Technische Universität Wien

Fakultät für Elektrotechnik und Informationstechnik

Bachelorarbeit

Numerische Lösung der Poisson-Gleichung mit dem Mehrgitterverfahren

Autor:
Alexander Schwed (1028752)

Betreuer:
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Erasmus Langer
MSc Dipl.-Ing. Dr.techn. Karl Rupp

2015/2016

Besonderen Dank möchte ich meiner Frau Carina, die mich stets in allen Lebensbereichen, auch im Rahmen dieser Arbeit geduldig und aufopfernd unterstützt hat, aussprechen. Trotz ihres anstrengenden Arbeitsalltags hatte sie noch ein offenes Ohr für mich, wenn es bei der Softwareentwicklung zu dieser Arbeit das eine oder andere Problem gab.
Außerdem bedanke ich mich bei meinen Eltern, die mich zu diesem Studium ermutigt, und nicht zuletzt auch finanziell unterstützt haben.
Zu guter Letzt gilt Karl Rupp mein besonderer Dank, ein äußerst gewissenhafter und zuvorkommender Betreuer, den ich auf jeden Fall weiterempfehlen möchte.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit soll dem Leser ein grundlegendes Verständnis zur Lösung partieller linearer Differentialgleichungen mit Hilfe des Mehrgitterverfahrens geben. Zunächst werden einige Verfahren und Begriffe, die für die Herangehensweise wichtig sind, ausgearbeitet, um schrittweise der Idee des Mehrgitterverfahrens näher zu kommen.

In der Wissenschaft trifft man häufig auf partielle Differentialgleichungen, deren analytischen Lösungen in vielen Fällen schwer oder unmöglich zu berechnen sind. Ein wichtiges Beispiel dafür liefert die Poisson-Gleichung, die unter anderem Anwendung in der Elektrodynamik findet. Die Methode der finiten Differenzen ist ein wichtiger Vertreter zur numerischen Behandlung von Randwertproblemen. Die Problemstellung wird im Raum diskretisiert, es werden partielle lineare Differentialgleichungen auf algebraische Gleichungen abgebildet. Zunächst soll das Ziel sein, diese Gleichungen mit dem Eliminationsverfahren von Gauß, der Gauß-Seidel Iteration, der Gauß-Seidel Relaxation, sowie der Jacobi Iteration zu lösen. Die Komplexität dieser Verfahren, in Abhängigkeit der Anzahl an Unbekannten n, ist für die zweidimensionale Poisson-Gleichung durchaus unterschiedlich. Das Eliminationsverfahren von Gauß benötigt $\mathcal{O}(n^3)$, die Gauß-Seidel Iteration, sowie das Jacobi Verfahren $\mathcal{O}(n^2 \log \epsilon)$, und die Gauß-Seidel Relaxation $\mathcal{O}(n^{\frac{3}{2}} \log \epsilon)$ Rechenschritte, je nach gewünschter Genauigkeit ϵ (Abbruchkriterium bei Iterationsverfahren) [7, S. 14]. Ziel soll es sein, diese Anzahl an Rechenschritten (Iterationen) mithilfe des iterativen Mehrgitterverfahrens auf $\mathcal{O}(n \log \epsilon)$ zu senken.

Die Grundidee besteht darin, die Fehleranteile der Lösungsfunktion (welche zu Beginn noch willkürliche Werte annimmt) zu glätten, sie sollen sich von Punkt zu Punkt wenig unterscheiden. Nach diesem Schritt können Berechnungen auf gröberen Gittern, mit weniger Unbekannten ausgeführt werden. Anschließend ist man in der Lage zwischen diesen errechneten Lösungen zu interpolieren, um die Werte der Gitterpunkte auf einem feineren Gitterlevel zu erhalten. Mithilfe dieses Vorgehens wird eine Lösung mit einer geringeren Anzahl an Iterationsschritten und kürzerer Berechnungszeit schneller und effizienter ($\mathcal{O}(n \log \epsilon)$) berechnet.
Anhand dieser Überlegungen sind Algorithmen zu formulieren, die in einer Software implementiert werden. Abschließend können praktische Vergleiche zwischen den vorgestellten Verfahren hinsichtlich ihrer Ausführungszeit und Anzahl an Iterationen angestellt werden.

2 Grundlagen

2.1 Partielle Differentialgleichungen

Zu Beginn wird die grundlegende Nomenklatur für partielle Differentialgleichungen festgelegt. Anfangs gehen wir von einer allgemeinen Differentialgleichung der Form $Lu = f$ aus. Es wird im Weiteren L als Differentialoperator zweiter Ordnung bezeichnet, der im Rahmen dieser Arbeit als linear anzunehmen ist. Die Funktion u ist die gesuchte Lösungsfunktion, und f die Inhomogenität (rechte Seite). Die Inhomogenität f sowie sämtliche Koeffizienten sollen für unsere Problemstellung nicht von Ableitungen der Lösungsfunktion, sondern ausschließlich von den Ortskoordinaten abhängig sein. Mit diesen Einschränkungen kann die Differentialgleichung zum Beispiel im $\mathbb{R}^2$ folgendermaßen angeschrieben werden[1]:

$$\begin{aligned} Lu &= a_{11}\frac{\partial^2 u}{\partial x^2} + a_{12}\frac{\partial^2 u}{\partial x \partial y} + a_{22}\frac{\partial^2 u}{\partial y^2} + a_1\frac{\partial u}{\partial x} + a_2\frac{\partial u}{\partial y} + a_0 u \\ &= a_{11}u_{xx} + a_{12}u_{xy} + a_{22}u_{yy} + a_1 u_x + a_2 u_y + a_0 u \end{aligned} \tag{2.1.1}$$

Unter Berücksichtigung weiterer Bedingungen klassifizieren wir den Differentialoperator L. Wir nennen L [7]:

- parabolisch, für $4a_{11}a_{22} = a_{12}^2$,
- hyperbolisch, für $4a_{11}a_{22} < a_{12}^2$,
- elliptisch, für $4a_{11}a_{22} > a_{12}^2$.

