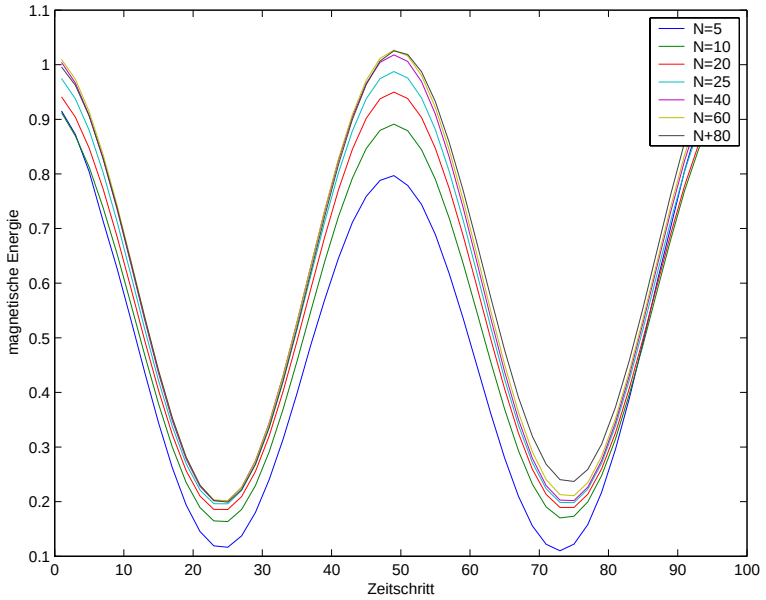

Abbildung 30: Magnetische Energie bei verschiedenen Ortsdiskretisierungen für $\tau = 0.001$ (normiert auf 1)

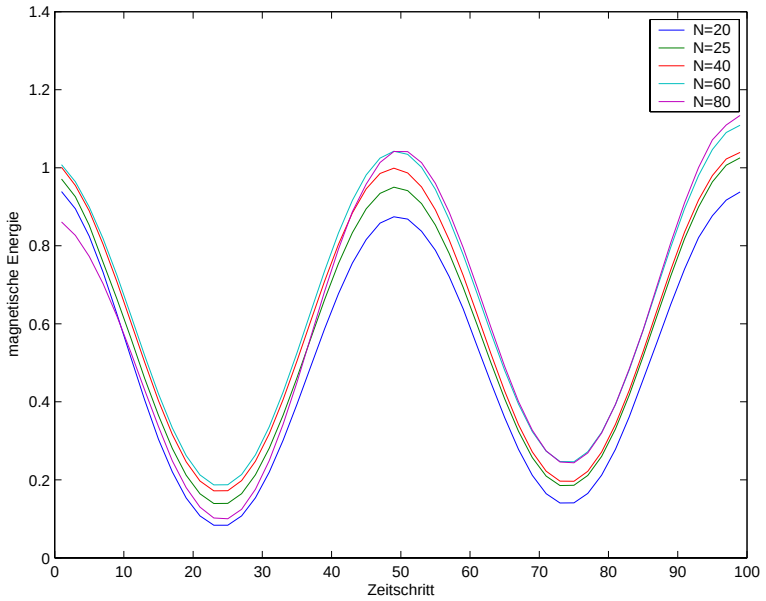

Abbildung 31: Magnetische Energie bei verschiedenen Ortsdiskretisierungen für $\tau = 0.01$ (normiert auf 1)

A Literatur

1. J. Kacur, *Solution of some free boundary problems by relaxation schemes*, SIAM J. Numer, Anal., 36 (1998), pp. 290-316.

2. J. Kacur, *Solution to strongly nonlinear parabolic problems by a linear approximation scheme*, IMA J. Numer. Anal., 19 (1999), pp. 119-145.

3. R. Hiptmair, Vorlesungsnotizen zur Vorlesung: *Computational Electromagnetism.*

4. R. Hiptmair, Vorlesungsnotizen zur Vorlesung: *Numerik der Differentialgleichungen*

B Matlab Code

Der Matlab Code besteht aus folgenden Modulen:

- **B_H_Solver**: Updateformel in Relaxationsverfahren
- **FEM_Euler**: In diesem Modul wurde ein implizites Zeitschema implementiert für einen einfachen linearen Zusammenhang zwischen **H** und **B**. Es diente hauptsächlich zu Testzwecken.
- **FEM_Fixpkt**: In diesem Modul wurde das Fixpunktiterations-Verfahren implementiert.
- **FEM_Relaxation**: Implementierung des Relaxationsverfahren.
- **grid1**: In diesem Modul wird das Dreiecksgitter aufgebaut und die Datenstruktur für Kanten, Knoten und Dreicke erstellt.
- **interpolate**: Berechnet ein interpoliertes Feld über den Dreieckskanten.
- **interpolate_triangles**: Berechnet ein interpoliertes Feld im Dreiecksschwerpunkt.
- **j0**: Erregerstrom.
- **Jacobian_B**: Berechnet die Jacobi-Matrix für das Relaxationsverfahren.
- **local_A**: lokale Elementmatrix $\mathbb{A}$
- **local_B**: lokale Elementmatrix $\mathbb{B}$
- **local_C**: lokaler Elementvektor $\mathbb{C}$
- **local_L**: lokale Elementmatrix $\mathbb{D}$
- **plot_abs**: Für die graphische Darstellung des Feldbetrages.
- **plot_int**: Für die graphische Darstellung mit Pfeilen.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% B_H_Solver.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function H = B_H_Solver(rhs_update, mu0, B0, epsilon)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet Update von H beim Relaxationsverfahren.
%
% Input-Parameter:
%
% -rhs_update: rechte Seite aus Updateformel
% -mu0: permeabilitaet
% -B0: Regularisierungsparameter%
% -epsilon: Regularisierungsparameter
%
% Output-Parameter:
%
% -H: Feld zum neuen Zeitpunkt
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k=1:size(rhs_update,1)
    Habs=abs((abs(rhs_update(k,1))-B0)/mu0);
    H(k,1)=rhs_update(k,1)/(mu0+B0/Habs);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FEM_Euler.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FEM Loesungsverfahren fuer einfachen linearen Zusammenhang
% zwischen B und H.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d=1;
%Ortsdiskretisierung
N=20;

%Gittergenerierung
[nodes,edges,triangle_nodes,triangle_edges]= grid1(d,N);

number_of_edges=2*N*(N+1)+N*N;
number_of_triangles=N*N*2;

%Allocate memory
A=sparse(number_of_edges,number_of_edges);
B=sparse(number_of_edges,number_of_edges);
C=sparse(number_of_edges,1);

%Assemblierung
for j=1:number_of_triangles
    p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
    p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
    p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

    %Global A
    a=local_A(j,N,p1,p2,p3);
    A(triangle_edges(j,2),triangle_edges(j,2))=A(triangle_edges(j,2),triangle_edges(j,2))+a(1,1);
    A(triangle_edges(j,2),triangle_edges(j,3))=A(triangle_edges(j,2),triangle_edges(j,3))+a(1,2);
    A(triangle_edges(j,2),triangle_edges(j,4))=A(triangle_edges(j,2),triangle_edges(j,4))+a(1,3);
    A(triangle_edges(j,3),triangle_edges(j,2))=A(triangle_edges(j,3),triangle_edges(j,2))+a(2,1);
    A(triangle_edges(j,3),triangle_edges(j,3))=A(triangle_edges(j,3),triangle_edges(j,3))+a(2,2);
    A(triangle_edges(j,3),triangle_edges(j,4))=A(triangle_edges(j,3),triangle_edges(j,4))+a(2,3);
    A(triangle_edges(j,4),triangle_edges(j,2))=A(triangle_edges(j,4),triangle_edges(j,2))+a(3,1);
    A(triangle_edges(j,4),triangle_edges(j,3))=A(triangle_edges(j,4),triangle_edges(j,3))+a(3,2);
    A(triangle_edges(j,4),triangle_edges(j,4))=A(triangle_edges(j,4),triangle_edges(j,4))+a(3,3);

    %Global C
    t=0;
    c=local_C(j,N,p1,p2,p3,t);
    C(triangle_edges(j,2),1)=C(triangle_edges(j,2),1)+c(1,1);
    C(triangle_edges(j,3),1)=C(triangle_edges(j,3),1)+c(2,1);
    C(triangle_edges(j,4),1)=C(triangle_edges(j,4),1)+c(3,1);

    %Global B
    b=local_B(j,N,p1,p2,p3);
    B(triangle_edges(j,2),triangle_edges(j,2))=B(triangle_edges(j,2),triangle_edges(j,2))+b(1,1);
    B(triangle_edges(j,2),triangle_edges(j,3))=B(triangle_edges(j,2),triangle_edges(j,3))+b(1,2);
    B(triangle_edges(j,2),triangle_edges(j,4))=B(triangle_edges(j,2),triangle_edges(j,4))+b(1,3);
    B(triangle_edges(j,3),triangle_edges(j,2))=B(triangle_edges(j,3),triangle_edges(j,2))+b(2,1);
    B(triangle_edges(j,3),triangle_edges(j,3))=B(triangle_edges(j,3),triangle_edges(j,3))+b(2,2);
    B(triangle_edges(j,3),triangle_edges(j,4))=B(triangle_edges(j,3),triangle_edges(j,4))+b(2,3);
    B(triangle_edges(j,4),triangle_edges(j,2))=B(triangle_edges(j,4),triangle_edges(j,2))+b(3,1);
    B(triangle_edges(j,4),triangle_edges(j,3))=B(triangle_edges(j,4),triangle_edges(j,3))+b(3,2);
    B(triangle_edges(j,4),triangle_edges(j,4))=B(triangle_edges(j,4),triangle_edges(j,4))+b(3,3);
end

%Movie Initialisierung
fig=figure;
set(fig,'DoubleBuffer','on');
set(gca,'xlim',[-80 80],'ylim',[-80 80],...
```

```
      'NextPlot','replace','Visible','off')
mov = avifile('test1.avi')

%Anfangsbedingung=Loesg. des stationaeren Problems
Hstart=bicgstab(A,C);
%Plot Hstart Feld
intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hstart,edges);
h=plot_abs(intfield,N);
shading interp
colormap jet
colorbar

set(h,'EraseMode','xor');
F = getframe(gca);
mov = addframe(mov,F);

%Speicherresevierung
Hold=zeros(number_of_edges,1);
Hnew=zeros(number_of_edges,1);

%Zeitschritt dt
dt=0.005;
%Konstanten
s=10^(-7);
sinverse=1/s;
mu0=0.01;

%EndZeit
T=0.15;
%Anzahl steps
n=T/dt;

Hold=Hstart;
for i=1:n
   %Berechne zeitabhaengige rechte Seite
   C=zeros(number_of_edges,1);

   for j=1:number_of_triangles
      t=i*dt;
      p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
      p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
      p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

      c=local_C(j,N,p1,p2,p3,t);
      C(triangle_edges(j,2),1)=C(triangle_edges(j,2),1)+c(1,1);
      C(triangle_edges(j,3),1)=C(triangle_edges(j,3),1)+c(2,1);
      C(triangle_edges(j,4),1)=C(triangle_edges(j,4),1)+c(3,1);
   end
   rs=dt*sinverse*C+B*(mu0*Hold);
   ls=mu0*B+dt*sinverse*A;
   %Loese Gleichungssystem
   Hnew=bicgstab(ls,rs);
   Hold=Hnew;

   %graphische Darstellung des Feldes
   intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hnew,edges);
   h=plot_abs(intfield,N);
   shading interp
   colormap jet
   colorbar
   hold on
   set(h,'EraseMode','xor');
   F = getframe(gca);
   mov = addframe(mov,F);
end
mov = close(mov);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FEM_Fixpkt.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FEM mit Fixpunktiteration
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Gitterlaenge
d=1;
%Ortsdiskretisierung
N=20;

%Gittergenerierung
[nodes,edges,triangle_nodes,triangle_edges]= grid1(d,N);

number_of_edges=2*N*(N+1)+N*N;
number_of_triangles=N*N*2;

%Speicher Reservation
A=sparse(number_of_edges,number_of_edges);
B=sparse(number_of_edges,number_of_edges);
C=sparse(number_of_edges,1);

%postprocs data
data=zeros(20,2);

%Assemblierung
for j=1:number_of_triangles
   p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
   p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
   p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

   %Global A
   a=local_A(j,N,p1,p2,p3);

   A(triangle_edges(j,2),triangle_edges(j,2))=A(triangle_edges(j,2),triangle_edges(j,2))+a(1,1);
   A(triangle_edges(j,2),triangle_edges(j,3))=A(triangle_edges(j,2),triangle_edges(j,3))+a(1,2);
   A(triangle_edges(j,2),triangle_edges(j,4))=A(triangle_edges(j,2),triangle_edges(j,4))+a(1,3);
   A(triangle_edges(j,3),triangle_edges(j,2))=A(triangle_edges(j,3),triangle_edges(j,2))+a(2,1);
   A(triangle_edges(j,3),triangle_edges(j,3))=A(triangle_edges(j,3),triangle_edges(j,3))+a(2,2);
   A(triangle_edges(j,3),triangle_edges(j,4))=A(triangle_edges(j,3),triangle_edges(j,4))+a(2,3);
   A(triangle_edges(j,4),triangle_edges(j,2))=A(triangle_edges(j,4),triangle_edges(j,2))+a(3,1);
   A(triangle_edges(j,4),triangle_edges(j,3))=A(triangle_edges(j,4),triangle_edges(j,3))+a(3,2);
   A(triangle_edges(j,4),triangle_edges(j,4))=A(triangle_edges(j,4),triangle_edges(j,4))+a(3,3);