Im Rahmen dieser Arbeit werden wir die zweidimensionale Poisson-Gleichung $-\Delta u = -u_{xx} - u_{yy} = f$ mit Dirichlet Randbedingungen numerisch lösen. Der Differentialoperator L entspricht in diesem Fall $L := -\Delta$, wobei Δ den Laplace-Operator bezeichnet. Ist die Lösungsfunktion, sowie die rechte Seite (Inhomogenität) im $\mathbb{R}^2$ definiert ($u(x, y)$, $f(x, y)$), so ergibt sich die resultierende Poisson-Gleichung zu:

$$-\Delta u(x, y) = f(x, y) \text{ mit } (x, y) \in \mathbb{R} \tag{2.1.2}$$

In der Elektrodynamik [5, S. 60f] findet die Gleichung 2.1.2 Verwendung, um das elektrostatische Potenzial φ zu berechnen:

$$\nabla^2 \varphi \equiv \Delta\varphi = -\frac{\rho}{\epsilon} \tag{2.1.3}$$

Der Parameter ρ ist in Gleichung 2.1.3 die Raumladung und ϵ die Permitivität. Exemplarisch soll diese Gleichung für den Spezialfall $\frac{\rho}{\epsilon} = 1$ im weiteren Vorgehen mithilfe verschiedener Verfahren gelöst werden.

[1] Die partiellen Ableitungen $\frac{\partial u}{\partial x}$ notieren wir zu u_x.

2.2 Grundlagen der finiten Differenzen

Um die analytische Gleichung im Raum zu diskretisieren, verwendet man eine bestimmte Anzahl an Stützstellen, an denen Werte zu berechnen sind. Die Gesamtheit aller Punkte nennen wir Gitter. Der eindimensionale Fall ist für das Mehrgitterverfahren nicht von praktischem Interesse, jedoch hilft dieser beim Beschreiben des Konzepts der finiten Differenzen. Bevor dazu ins Detail gegangen wird, müssen wir uns auf ein Koordinatensystem festlegen. Im Folgenden werden wir das orthogonale kartesische Koordinatensystem verwenden. Es sei an dieser Stelle noch erwähnt, dass unsere Problemstellung auch in Kreiszylinderkoordinaten, Kugelkoordinaten, etc. angeschrieben werden kann[2]. Bei dem Verfahren der finiten Differenzen wird laut [4, S.20] zwischen

- vorwärtsgenommenem Differenzenquotient $\partial^+ u(x)$,
- rückwärtsgenommenem Differenzenquotient $\partial^- u(x)$,
- zentralem Differenzenquotient $\partial^0 u(x)$

unterschieden. In Abb. 2.2.1 ist die Funktion $u(x_i)$ mithilfe der drei verschiedenen Diskretisierungsmethoden (abgekürzt: Vorwärtsdifferenz, Rückwärtsdifferenz und Zentraldifferenz) auf die Funktion $u(x)$ abgebildet worden. Die Unterschiede zwischen den drei Verfahren werden der Reihe nach untersucht.

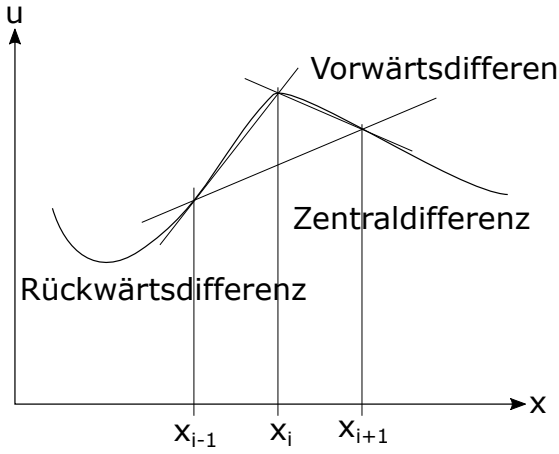

Abbildung 2.2.1: Finite Differenzen

Aus Abb. 2.2.1 kann man direkt die Ableitungen der angegebenen Funktion $u_h(x_i)$ geometrisch herleiten. Der Abstand zwischen zwei Punkten $\delta x = h$ gibt uns bereits Auskunft über die Genauigkeit der Diskretisierung. Im äquidistanten Fall (alle Punkte sind gleich weit voneinander entfernt) können die Differenzenquotienten folgendermaßen angegeben werden:

[2] An späterer Stelle wird gezeigt, dass wir alle Gitterpunkte orthogonal anordnen, das kartesische Koordinatensystem ist dafür die beste Wahl.

- Vorwärtsdifferenz

$$(\frac{\partial u}{\partial x})_i \approx \frac{u_{i+1} - u_i}{\delta x} \text{ für } \delta x = h = |x_{i+1} - x_i| \tag{2.2.1}$$

- Rückwärtsdifferenz

$$(\frac{\partial u}{\partial x})_i \approx \frac{u_i - u_{i-1}}{\delta x} \text{ für } \delta x = h = |x_i - x_{i-1}| \tag{2.2.2}$$

- Zentraldifferenz

$$(\frac{\partial u}{\partial x})_i \approx \frac{u_{i-1} - u_{i+1}}{2\delta x} \text{ für } \delta x = h = |x_{i+1} - x_{i-1}| \tag{2.2.3}$$

$$(\frac{\partial^2 u}{\partial x^2})_i \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{(\delta x)^2} \text{ für } \delta x = h = |x_{i+1} - x_{i-1}| \tag{2.2.4}$$