   %Global C
   t=0;
   c=local_C(j,N,p1,p2,p3,t);
   C(triangle_edges(j,2),1)=C(triangle_edges(j,2),1)+c(1,1);
   C(triangle_edges(j,3),1)=C(triangle_edges(j,3),1)+c(2,1);
   C(triangle_edges(j,4),1)=C(triangle_edges(j,4),1)+c(3,1);

   %Global B
   b=local_B(j,N,p1,p2,p3);
   B(triangle_edges(j,2),triangle_edges(j,2))=B(triangle_edges(j,2),triangle_edges(j,2))+b(1,1);
   B(triangle_edges(j,2),triangle_edges(j,3))=B(triangle_edges(j,2),triangle_edges(j,3))+b(1,2);
   B(triangle_edges(j,2),triangle_edges(j,4))=B(triangle_edges(j,2),triangle_edges(j,4))+b(1,3);
   B(triangle_edges(j,3),triangle_edges(j,2))=B(triangle_edges(j,3),triangle_edges(j,2))+b(2,1);
   B(triangle_edges(j,3),triangle_edges(j,3))=B(triangle_edges(j,3),triangle_edges(j,3))+b(2,2);
   B(triangle_edges(j,3),triangle_edges(j,4))=B(triangle_edges(j,3),triangle_edges(j,4))+b(2,3);
   B(triangle_edges(j,4),triangle_edges(j,2))=B(triangle_edges(j,4),triangle_edges(j,2))+b(3,1);
   B(triangle_edges(j,4),triangle_edges(j,3))=B(triangle_edges(j,4),triangle_edges(j,3))+b(3,2);
   B(triangle_edges(j,4),triangle_edges(j,4))=B(triangle_edges(j,4),triangle_edges(j,4))+b(3,3);
end
```

```
%Movie Initialisierung
fig=figure;
set(fig,'DoubleBuffer','on');
set(gca,'xlim',[-80 80],'ylim',[-80 80],...
      'NextPlot','replace','Visible','off')
mov = avifile('test9.avi')


%Fixpunktiteration

tol=10;

%Anfangsbedingung=Loesung des stationaeren Problems
Hstart=bicgstab(A,C,1e-5,100);


%Plot Hstart Feld
intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hstart,edges);
h=plot_abs(intfield,N);
shading interp
colormap jet
colorbar
set(h,'EraseMode','xor');

F = getframe(gca);
mov = addframe(mov,F);

%Speicherreservierung
Hold=zeros(number_of_edges,1);
Hnew=zeros(number_of_edges,1);
Bold=zeros(number_of_edges,1);
Bnew=zeros(number_of_edges,1);

%Zeitschritt dt
dt=0.001;
%constant sigma
s=1;
sinverse=1/s;

%Konstanten
mu0=0.001;
B0=0.01;
epsilon=0.001;

%lambda fuer update
lambda=0.9;
%EndZeit
T=20*dt
%Anzahl Zeitschritte
n=T/dt;

for i=0:n
   t=i*dt;
   %Berechne zeitabhaengiges C
   C=sparse(number_of_edges,1);
   for j=1:number_of_triangles

      p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
      p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
      p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

      c=local_C(j,N,p1,p2,p3,t);
      C(triangle_edges(j,2),1)=C(triangle_edges(j,2),1)+c(1,1);
      C(triangle_edges(j,3),1)=C(triangle_edges(j,3),1)+c(2,1);
      C(triangle_edges(j,4),1)=C(triangle_edges(j,4),1)+c(3,1);
   end

   Hold=Hstart;
   %Zaehler fuer Iterationen
   iter=0;
```

```
    for k=1:size(Hstart,1)
        if Hstart(k,1)==0
            Bstart(k,1)=0;
        else
            Bstart(k,1)=mu0*Hstart(k,1)+B0*Hstart(k,1)/(abs(Hstart(k,1)) + epsilon^2);
        end
    end

    %Iteration
    while tol>0.0005
        iter=iter+1;
        for k=1:size(Hold,1)
            if Hold(k,1)==0
                Bold(k,1)=0;
               else
                Bold(k,1)=mu0*Hold(k,1)+B0*Hold(k,1)/(abs(Hold(k,1)) + epsilon^2);
            end
        end
        rs=dt*sinverse*C+B*Bstart-B*Bold;
        ls=dt*sinverse*A;
        Hnew=bicgstab(ls,rs,1e-5,100);
        %update
        Hnew=lambda*Hnew+(1-lambda)*Hold;
        tol=norm(Hnew-Hold);
        Hold=Hnew;
        if iter==300
            tol=0.000001;
        end
        i
    end
    data(i+1,1)=i+1;
    data(i+1,2)=iter;
    intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hnew,edges);

    set(h,'EraseMode','xor');
    F = getframe(gca);
    mov = addframe(mov,F);
    tol=10;
    Hstart=Hnew;
end
 mov = close(mov);
```

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FEM_Relaxation.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FEM mit Relaxationsverfahren
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Gitterlaenge
d=1;
%Ortsdiskretisierung
N=20;

%Gittergenerierung
[nodes,edges,triangle_nodes,triangle_edges]= grid1(d,N);

number_of_edges=2*N*(N+1)+N*N;
number_of_triangles=N*N*2;

%fuer Auswertung
innercounter=0;
subcounter=0;
data1=zeros(50,50);
data2=zeros(50,50);



%Speicherreservierung
A=sparse(number_of_edges,number_of_edges);
C=sparse(number_of_edges,1);
L=sparse(number_of_edges,number_of_edges);
L_old=sparse(number_of_edges,number_of_edges);
L_new=sparse(number_of_edges,number_of_edges);

%Assemblierung
for j=1:number_of_triangles
    p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
    p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
    p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

    %Global A
    a=local_A(j,N,p1,p2,p3);
    A(triangle_edges(j,2),triangle_edges(j,2))=A(triangle_edges(j,2),triangle_edges(j,2))+a(1,1);
    A(triangle_edges(j,2),triangle_edges(j,3))=A(triangle_edges(j,2),triangle_edges(j,3))+a(1,2);
    A(triangle_edges(j,2),triangle_edges(j,4))=A(triangle_edges(j,2),triangle_edges(j,4))+a(1,3);
    A(triangle_edges(j,3),triangle_edges(j,2))=A(triangle_edges(j,3),triangle_edges(j,2))+a(2,1);
    A(triangle_edges(j,3),triangle_edges(j,3))=A(triangle_edges(j,3),triangle_edges(j,3))+a(2,2);
    A(triangle_edges(j,3),triangle_edges(j,4))=A(triangle_edges(j,3),triangle_edges(j,4))+a(2,3);
    A(triangle_edges(j,4),triangle_edges(j,2))=A(triangle_edges(j,4),triangle_edges(j,2))+a(3,1);
    A(triangle_edges(j,4),triangle_edges(j,3))=A(triangle_edges(j,4),triangle_edges(j,3))+a(3,2);
    A(triangle_edges(j,4),triangle_edges(j,4))=A(triangle_edges(j,4),triangle_edges(j,4))+a(3,3);

    %Global C
    t=0;
    c=local_C(j,N,p1,p2,p3,t);
    C(triangle_edges(j,2),1)=C(triangle_edges(j,2),1)+c(1,1);
    C(triangle_edges(j,3),1)=C(triangle_edges(j,3),1)+c(2,1);
    C(triangle_edges(j,4),1)=C(triangle_edges(j,4),1)+c(3,1);
end

%Movie Initialisierung
fig=figure;
set(fig,'DoubleBuffer','on');
set(gca,'xlim',[-80 80],'ylim',[-80 80],...
        'NextPlot','replace','Visible','off')
mov = avifile('test5.avi')
```

```
%Anfangsbedingung=Loesung stat. Problem
Hstart=bicgstab(A,C,1e-5,100);

%Plot Hstart Feld
intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hstart,edges);
 h=plot_abs(intfield,N);
 shading interp
 colormap jet
 colorbar

h=plot_int(nodes,intfield)
set(h,'EraseMode','xor');
F = getframe(gca);
mov = addframe(mov,F);

%Zeitschritt dt
dt=0.0001;
%EndZeit
T=0.0005;
%Anzahl steps
n=T/dt;

%physical constants
s=1;
sinverse=1/s;
mu0=0.001;

%Relaxation Parameter
B0=0.01;
gamma=0.9999;
d=0.25;
K=1000;

epsilon=B0/(dt^(-d)-mu0)

%berechne interpolierte Feldet
intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hstart,edges);
intfield_triangles=interpolate_triangles(intfield,triangle_nodes,number_of_triangles);

for j=1:number_of_triangles
    p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
    p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
    p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

    %Berechen Jacobi Matrix
    DB_eps=Jacobian_B(intfield_triangles(j,2:3)', epsilon, mu0, B0);

    %Globale D Matrix, heisst hier L!!
    l=local_L(j,N,p1,p2,p3,DB_eps);
    L(triangle_edges(j,2),triangle_edges(j,2))=L(triangle_edges(j,2),triangle_edges(j,2))+l(1,1);
    L(triangle_edges(j,2),triangle_edges(j,3))=L(triangle_edges(j,2),triangle_edges(j,3))+l(1,2);
    L(triangle_edges(j,2),triangle_edges(j,4))=L(triangle_edges(j,2),triangle_edges(j,4))+l(1,3);
    L(triangle_edges(j,3),triangle_edges(j,2))=L(triangle_edges(j,3),triangle_edges(j,2))+l(2,1);
    L(triangle_edges(j,3),triangle_edges(j,3))=L(triangle_edges(j,3),triangle_edges(j,3))+l(2,2);
    L(triangle_edges(j,3),triangle_edges(j,4))=L(triangle_edges(j,3),triangle_edges(j,4))+l(2,3);
    L(triangle_edges(j,4),triangle_edges(j,2))=L(triangle_edges(j,4),triangle_edges(j,2))+l(3,1);
    L(triangle_edges(j,4),triangle_edges(j,3))=L(triangle_edges(j,4),triangle_edges(j,3))+l(3,2);
    L(triangle_edges(j,4),triangle_edges(j,4))=L(triangle_edges(j,4),triangle_edges(j,4))+L(3,3);
end



Hold=Hstart;
L_old=L;

intfield_old=intfield;
intfield_triangles_old=intfield_triangles;
tol=1000;
```

```
for i=0:n
    %Berechne zeitabhaengiges C
    tempppp=C;
    C=sparse(number_of_edges,1);
    for j=1:number_of_triangles
        t=i*dt;
        p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
        p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
        p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];

        c=local_C(j,N,p1,p2,p3,t);
        C(triangle_edges(j,2),1)=C(triangle_edges(j,2),1)+c(1,1);
        C(triangle_edges(j,3),1)=C(triangle_edges(j,3),1)+c(2,1);
        C(triangle_edges(j,4),1)=C(triangle_edges(j,4),1)+c(3,1);
    end

    innercounter=0;

    %innere Iteration
    while tol>1/2*dt
        innercounter=innercounter+1;
        if innercounter>=40
            break;
        end
        rs=dt*sinverse*C + L_old*Hold;
        ls=dt*sinverse*A + L_old;
        Htilde=bicgstab(ls,rs,1e-5,100);

        %berechne Interpolierte felder
        intfield_new=interpolate(N, nodes, triangle_nodes, triangle_edges,Htilde,edges);
        intfield_triangles_new=interpolate_triangles(intfield_new,triangle_nodes,number_of_triangles);

        %Berechene neues L
        L_new=sparse(number_of_edges,number_of_edges);
        for j=1:number_of_triangles
            p1=[nodes(triangle_nodes(j,2),2); nodes( triangle_nodes(j,2),3)];
            p2=[nodes(triangle_nodes(j,3),2); nodes( triangle_nodes(j,3),3)];
            p3=[nodes(triangle_nodes(j,4),2); nodes( triangle_nodes(j,4),3)];


            %berechne das Integral fuer Lnew mit Trapez Regel
            %Parameter: NN=Anzahl steps,h=Schrittweite
            NN=10;
            h=1/NN;
            temp=0;
            for n=1:NN-1
                temp=temp + h* Jacobian_B(intfield_triangles_old(j,2:3)'+n*h*gamma*(intfield_triangles_new(j,2:3)'-intfield_triangles_old(
j,2:3)'), epsilon, mu0, B0);
            end
            L_temp=1/2*h*Jacobian_B(intfield_triangles_old(j,2:3)',epsilon,mu0,B0) + 1/2*h*Jacobian_B(intfield_triangles_old(j,2:3)'+g
amma*(intfield_triangles_new(j,2:3)'-intfield_triangles_old(j,2:3)'), epsilon, mu0, B0);
            L_temp=gamma*(L_temp + temp);

            %Berechene L Matrix
            l=local_L(j,N,p1,p2,p3,L_temp);
            L_new(triangle_edges(j,2),triangle_edges(j,2))=L_new(triangle_edges(j,2),triangle_edges(j,2))+l(1,1);
            L_new(triangle_edges(j,2),triangle_edges(j,3))=L_new(triangle_edges(j,2),triangle_edges(j,3))+l(1,2);
            L_new(triangle_edges(j,2),triangle_edges(j,4))=L_new(triangle_edges(j,2),triangle_edges(j,4))+l(1,3);
            L_new(triangle_edges(j,3),triangle_edges(j,2))=L_new(triangle_edges(j,3),triangle_edges(j,2))+l(2,1);
            L_new(triangle_edges(j,3),triangle_edges(j,3))=L_new(triangle_edges(j,3),triangle_edges(j,3))+l(2,2);
            L_new(triangle_edges(j,3),triangle_edges(j,4))=L_new(triangle_edges(j,3),triangle_edges(j,4))+l(2,3);
            L_new(triangle_edges(j,4),triangle_edges(j,2))=L_new(triangle_edges(j,4),triangle_edges(j,2))+l(3,1);