Zusätzlich zu dieser geometrischen - anschaulichen - Bildung der Differenzenquotienten, ist es hinsichtlich der Fehlerabschätzung sinnvoll, die Taylorreihen anzugeben [1]:

$$\text{Vorwärtsdifferenz: } (\frac{\partial u}{\partial x})_i = \frac{u_{i+1} - u_i}{\delta x} - \frac{\delta x}{2} \cdot (\frac{(\partial)^2 u}{\partial x^2})_i + \cdots$$

$$\text{Rückwärtsdifferenz: } (\frac{\partial u}{\partial x})_i = \frac{u_i - u_{i-1}}{\delta x} + \frac{\delta x}{2} \cdot (\frac{(\partial)^2 u}{\partial x^2})_i + \cdots \tag{2.2.5}$$

$$\text{Zentraldifferenz: } (\frac{\partial u}{\partial x})_i = \frac{u_{i+1} - u_{i-1}}{2\delta x} - \frac{(\delta x)^2}{6} \cdot (\frac{(\partial)^3 u}{\partial x^3})_i + \cdots$$

An dieser Stelle gestaltet sich die Fehlerabschätzung einfach, die Genauigkeit der Zentraldifferenz ist am höchsten. Sowohl bei der Vorwärts,- als auch Rückwärtsdifferenz sind Restterme von $\mathcal{O}(\delta x)$ vorhanden. Bei der Zentraldifferenz nehmen diese quadratisch $\mathcal{O}(\delta x^2)$ ab, weshalb wir mit der Zentraldifferenz weiter arbeiten, um höhere beziehungsweise gemischte partielle Ableitungen auf ähnliche Weise zu konstruieren. Um die Poisson-Gleichung für das Modellproblem im $\mathbb{R}^2$ zu diskretisieren, bleiben wir mit den Ortskoordinaten innerhalb des Einheitsquadrates $\Omega = (0,1)^2 \subset \mathbb{R}^2$. Der elementare (kleinste) Abstand zwischen zwei Punkten in einer Raumrichtung ist $h = \frac{1}{N}$ für $N \in \mathbb{N}$. Des Weiteren schränken wir uns auf ein äquidistantes Gitter ein, es sind die Bedingungen $\delta x = h$, sowie $\delta y = h$ zu erfüllen. Der Parameter N gibt die Anzahl der Stützstellen in einer Koordinatenrichtung an, ohne jedoch den Nullpunkt miteinzubeziehen, tatsächlich gibt es daher $N+1$ Stützstellen pro Raumdimension. Die Gleichung 2.1.2 werden wir mit diesen Festlegungen auf die diskrete 2D-Poisson-Gleichung abbilden, wobei der Index h für die Diskretisierung der Operatoren und Funktionen steht:

$$-\Delta_h u_h^{\Omega_h}(x,y) = f_h^{\Omega_h}(x,y) \text{ für } (x,y) \in \Omega_h \tag{2.2.6}$$

Unser Differentialoperator L kann auf diese Weise zu $L_h = -\Delta_h$ - mit einem abschätzbaren Quantisierungsfehler - approximiert werden. Der dadurch entstehende absolute Fehler ist $Lu - L_h u \approx \mathcal{O}(h^2)$, und verschwindet für $h \to 0$. Nach diesen Überlegungen wird mithilfe der Methode der finiten Differenzen unser Laplace-Operator für zwei Dimensionen diskret angeschrieben. Um eine bessere Übersicht zu erreichen, verwenden wir die Notation $u_h(x_i, y_j) = u_h(i \cdot h; j \cdot h) := u_{i,j}$.

$$\begin{aligned} \Delta_h u_{i,j} &= \left(\frac{\partial^2 u_{i,j}}{\partial x^2}\right) + \left(\frac{\partial^2 u_{i,j}}{\partial y^2}\right) \\ &= \frac{u_{i+h,j} - 2u_{i,j} + u_{i-h,j}}{h^2} + \frac{u_{i,j+h} - 2u_{i,j} + u_{i,j-h}}{h^2} \\ &= \frac{u_{i+h,j} + u_{i-h,j} - 4u_{i,j} + u_{i,j+h} + u_{i,j-h}}{h^2} \end{aligned} \tag{2.2.7}$$

Wird der Laplace-Operator in die 2D-Poisson-Gleichung eingesetzt, so erhält man:

$$\begin{aligned} -\Delta_h u_{i,j} &= f_{i,j} \\ \frac{4u_{i,j} - u_{i+h,j} - u_{i-h,j} - u_{i,j+h} - u_{i,j-h}}{h^2} &= f_{i,j} \end{aligned} \tag{2.2.8}$$

Bei der Anwendung der Formel 2.2.8-2 ist darauf zu achten, dass wir die Berechnungen stets auf das quadratische Gebiet beschränken $(1,1) \leq (i,j) \leq (N,N)$. Das Randgebiet muss nicht berechnet werden, die Werte dafür sind durch die Dirichlet Randbedingungen festgelegt. Für sämtliche folgende Berechnungen, sowie Grafiken ist der Rand des Gebietes auf $\partial\Omega_h = 1$ gesetzt. Zu jedem Gitterpunkt $u_{i,j}$ können wir die Gleichung 2.2.8-2 aufstellen. Der mittlere Gitterpunkt hat jeweils zwei Nachbarn in jede Raumrichtung. Um den fünften Punkt zu berechnen (wie in Abb. 2.2.2), brauchen wir mit dieser Methode für unsere Problemstellung insgesamt die Werte vierer umliegender Punkte, darum nennen wir das Verfahren 5-Punkte Stern Differenzenverfahren [4].

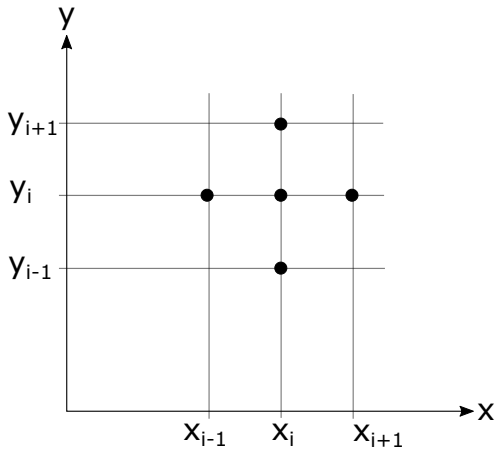

Abbildung 2.2.2: 5-Punkte-Stern

Zur Veranschaulichung lassen sich die Lösungspunkte eines Gitters mit $N = 3$ Stützstellen in jede Raumdimension laut Abb. 2.2.3 anordnen[3]. In weiterer Folge werden wir ein solches Gitter als G_h bezeichnen, in diesem Fall (Abb. 2.2.3) ist ein $G_{\frac{1}{3}}$ Gitter mit 16 Punkten zu sehen.