            L_new(triangle_edges(j,4),triangle_edges(j,3))=L_new(triangle_edges(j,4),triangle_edges(j,3))+l(3,2);
            L_new(triangle_edges(j,4),triangle_edges(j,4))=L_new(triangle_edges(j,4),triangle_edges(j,4))+L(3,3);
        end
        %Korrektur um Explosion von L zu verhindern
        lambda=eigs(L_new,1);
        lambdamax=lambda(1,1);
        if K/lambdamax<1
            L_new=K/lambdamax*L_new;
```

```
        end

        tol=norm(L_new-L_old,inf)

        L_old=L_new;
        data1(i+1,innercounter)=tol;
        data2(i+1,innercounter)=lambdamax;
    end
    %Update
    for k=1:size(Hold,1);
        if Hold(k,1)==0
            rhs_update(k,1)=0;
        else
            rhs_update(k,1)=mu0*Hold(k,1) + B0*Hold(k,1)/abs(Hold(k,1));
        end
    end

    rhs_update=rhs_update + L_old*(Htilde-Hold);

    intfield_old=intfield_new;
    intfield_triangles_old=intfield_triangles_new;

    temp=Hold;
    Hold=B_H_Solver(rhs_update, mu0, B0,epsilon);
    aaa=norm(Hold-temp)
    intfield=interpolate(N, nodes, triangle_nodes, triangle_edges,Hold,edges);
    h=plot_abs(intfield,N);
    shading interp
    colormap jet
    colorbar
    h=plot_int(nodes,intfield)
    set(h,'EraseMode','xor');
    F = getframe(gca);
    mov = addframe(mov,F);

    tol=1000;
end

mov = close(mov);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% grid1.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  [nodes,edges,triangle_nodes,triangle_edges]= grid1(d,N)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Generiert Gitter fuer FEM.
%
% Input-Parameter:
%
% -d: Laenge einer Plattenseite
% -N: dx=d/N
%
% Output-Parameter:
%
% -nodes:  node_number    x_coord   y_coord
% -edges:  edge_number    node1     node2
% -triangle_nodes:  Triangle_number    node1    node2    node3
% -triangle_deges:  Triangle_number    edge1    edge2    edge3
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%meshlength
%d=1;
%Number of trinagles in each direction
%N=3;
%meshwidth
%h=d/N;
%Counter for nodes
node_number=0;

%A=node list
%
%      node_number   x_coord   y_coord
%
for j=1:N+1
    for i=1:N+1
        node_number=node_number+1;
        A(node_number,1)=node_number;
        A(node_number,2)=(i-1)*d/N;
        A(node_number,3)=(j-1)*d/N;
    end
end

%B=edge List
%
%      edge_number    node1    node2
%

%Horizobtal edges

edge_number=0;
for j=1:(N+1)
    for i=1:N
        edge_number=edge_number+1;
        B(edge_number,1)=edge_number;
        B(edge_number,2)=i+(j-1)*(N+1);
        B(edge_number,3)=i+(j-1)*(N+1)+1;
    end
end
%Vertikal edges
for j=1:N
    for i=1:N+1
        edge_number=edge_number+1;
        B(edge_number,1)=edge_number;
        B(edge_number,2)=i+(j-1)*(N+1);
```

```
        B(edge_number,3)=i+j*(N+1);
    end
end

%Diagonal edges
for j=1:N
    for i=1:N
        edge_number=edge_number+1;
        B(edge_number,1)=edge_number;
        B(edge_number,2)=i+(j-1)*(N+1);
        B(edge_number,3)=i+j*(N+1)+1;
    end
end


%Triangle_Node_List
%
%       Triangle_number    node1    node2    node3
%

triangle_number=0;
for j=1:N
    %1. Reihe
    for i=1:N
        triangle_number=triangle_number+1;
        C(triangle_number,1)=triangle_number;
        C(triangle_number,2)=i+(j-1)*(N+1);
        C(triangle_number,3)=i+(j-1)*(N+1)+1;
        C(triangle_number,4)=i+(j)*(N+1)+1;
    end
    %2.Reihe
    for i=1:N
        triangle_number=triangle_number+1;
        C(triangle_number,1)=triangle_number;
        C(triangle_number,2)=i+(j-1)*(N+1);
        C(triangle_number,3)=i+j*(N+1);
        C(triangle_number,4)=i+j*(N+1)+1;
    end
end


%Triangle_Edge List
%
%       Triangle_number edge1    edge2    edge3
%

triangle_number=0;

for j=1:N
    %1. Reihe
    for i=1:N
        triangle_number=triangle_number+1;
        D(triangle_number,1)=triangle_number;
        D(triangle_number,2)=i+(j-1)*N;
        D(triangle_number,3)=i+N*(N+1)+1+(j-1)*(N+1);
        D(triangle_number,4)=i+2*N*(N+1)+(j-1)*N;
    end
    %2.Reihe
    for i=1:N
        triangle_number=triangle_number+1;
        D(triangle_number,1)=triangle_number;
        D(triangle_number,2)=i+N*(N+1)+(j-1)*(N+1);
        D(triangle_number,3)=i+j*N;
        D(triangle_number,4)=i+2*N*(N+1)+(j-1)*N;
    end
end

nodes=A;
edges=B;
triangle_nodes=C;
```

```
triangle_edges=D;

%Changes for correct orientation
number_of_triangles=N*N*2;
for j=1:number_of_triangles
    if mod((ceil(j/N)),2)==0
        temp1=triangle_edges(j,2);
        temp2=triangle_edges(j,3);
        temp3=triangle_edges(j,4);
        triangle_edges(j,2)=temp3;
        triangle_edges(j,3)=temp1;
        triangle_edges(j,4)=temp2;
     end
end

number_of_triangles=N*N*2;
for j=1:number_of_triangles
    if mod((ceil(j/N)),2)==0
        temp1=triangle_nodes(j,2);
        temp2=triangle_nodes(j,3);
        temp3=triangle_nodes(j,4);
        triangle_nodes(j,2)=temp3;
        triangle_nodes(j,3)=temp1;
        triangle_nodes(j,4)=temp2;
     end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% interpolate.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function intfield= interpolate(N, nodes, triangle_nodes, triangle_edges,H,edges )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet das Feld im Mittelpunkt jeder Kante jedes Dreiecks
%
% Input-Parameter:
%
% -N,nodes, edges, triangle_nodes, triangle_edges: Gitterstruktur
% -H: zu interpolierendes feld
%
% Output-Parameter:
%
% -intfield: Interpoliertes Feld
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

number_of_edges=2*N*(N+1)+N*N;
number_of_triangles=N*N*2;
number_of_nodes=(N+1)*(N+1);

counter=0;
for j=1:number_of_triangles

   p1=[nodes(triangle_nodes(j,2),2); nodes(triangle_nodes(j,2),3)];
   p2=[nodes(triangle_nodes(j,3),2); nodes(triangle_nodes(j,3),3)];
   p3=[nodes(triangle_nodes(j,4),2); nodes(triangle_nodes(j,4),3)];

   Area=0.5*det([1 p1(1,1) p1(2,1); 1 p2(1,1) p2(2,1); 1 p3(1,1) p3(2,1)]);
   Area=abs(Area);
   a1=det([p2(1,1) p2(2,1); p3(1,1) p3(2,1)]);
   b1=-det([1 p2(2,1); 1 p3(2,1)]);
   c1=det([1 p2(1,1); 1 p3(1,1)]);

   a2=det([p3(1,1) p3(2,1); p1(1,1) p1(2,1)]);
   b2=-det([1 p3(2,1); 1 p1(2,1)]);
   c2=det([1 p3(1,1); 1 p1(1,1)]);

   a3=det([p1(1,1) p1(2,1); p2(1,1) p2(2,1)]);
   b3=-det([1 p1(2,1); 1 p2(2,1)]);
   c3=det([1 p1(1,1); 1 p2(1,1)]);

   %midpoint 1
   m=p1+(p2-p1)/2;
   x=m(1,1);
   y=m(2,1);
   psi1=1/2/Area*(a1 + b1*x + c1*y);
   psi2=1/2/Area*(a2 + b2*x + c2*y);
   psi3=1/2/Area*(a3 + b3*x + c3*y);

   N1(1,1)=1/4/(Area^2)*(psi1*b2-psi2*b1);
   N1(2,1)=1/4/(Area^2)*(psi1*c2-psi2*c1);
   N2(1,1)=1/4/(Area^2)*(psi2*b3-psi3*b2);
   N2(2,1)=1/4/(Area^2)*(psi2*c3-psi3*c2);
   N3(1,1)=1/4/(Area^2)*(psi3*b1-psi1*b3);
   N3(2,1)=1/4/(Area^2)*(psi3*c1-psi1*c3);
   H1=H(triangle_edges(j,2),1);
   H2=H(triangle_edges(j,3),1);
   H3=H(triangle_edges(j,4),1);

   counter=counter+1;
   field(counter,1)=triangle_edges(j,2);
   field(counter,2)=H1*N1(1,1)+H2*N2(1,1)+H3*N3(1,1);
   field(counter,3)=H1*N1(2,1)+H2*N2(2,1)+H3*N3(2,1);
```

```
    %midpoint2
    m=p2+(p3-p2)/2;
    x=m(1,1);
    y=m(2,1);
    psi1=1/2/Area*(a1 + b1*x + c1*y);
    psi2=1/2/Area*(a2 + b2*x + c2*y);
    psi3=1/2/Area*(a3 + b3*x + c3*y);

    N1(1,1)=1/4/(Area^2)*(psi1*b2-psi2*b1);
    N1(2,1)=1/4/(Area^2)*(psi1*c2-psi2*c1);
    N2(1,1)=1/4/(Area^2)*(psi2*b3-psi3*b2);
    N2(2,1)=1/4/(Area^2)*(psi2*c3-psi3*c2);
    N3(1,1)=1/4/(Area^2)*(psi3*b1-psi1*b3);
    N3(2,1)=1/4/(Area^2)*(psi3*c1-psi1*c3);
    H1=H(triangle_edges(j,2),1);
    H2=H(triangle_edges(j,3),1);
    H3=H(triangle_edges(j,4),1);

    counter=counter+1;
    field(counter,1)=triangle_edges(j,3);
    field(counter,2)=H1*N1(1,1)+H2*N2(1,1)+H3*N3(1,1);
    field(counter,3)==H1*N1(2,1)+H2*N2(2,1)+H3*N3(2,1);

    %midpoint3
    m=p3+(p1-p3)/2;
    x=m(1,1);
    y=m(2,1);
    psi1=1/2/Area*(a1 + b1*x + c1*y);
    psi2=1/2/Area*(a2 + b2*x + c2*y);
    psi3=1/2/Area*(a3 + b3*x + c3*y);

    N1(1,1)=1/4/(Area^2)*(psi1*b2-psi2*b1);
    N1(2,1)=1/4/(Area^2)*(psi1*c2-psi2*c1);
    N2(1,1)=1/4/(Area^2)*(psi2*b3-psi3*b2);
    N2(2,1)=1/4/(Area^2)*(psi2*c3-psi3*c2);
    N3(1,1)=1/4/(Area^2)*(psi3*b1-psi1*b3);
    N3(2,1)=1/4/(Area^2)*(psi3*c1-psi1*c3);
    H1=H(triangle_edges(j,2),1);
    H2=H(triangle_edges(j,3),1);
    H3=H(triangle_edges(j,4),1);

    counter=counter+1;
    field(counter,1)=triangle_edges(j,4);
    field(counter,2)=H1*N1(1,1)+H2*N2(1,1)+H3*N3(1,1);
    field(counter,3)==H1*N1(2,1)+H2*N2(2,1)+H3*N3(2,1);
end
number_of_midpoints=counter;
edgecounter=0;
x=0;
y=0;
for j=1:number_of_nodes
    for k=1:number_of_edges
        if edges(k,2)==j | edges(k,3)==j
            for m=1:number_of_midpoints
                if field(m,1)==k
                    x=x+field(m,2);
                    y=y+field(m,3);
                    edgecounter=edgecounter+1;
                end
            end
        end
    end
    x=x/edgecounter;
    y=y/edgecounter;
    intfield(j,1)=j;
    intfield(j,2)=x;
    intfield(j,3)=y;
    edgecounter=0;
    x=0;
```

```
    y=0;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% interpolate_triangles.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function int_field_triangles= interpolate_triangles( intfield,triangle_nodes,number_of_triangles)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet das Feld im Mittelpunkt jedes Dreiecks aus dem interpolierten
% Feld ueber den Kanten
%
% Input-Parameter:
%
% -triangle_nodes, number_of_triangles: Gitterstruktur
% -H: zu interpolierendes feld
%
% Output-Parameter:
%
% -int_field_triangles: Interpoliertes Feld
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:number_of_triangles
    int_field_triangles(j,1)=j;
    int_field_triangles(j,2)=1/3*(intfield(triangle_nodes(j,2),2)+intfield(triangle_nodes(j,3),2)+intfield(triangle_nodes(j,4),2));
    int_field_triangles(j,3)=1/3*(intfield(triangle_nodes(j,2),3)+intfield(triangle_nodes(j,3),3)+intfield(triangle_nodes(j,4),3));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% j0.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function j= j0(x,y,t)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Erregerstrom
%
% Input-Parameter:
%
% -x,y: Ortskoordinaten
% -t: Zeitkoordinate
%
% Output-Parameter:
%
% -j: =j0(x,y,t)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

j=1000*y*y*sin(2*pi*t/0.019);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Jacobian_B.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function DB_eps = Jacobian_B(H, epsilon, mu0, B0)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet die Jacobi Matrix fuer das Relaxationsverfahren
%
% Input-Parameter:
%
% -epsilon, mu0, B0: Regularisierungsparameter
% -H: aktuelles Feld
%
% Output-Parameter:
%
% -DB_eps: Jacobi Matrix
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DB_eps=mu0*eye(2) + B0*(1/sqrt((H(1,1)^2+H(2,1)^2)+epsilon^2)*eye(2) - H*H'/sqrt((H(1,1)^2+H(2,1)^2)+epsilon^2)^3);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% local_A.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function a= local_A( number_of_triangle,N,p1,p2,p3)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet die lokale Elementmatrix A
%
% Input-Parameter:
%
% -number_of_triangle: aktuelles Dreieck
% -N: Diskretisierung
% -p1, p2, p3: Dreieckseckpunkte
%
% Output-Parameter:
%
% -a: lokale Elementmatrix
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Cooefficients for the triangle
j=number_of_triangle;
Area=0.5*det([1 p1(1,1) p1(2,1); 1 p2(1,1) p2(2,1); 1 p3(1,1) p3(2,1)]);
Area=abs(Area);
a1=det([p2(1,1) p2(2,1); p3(1,1) p3(2,1)]);
b1=-det([1 p2(2,1); 1 p3(2,1)]);
c1=det([1 p2(1,1); 1 p3(1,1)]);

a2=det([p3(1,1) p3(2,1); p1(1,1) p1(2,1)]);
b2=-det([1 p3(2,1); 1 p1(2,1)]);
c2=det([1 p3(1,1); 1 p1(1,1)]);

a3=det([p1(1,1) p1(2,1); p2(1,1) p2(2,1)]);
b3=-det([1 p1(2,1); 1 p2(2,1)]);
c3=det([1 p1(1,1); 1 p2(1,1)]);

%Local Matrix a
factor=1/4/(Area^3);
a(1,1)=factor*(b1*c2-b2*c1)^2;
a(1,2)=factor*(b1*c2-b2*c1)*(b2*c3-c2*b3);
a(1,3)=factor*(b1*c2-b2*c1)*(b3*c1-c3*b1);
a(2,1)=a(1,2);
a(2,2)=factor*(b2*c3-c2*b3)^2;
a(2,3)=factor*(b2*c3-c2*b3)*(b3*c1-c3*b1);
a(3,1)=a(1,3);
a(3,2)=a(2,3);
a(3,3)=factor*(b3*c1-c3*b1)^2;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% local_B.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function b = local_B( number_of_triangle,N,p1,p2,p3 )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet die lokale Elementmatrix B
%
% Input-Parameter:
%
% -number_of_triangle: aktuelles Dreieck
% -N: Diskretisierung
% -p1, p2, p3: Dreieckseckpunkte
%
% Output-Parameter:
%
% -b: lokale Elementmatrix
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


j=number_of_triangle;
Area=0.5*det([1 p1(1,1) p1(2,1); 1 p2(1,1) p2(2,1); 1 p3(1,1) p3(2,1)]);
Area=abs(Area);
a1=det([p2(1,1) p2(2,1); p3(1,1) p3(2,1)]);
b1=-det([1 p2(2,1); 1 p3(2,1)]);
c1=det([1 p2(1,1); 1 p3(1,1)]);

a2=det([p3(1,1) p3(2,1); p1(1,1) p1(2,1)]);
b2=-det([1 p3(2,1); 1 p1(2,1)]);
c2=det([1 p3(1,1); 1 p1(1,1)]);

a3=det([p1(1,1) p1(2,1); p2(1,1) p2(2,1)]);
b3=-det([1 p1(2,1); 1 p2(2,1)]);
c3=det([1 p1(1,1); 1 p2(1,1)]);

%Local Matrix b

factor1=1/24/(Area)^2;
factor2=factor1/2;

b(1,1)=b2*b2 + c2*c2 - (b1*b2 + c1*c2) + b1*b1 + c1*c1;
b(1,1)=factor1*b(1,1);

b(1,2)=b2*b3 + c2*c3 - (b2*b2 + c2*c2) -2*(b1*b3 +c1*c3) + b1*b2 + c1*c2;
b(1,2)=factor2*b(1,2);

b(1,3)=b2*b1 + c2*c1 - 2*(b2*b3 +c2*c3) - (b1*b1 + c1*c1) + b1*b3 + c1*c3;
b(1,3)=factor2*b(1,3);

b(2,1)=b(1,2);

b(2,2)=b3*b3 + c3*c3 - (b2*b3 + c2*c3) + b2*b2 + c2*c2;
b(2,2)=factor1*b(2,2);

b(2,3)=b3*b1 + c3*c1 - (b3*b3 + c3*c3) - 2*(b2*b1 + c2*c1) + b2*b3 + c2*c3;
b(2,3)=factor2*b(2,3);

b(3,1)=b(1,3);

b(3,2)=b(2,3);

b(3,3)=b1*b1 + c1*c1 - (b1*b3 + c1*c3) + b3*b3 + c3*c3;
b(3,3)=factor1*b(3,3);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% local_C.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function c = local_C( number_of_triangle,N,p1,p2,p3,t)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet den lokalen Vektor C
%
% Input-Parameter:
%
% -number_of_triangle: aktuelles Dreieck
% -N: Diskretisierung
% -p1, p2, p3: Dreieckseckpunkte
% -t: aktuelle Zeit
%
% Output-Parameter:
%
% -c: lokaler Elementvektor
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Cooefficients for the triangle
j=number_of_triangle;
Area=0.5*det([1 p1(1,1) p1(2,1); 1 p2(1,1) p2(2,1); 1 p3(1,1) p3(2,1)]);
Area=abs(Area);
a1=det([p2(1,1) p2(2,1); p3(1,1) p3(2,1)]);
b1=-det([1 p2(2,1); 1 p3(2,1)]);
c1=det([1 p2(1,1); 1 p3(1,1)]);

a2=det([p3(1,1) p3(2,1); p1(1,1) p1(2,1)]);
b2=-det([1 p3(2,1); 1 p1(2,1)]);
c2=det([1 p3(1,1); 1 p1(1,1)]);

a3=det([p1(1,1) p1(2,1); p2(1,1) p2(2,1)]);
b3=-det([1 p1(2,1); 1 p2(2,1)]);
c3=det([1 p1(1,1); 1 p2(1,1)]);

%Local Vector c

factor=1/(Area^2)/2;
m1=(p1+p2)/2;
m2=(p2+p3)/2;
m3=(p3+p1)/2;

integral=1/3*Area*(j0(m1(1,1),m1(2,1),t) + j0(m2(1,1),m2(2,1),t) + j0(m3(1,1),m3(2,1),t));
c(1,1)=(b1*c2-b2*c1)*integral*factor;
c(2,1)=(b2*c3-b3*c2)*integral*factor;
c(3,1)=(b3*c1-b1*c3)*integral*factor;
```