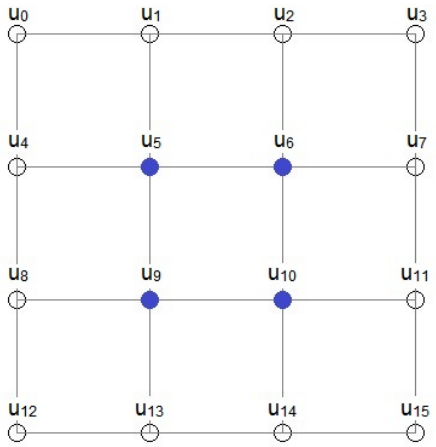

Abbildung 2.2.3: Anordnung der Gitterpunkte

Als erstes Beispiel können wir anhand der Abb. 2.2.3 den 5-Punkte Stern (Gleichung 2.2.8) für den Gitterpunkt u_5 auflösen:

$$-3^2 \cdot u_1 - 3^2 \cdot u_4 + 4 \cdot 3^2 \cdot u_5 - 3^2 \cdot u_6 - 3^2 \cdot u_9 = f_5 \tag{2.2.9}$$

Aufgrund der Randbedingungen müssen die Werte an den Punkten u_0-u_3, u_4, u_7, u_8, u_{11}, sowie u_{12}-u_{15} nicht durch den 5-Punkte Stern ausgedrückt werden. Diese treten

[3]Die Gitterindizes sind nicht in Matrixschreibweise $u_{i,j}$ angegeben, sondern in vektorieller Form u_i.

gemeinsam mit der Inhomogenität auf, daraus ergibt sich die triviale Gleichung:

$$1 \cdot u_0 = f_0 \tag{2.2.10}$$

Die Gleichungen des Typs 2.2.9 und 2.2.10 können anschließend für sämtliche Punkte in ein lineares Gleichungssystem in Matrixschreibweise zusammengefasst werden.

$$\begin{pmatrix}
1 & & & & & & & & & & & & & & & \\
& 1 & & & & & & & & & & & & & & \\
& & 1 & & & & & & & & & & & & & \\
& & & 1 & & & & & & & & & & & & \\
& & & & 1 & & & & & & & & & & & \\
-9 & & & -9 & 36 & -9 & & -9 & & & & & & & & \\
& -9 & & & -9 & 36 & -9 & & -9 & & & & & & & \\
& & & & & & 1 & & & & & & & & & \\
& & & & & & & 1 & & & & & & & & \\
& & & & -9 & & & -9 & 36 & -9 & & -9 & & & & \\
& & & & & -9 & & & -9 & 36 & -9 & & -9 & & & \\
& & & & & & & & & & 1 & & & & & \\
& & & & & & & & & & & 1 & & & & \\
& & & & & & & & & & & & 1 & & & \\
& & & & & & & & & & & & & 1 & & \\
& & & & & & & & & & & & & & 1 &
\end{pmatrix} \cdot \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \\ u_{15} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \\ f_{11} \\ f_{12} \\ f_{13} \\ f_{14} \\ f_{15} \end{pmatrix} \tag{2.2.11}$$

Von diesen 16 Punkten (u_0-u_{15}) sind die Werte an vier Stützstellen mit dem 5-Punkte Stern zu berechnen, die restlichen Punkte liegen am Rand. Im Folgenden werden diese vier Punkte als innere Punkte, und die gesamte $(N+1)^2 \times (N+1)^2$ Matrix als Systemmatrix A bezeichnet. Schreibt man das Gleichungssystem für ein größeres als das $G_{\frac{1}{3}}$ (Glg.: 2.2.11) Gitter an, so lässt sich ein Schema für die Aufstellung der Pentadiagonalmatrix ablesen. Jedes von 0 verschiedene Element $a_{i,j}$ der Matrix A lässt sich gemäß des 5-Punkte Sterns laut Algorithmus 2.2.1 beschreiben[4].

[4]Die Nummerierung der Zeilen (i)/Spalten(j) beginnt bei 0.

Daten : Anzahl der Stützstellen N
Ausgabe : Systemmatrix A
solange $i \leq N$ **tue**
  $a_{i,i} = 1$; `/* Die Randpunkte müssen nicht berechnet werden. */`
  inkrementiere i
solange $(i > N) \wedge [i < (N^2 + N - 2)]$ **tue**
  Wiederhole folgendes $N - 1$ **mal:**
    $a_{i,i} = 1$; `/* linker Randpunkt */`
    inkrementiere i
    Wiederhole folgendes $N - 1$ **mal:**
      $a_{i,x-N-1} = -1 \cdot N^2$
      $a_{i,x-1} = -1 \cdot N^2$
      $a_{i,i} = 4 \cdot N^2$
      $a_{i,x+1} = -1 \cdot N^2$
      $a_{i,x+N+1} = -1 \cdot N^2$
      inkrementiere i
    $a_{i,i} = 1$; `/* rechter Randpunkt */`
    inkrementiere i
  inkrementiere i
solange $i < (N + 1)^2$ **tue**
  $a_{i,i} = 1$; `/* untere Randpunkte */`
  inkrementiere i
Ende

Algorithmus 2.2.1 : Initialisierung der Matrix A

Der Algorithmus 2.2.1 zur Befüllung der Matrix A ist für das weitere Vorgehen von geringer Bedeutung. Würde man die komplette Matrix (inklusive aller 0-Elemente) als mehrdimensionales, dynamisch alloziertes Array speichern, so würden $\left[(N+1)^2\right]^2 \cdot sizeof(double)$[5] Bytes benötigt.
Man braucht für 256 Stützstellen einen RAM-Speicher von zirka 34, 9GB, jedoch sind von den darin enthaltenen $4,362 \cdot 10^9$ Elementen lediglich $(N+1)^2 + (N-1) \cdot 4 = 67,069 \cdot 10^3$ von Bedeutung. Die komplette Matrix A abzuspeichern ist demnach nicht zweckmäßig. An späterer Stelle werden wir zeigen, dass diese Matrix unter Verwendung verschiedener Iterationsverfahren nicht abgespeichert werden muss. Es reicht aus, die Anzahl an Stützstellen N zu kennen. Ist das Gebiet ein veränderlicher Parameter (zum Beispiel Dreiecksgebiet, L-Gebiet), so ist das Aufstellen der Pentadiagonalmatrix unerlässlich.

[5]Hier findet die intrinsische C-Funktion *sizeof*(Typ) Verwendung, die als Rückgabewert den erforderlichen Speicher für Typ in Bytes zurückgibt. In 32-bit-Rechnerarchitekturen erhält man den Rückgabewert von *sizeof*(*double*) für den doppelt genauen Gleitpunktdatentyp *double* zu 8 Bytes [3].

3 Lösen von Gleichungssystemen

Im vorherigen Abschnitt haben wir die 2D-Poisson-Gleichung $-\Delta u(x,y) = f(x,y)$ mit $(x,y) \in \mathbb{R}$ mithilfe des 5-Punkte Sterns in ein algebraisches Gleichungssystem (in Matrixschreibweise) der Form $Au = f$ umgewandelt. In diesem Kapitel sollen ausgewählte direkte und indirekte Lösungsverfahren vorgestellt werden.

3.1 Direkte Lösungsverfahren

Wir gehen vom Gleichungssystem mit den in Abschnitt 2.2 diskutierten Diskretisierungsfehlern behaftet aus, welches direkt (ohne Abbruchkriterium) gelöst werden soll. Rundungsfehler werden in [6, S.8ff] behandelt. Beim direkten Verfahren - hier das Eliminationsverfahren von Gauß - werden elementare Methoden, wie Zeilenoperationen- und Vertauschungen ausgeführt. Im Folgenden wird der Lösungsweg anhand eines allgemeinen Gleichungssystems beschrieben:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{3.1.1}$$

Wir werden mit elementaren Umformungen die Vorwärtselimination durchführen, um eine obere Dreiecksmatrix zu kreieren. Mit der Rückwärtselemination erreichen wir, dass ausschließlich die Hauptdiagonale der Matrix A mit Einträgen $a_{i,j} \neq 0$ besetzt ist.[6]

$$\begin{bmatrix} \tilde{a_{11}} & & \\ & \tilde{a_{22}} & \\ & & \tilde{a_{33}} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \tilde{f_1} \\ \tilde{f_2} \\ \tilde{f_3} \end{bmatrix} \tag{3.1.2}$$

Den Lösungsvektor u erhalten wir, indem der Quotient aus den Einträgen der Inhomogenität f und den Werten der Matrix A gebildet wird.

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \frac{f_1}{\tilde{a_{11}}} \\ \frac{f_2}{\tilde{a_{22}}} \\ \frac{f_3}{\tilde{a_{33}}} \end{bmatrix} \tag{3.1.3}$$

Für dieses direkte Lösungsverfahren benötigen wir bei einer $(N+1) \times (N+1)$ Matrix mit $(N+1)^2$ Einträgen zirka $\mathcal{O}(N^3)$ Rechenschritte. Es sei an dieser Stelle erwähnt, dass es alternative Herangehensweisen, wie zum Beispiel LR-Zerlegung, oder Pivotisierung, sowie das Cholesky-Verfahren gibt [6, S.28ff].

In Abschnitt 2.2 haben wir bereits festgestellt, dass es nicht notwendig ist, die komplette Matrix A mit allen Redundanzen abzuspeichern. Für das Eliminationsverfahren von

[6] Um bei der Programmierung komplizierte Schleifenoperationen (drei Schleifen ineinander) zu vermeiden, kann - wenn schon auf die obere Dreiecksmatrix umgerechnet wurde - von unten nach oben in die Matrix eingesetzt werden.

Gauß benötigen wir trotzdem Einträge, die vorerst mit 0 initialisiert sind. Wir werden die Systemmatrix A der Gleichung 2.2.11 zu diesem Zweck zu $\widetilde{A}$ abändern.

$$\begin{pmatrix}
 & & 1 & & \\
 & & 1 & & \\
 & & 1 & & \\
 & & 1 & & \\
 & & 1 & & \\
-9 & -9 & 36 & -9 & -9 \\
-9 & -9 & 36 & -9 & -9 \\
 & & 1 & & \\
 & & 1 & & \\
-9 & -9 & 36 & -9 & -9 \\
-9 & -9 & 36 & -9 & -9 \\
 & & 1 & & \\
 & & 1 & & \\
 & & 1 & & \\
 & & 1 & & \\
 & & 1 & &
\end{pmatrix} \tag{3.1.4}$$

Bei der reduzierten Matrix $\widetilde{A}$ ist keinerlei Information verloren gegangen. Alle Einträge, die wir für das Eliminationsverfahren brauchen, sind vorhanden, es werden lediglich Redundanzen entfernt. Die vorgenommene Speicherersparnis ergibt sich zu:

$$sizeof(double) \cdot \left[(N+1)^2\right]^2 - (N+1)^2 \cdot (2 \cdot N + 3)$$

Für 256 Stützstellen lässt sich eine Speicherersparnis von zirka $34,63$GB feststellen, die neue Matrix benötigt einen Speicher von $272,12$MB.[7] Es gibt noch die Möglichkeit, durch angepasste Algorithmen einen weiteren großen Teil der Matrix zu entfernen, sodass die $N-1$ inneren Punkte für die Aufstellung der Matrix berücksichtigt werden. Im besten Fall hätten wir eine $(N-1) \times (2 \cdot N + 3)$ Matrix konstruiert. Ziel dieser Arbeit ist jedoch nicht, eine möglichst effiziente Lösung mittels Gauß-Elimination zu erarbeiten, deswegen genügt die hier vorgestellte Speicherersparnis unseren Anforderungen. Die reduzierte $((N+1)^2 \times (2N+3))$ Matrix $\widetilde{A}$ kann auch für allgemein große Gitter mit der Stützstellenanzahl N und für die Matrixelemente $\widetilde{a}_{i,j}$ angeschrieben werden.