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% local_L.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function l= local_L( number_of_triangle,N,p1,p2,p3,L)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Berechnet die lokale Elementmatrix D
%
% Input-Parameter:
%
% -number_of_triangle: aktuelles Dreieck
% -N: Diskretisierung
% -p1, p2, p3: Dreieckseckpunkte
% -L:Matrix L
%
% Output-Parameter:
%
% -l: lokale Elementmatrix d
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Cooefficients for the triangle
j=number_of_triangle;
Area=0.5*det([1 p1(1,1) p1(2,1); 1 p2(1,1) p2(2,1); 1 p3(1,1) p3(2,1)]);
Area=abs(Area);
a1=det([p2(1,1) p2(2,1); p3(1,1) p3(2,1)]);
b1=-det([1 p2(2,1); 1 p3(2,1)]);
c1=det([1 p2(1,1); 1 p3(1,1)]);

a2=det([p3(1,1) p3(2,1); p1(1,1) p1(2,1)]);
b2=-det([1 p3(2,1); 1 p1(2,1)]);
c2=det([1 p3(1,1); 1 p1(1,1)]);

a3=det([p1(1,1) p1(2,1); p2(1,1) p2(2,1)]);
b3=-det([1 p1(2,1); 1 p2(2,1)]);
c3=det([1 p1(1,1); 1 p2(1,1)]);