[7] Dennoch sei an dieser Stelle erwähnt, dass auch hier der RAM-Speicher schnell ausgeschöpft ist, bei 512 Stützstellen werden zum Beispiel bereits $2,16$GB benötigt.

Daten : Anzahl der Stützstellen N
Ausgabe : Systemmatrix $\widetilde{A}$
solange $i \leq N$ **tue**
  $\widetilde{a}_{i,N+1} = 1$
  inkrementiere i
solange $(i > N) \wedge [i < (N^2 + N - 2)]$ **tue**
  Wiederhole folgendes $N - 1$ **mal:**
    $\widetilde{a}_{i,N+1} = 1$; `/* linker Randpunkt */`
    inkrementiere i
    Wiederhole folgendes $N - 1$ **mal:**
      $\widetilde{a}_{i,0} = -1 \cdot N^2$
      $\widetilde{a}_{i,N} = -1 \cdot N^2$
      $\widetilde{a}_{i,N+1} = 4 \cdot N^2$
      $\widetilde{a}_{i,N+2} = -1 \cdot N^2$
      $\widetilde{a}_{i,2N+2} = -1 \cdot N^2$
      inkrementiere i
    $\widetilde{a}_{i,N+1} = 1$; `/* rechter Randpunkt */`
    inkrementiere i
  inkrementiere i
solange $i < (N + 1)^2$ **tue**
  $\widetilde{a}_{i,N+1} = 1$
  inkrementiere i
Ende

Algorithmus 3.1.1 : Initialisierung der reduzierten Matrix $\widetilde{A}$

Anhand der Vorschrift 3.1.1 wird die Systemmatrix für unsere Zwecke initialisiert.[8] Um das Eliminationsverfahren von Gauß ausführen zu können, ist ein neuer Algorithmus - angewendet auf die Matrix $\widetilde{A}$ - zu entwerfen. Die Zeilen und Spaltenumformungen werden zu diesem Zweck angepasst.

i,j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	0	0	0
5	-9	0	0	-9	36	-9	0	0	-9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Abbildung 3.1.1: Erster Schritt des Eliminationsverfahren von Gauß

[8] Der Lösungsvektor u muss nicht explizit vorinitialisiert werden, bei Iterationsverfahren ist dieser Schritt jedoch notwendig.

Als ersten Schritt wird jedes Element der Zeile 1 mit $-\frac{\widetilde{a}_{5,0}}{\widetilde{a}_{1,4}}$ multipliziert.[9] Weiters werden die Elemente der Zeile 5 mit jenen der Zeile 1 addiert. Dieser Vorgang wird für die nächste Zeile, sowie anschließend für die Zeilen 2-4 wiederholt.

i,j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	9	0	0	0	0
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	0	0	0
5	0	0	0	-9	36	-9	0	0	-9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Abbildung 3.1.2: Zweiter Schritt des Eliminationsverfahren von Gauß

Anhand der Vorgehensweise aus den Abb. 3.1.1 und 3.1.2 beschriebenen Schritten wird eine obere Dreiecksmatrix, in unserem Fall eine Matrix mit den Einträgen $\widetilde{a}_{i,j} = 0$ für die Spalten $j < N$, berechnet. Abschließend wird von unten nach oben in die Gleichungen eingesetzt, wodurch man den Lösungsvektor u erhält. Für unser konkretes Problem ergibt sich der Algorithmus 3.1.2.

[9] Die Inhomogenität f ist ebenfalls mit $-\frac{\widetilde{a}_{5,0}}{\widetilde{a}_{1,4}}$ zu multiplizieren.

Funktionsname : Gauß
Daten : f_h; `/* Die Stützstellenanzahl` N `ist somit bekannt. */`
Ergebnis : u_h
initialisiere dieMatrix $\widetilde{A}$ mithilfe des Algorithmus 3.1.1
$x = 0$; `/* Schleifenoperator */`
$y = 0$; `/* Schleifenoperator */`
solange $i < (N+1)^2$ **tue**
  wenn *in der Zeile i mehr als* 1 *Element mit einem Wert* $\neq 0$ *ist* **dann**
    solange $x < N+1$ **tue**
      $Zeilenmultiplikator = \frac{\widetilde{a}_{i,x}}{\widetilde{a}_{i-N-1+x,N+1}}$
      solange $y \leq N+1$ **tue**
        $\widetilde{a}_{i,x+y} = \widetilde{a}_{i,x+y} - Zeilenmultiplikator \cdot \widetilde{a}_{i-N-1+x,N+1+y}$
        inkrementiere y; `/* rücke eine Zelle weiter rechts */`
      $f_i = f_i - Zeilenmultiplikator \cdot f_{i-N-1+x}$
      inkrementiere x; `/* rücke eine Zelle weiter rechts */`
  inkrementiere i; `/* rücke eine Zeile weiter */`
; `/* Die Matrix` $\widetilde{A}$ `hat jetzt folgende Einträge` $\widetilde{a}_{i,j} = 0$ `für` $j \leq N$`. */`
; `/* Es soll von unten nach oben eingesetzt werden. */`
$i = (N+1)^2$
$x = N+2$
solange $i > 0$ **tue**
  wenn *in der Zeile i mehr als* 1 *Element mit einem Wert* $\neq 0$ *ist* **dann**
    $Zwischenergebnis = 0$
    solange $x < (2N+1)$ **tue**
      $Zwischenergebnis = Zwischenergebnis + u_{i+x-N-1} \cdot \widetilde{a}_{i,x}$
      inkrementiere x
    $u_i = \frac{f_i - Zwischenergebnis}{\widetilde{a}_{i,N+1}}$
  dekrementiere i
reinitialisiere f_h; `/* Die Daten der Inhomogenität wurden im Verlauf */`
; `/* dieses Algorithmus mehrmals überschrieben. */`
Ende