l11=L(1,1);
l12=L(1,2);
l21=L(2,1);
l22=L(2,2);


%Local Matrix d
factor=1/8/(Area)/24;

d(1,1)=l11*(2*b2*b2 - 2*b1*b2 + 2*b1*b1)+...
    l12*(2*b2*c2 - b1*c2 - b2*c1 + 2*b1*c1)+...
    l21*(2*b2*c2 - b2*c1 - b1*c2 + 2*b1*c1)+...
    l22*(2*c2*c2 - 2*c1*c2 + 2*c1*c1);
d(1,1)=d(1,1)*factor;
d(1,2)=l11*(b2*b3 - b2*b2 - 2*b1*b3 + b1*b2)+...
    l12*(c2*b3 - c2*b2 - 2*b3*c1 + b2*c1)+...
    l21*(b2*c3 - b2*c2 - 2*b1*c3 + b1*c2)+...
    l22*(c2*c3 - c2*c2 - 2*c1*c3 + c1*c2);
d(1,2)=d(1,2)*factor;
d(1,3)=l11*(b2*b1 - 2*b2*b3 - b1*b1 + b1*b3)+...
    l12*(b1*c2 - 2*c2*b3 - c1*b1 + b3*c1)+...
    l21*(b2*c1 - 2*b2*c3 - b1*c1 + b1*c3)+...
    l22*(c2*c1 - 2*c2*c3 - c1*c1 + c1*c3);
d(1,3)=d(1,3)*factor;
d(2,1)=d(1,2);
d(2,2)=l11*(2*b3*b3 - 2*b2*b3 + 2*b2*b2)+...
    l12*(2*b3*c3 - b2*c3 - b3*c2 + 2*b2*c2)+...
    l21*(2*b3*c3 - b3*c2 - b2*c3 + 2*b2*c2)+...
    l22*(2*c3*c3 - 2*c2*c3 + 2*c2*c2);
```

```
d(2,2)=d(2,2)*factor;
d(2,3)=l11*(b3*b1 - b3*b3 - 2*b1*b2 + b2*b3)+...
      l12*(b1*c3 - b3*c3 - 2*b1*c2 + c2*b3)+...
      l21*(b3*c1 - b3*c3 - 2*b2*c1 + b2*c3)+...
      l22*(c1*c3 - c3*c3 - 2*c2*c1 + c2*c3);
d(2,3)=d(2,3)*factor;
d(3,1)=d(1,3);
d(3,2)=d(2,3);
d(3,3)=l11*(2*b1*b1 - 2*b1*b3 + 2*b3*b3)+...
      l12*(2*b1*c1 - b3*c1 - b1*c3 + 2*b3*c3)+...
      l21*(2*b1*c1 - b1*c3 - b3*c1 + 2*b3*c3)+...
      l22*(2*c1*c1 - 2*c1*c3 + 2*c3*c3);
d(3,3)=d(3,3)*factor;
l=d;
```

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% plot_abs.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plot = plot_abs(intfield,N)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Graphische Darstellung Betrages des interpolierten Feldes
%
% Input-Parameter:
%
% -intfield: interpoliertes Feld
% -N: Diskretisierung
%
% Output-Parameter:
%
% -plot: Referenz auf einen Plot
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

absolute_value=sqrt((intfield(:,2)).^2+(intfield(:,3)).^2);
absolute_value=absolute_value';

x=[0:1/N:1];
y=[0:1/N:1];
[X,Y]=meshgrid(x,y);

for j=1:N+1
   Z(j,1:N+1)=absolute_value((j-1)*(N+1)+1:(j-1)*(N+1)+N+1);
end

plot=pcolor(X,Y,Z);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% plot_int.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plot = plot_int( nodes,intfield )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Graphische Darstellung des interpolierten Feldes mit Pfeilen
%
% Input-Parameter:
%
% -intfield: interpoliertes Feld
% -nodes: Gitter
%
% Output-Parameter:
%
% -plot: Referenz auf einen Plot
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    plot=quiver(nodes(:,2),nodes(:,3),intfield(:,2),intfield(:,3))
end
```