Algorithmus 3.1.2 : Eliminationsverfahren von Gauß

3.2 Iterative Lösungsverfahren

Eine andere Herangehensweise als das direkte Verfahren bieten iterative Lösungsverfahren, die schrittweise der Lösung, ausgehend von einem Startwert u_h^0, näher kommen[10]. Die Herleitung für die im Folgenden vorgestellten Iterationsverfahren ist auf den Banach'schen Fixpunktsatz zurückzuführen. Wir werden nicht näher darauf eingehen, es sei auf [6] verwiesen. Außerdem werden im Rahmen dieser Arbeit Konvergenzbeweise nicht näher behandelt[11].

Das Einzelschrittverfahren (Gauß-Seidel-Verfahren):

Die Formel für die $(m+1)$-te Iteration zur Lösung des Gleichungssystems $Au = f$ lautet:

$$u_i^{(m+1)} := \frac{1}{a_{ii}}\Big[f_i - \sum_{k=1}^{i-1}(a_{ik} \cdot u_k^{(m+1)}) - \sum_{k=i+1}^{n}(a_{ik} \cdot u_k^{(m)})\Big] \quad [6] \tag{3.2.1}$$

Bei Anwendung der Formel 3.2.1 ist darauf zu achten, dass die Matrix A regulär ist, es also keine Elemente $a_{i,i} = 0$ gibt. Da die Elemente und die Positionen der Elemente unserer dünn besetzten Systemmatrix bekannt sind, und - im Gegensatz zum Eliminationsverfahren - lediglich der Lösungsvektor verändert wird, muss auf die Matrix A nicht explizit zugegriffen werden, dieser Umstand löst sämtliche Speicherkomplikationen. Zur Lösung unserer Problemstellung mit dem Einzelschrittverfahren ist der Algorithmus 3.2.1 zu verwenden.

Funktionsname : GaußSeidel
Daten : u_h^m, f_h
Ergebnis : u_h^{m+1}
solange $i < (N+1)^2$ **tue**
  wenn *i ein innerer Punkt ist;* `/* laut Abb. 2.2.3 */`
  dann
    $u_i^{m+1} = \frac{f_i + u_{i-1}^{m+1} \cdot N^2 + u_{i+1}^{m} \cdot N^2 + u_{i-N-1}^{m+1} \cdot N^2 + u_{i+N+1}^{m} \cdot N^2}{4 \cdot N^2}$
  inkrementiere i; `/* rücke um einen Punkt weiter */`
Ende

Algorithmus 3.2.1 : Gauß-Seidel-Iteration

Für die vollständige Lösung unseres Modellproblems ist dieses Verfahren bei vielen Stützstellen ungeeignet, da der entstehende Fehler $u_i^{(m+1)} - u_i^{(m)}$ bei größeren Gleichungssystemen zu langsam kleiner wird. Als Randbemerkung sei an dieser Stelle erwähnt, dass die Gauß-Seidel Iteration für das Parallelrechnen nicht brauchbar ist, da jedes errechnete Ergebnis für danach folgende Berechnungen Verwendung findet. Für unser Ziel - die

[10] In unserem konkreten Fall ist die initiale Startlösung u_h^0 mit Werten zwischen 0 und 1 willkürlich besetzt.

[11] Die Konvergenzeigenschaft der in diesem Kapitel behandelten iterativen Verfahren ist für das Mehrgitterverfahren nicht relevant, viel wichtiger ist die in Abschnitt 4.1 diskutierte Glättungseigenschaft.

Lösung der Poisson-Gleichung mit dem Mehrgitterverfahren - ist die Tatsache besonders wichtig, dass der Fehler mit jedem Iterationsschritt sukzessive geglättet wird. Wir werden dieses Phänomen an späterer Stelle (4.1) aufgreifen und vertiefen. Das Gauß-Seidel Verfahren ist dennoch sehr einfach und schnell zu implementieren, weshalb es bei kleinen Gleichungssystemen Anwendung findet.

Das SOR-Verfahren (Gauß-Seidel-Relaxation):

Das „Successive Over-Relaxation“ Verfahren funktioniert ähnlich wie die Gauß-Seidel Iteration, jedoch spielt zusätzlich der Relaxationsparameter ω eine elementare Rolle. Die zeitliche und rechnerische Geschwindigkeit, in der die Konvergenz erreicht wird[12], wird durch diesen Parameter erheblich beeinflusst. Die Formel für die $(m+1)$-te Iteration des SOR-Verfahrens lautet:

$$u_i^{(m+1)} := (1-\omega)\cdot u_i^{(m)} + \frac{\omega}{a_{ii}}\Big[f_i - \sum_{k=1}^{i-1}(a_{ik}\cdot u_k^{(m+1)}) - \sum_{k=i+1}^{n}(a_{ik}\cdot u_k^{(m)})\Big] \quad [6] \quad (3.2.2)$$

Dieses Verfahren wird zur Gauß-Seidel Iteration, wenn die Bedingung $\omega = 1$ gesetzt ist, darum hat die Einschränkung $a_{i,i} = 0$ ebenfalls Gültigkeit. Der Algorithmus 3.2.1 wird um den Parameter ω ergänzt:

Funktionsname : SOR
Daten : u_h^m, f_h
Ergebnis : u_h^{m+1}
solange $i < (N+1)^2$ **tue**
  wenn i *ein innerer Punkt ist;* `/* laut Abb.: 2.2.3 */`
  dann
    $u_{alt_i} = u_i^m$
    $u_i^{m+1} = u_{alt_i}\cdot(1-\omega) + \omega\cdot\frac{f_i + u_{i-1}^{m+1}\cdot N^2 + u_{i+1}^{m}\cdot N^2 + u_{i-N-1}^{m+1}\cdot N^2 + u_{i+N+1}^{m}\cdot N^2}{4\cdot N^2}$
  inkrementiere i; `/* rücke um einen Punkt weiter */`
Ende

Algorithmus 3.2.2 : SOR-Verfahren

Für unsere konkrete Problemstellung ist der optimale Relaxationsparameter von der Gitterweite $h = \frac{1}{N}$ laut [7, S. 32] abhängig und ist - damit die Lösung am schnellsten unseren Konvergenzanforderungen entspricht - folgendermaßen zu wählen:

$$\omega = \frac{2}{1+\sin(\pi\cdot h)} \quad (3.2.3)$$

[12] Ab wann die Lösung unsere Anforderungen (Konvergenz) erfüllt, wird in Kapitel 3.3 erörtert. Wir werden diese Geschwindigkeit in Zukunft Konvergenzgeschwindigkeit nennen.

Das Gesamtschrittverfahren (Jacobiverfahren):

Dieses Verfahren von Carl Gustav Jakob Jacobi hat ebenfalls einen ähnlichen Charakter wie die Gauß-Seidel Iteration. Es konvergiert auch für das Gleichungssystem $Au = f$ in einer bestimmten Anzahl Schritten, abhängig von der Schrittweite. Die Matrix A ist für die Anwendung des Jacobiverfahrens auf Regularität zu überprüfen.

$$u_i^{(m+1)} := \frac{1}{a_{ii}}\Big[f_i - \sum_{i \neq k} (a_{ik} \cdot u_k^{(m)})\Big] \quad [6] \tag{3.2.4}$$

Für das Jacobiverfahren ist der Algorithmus 3.2.3 zu verwenden.

Funktionsname : Jacobi
Daten : u_h^m, f_h
Ergebnis : u_h^{m+1}
$u_{neu} = u^m$
solange $i < (N+1)^2$ **tue**
  wenn i *ein innerer Punkt ist;* `/* laut Abb.: 2.2.3 */`
  dann
    $u_{neu_i}^{m+1} = \frac{f_i + u_{i-1}^m \cdot N^2 + u_{i+1}^m \cdot N^2 + u_{i-N-1}^m \cdot N^2 + u_{i+N+1}^m \cdot N^2}{4 \cdot N^2}$
  inkrementiere i; `/* rücke um einen Punkt weiter */`
Ende

Algorithmus 3.2.3 : Jacobi-Verfahren

3.3 Ergebnisse

Mithilfe der Verfahren gemäß den Abschnitten 3.1 und 3.2 wird das Modellproblem unterschiedlich gelöst. Jedes Verfahren hat optimale Anwendungsgebiete, für unser Problem sieht die Lösung abhängig von N wie in Abb. 3.3.1 oder Abb. 3.3.2 aus.[13]

[13] Diese Grafiken wurden mit dem Programm gnuplot5.0 mithilfe des pm3d Befehls aufgenommen.

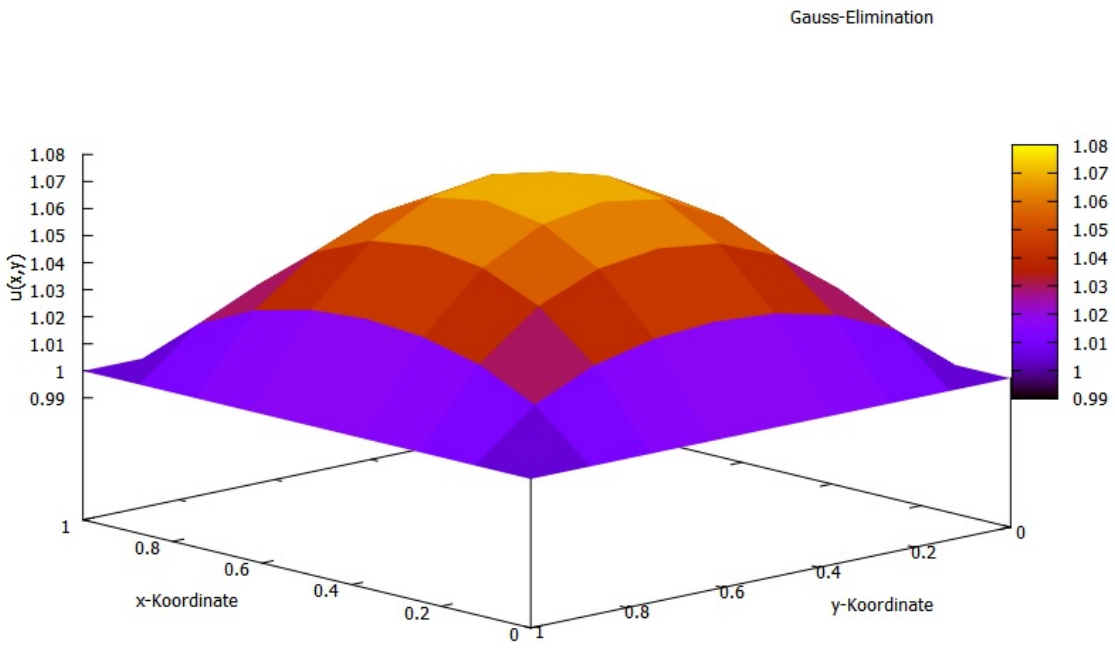

Abbildung 3.3.1: Lösung mittels Eliminationsverfahren von Gauß bei $N = 8$

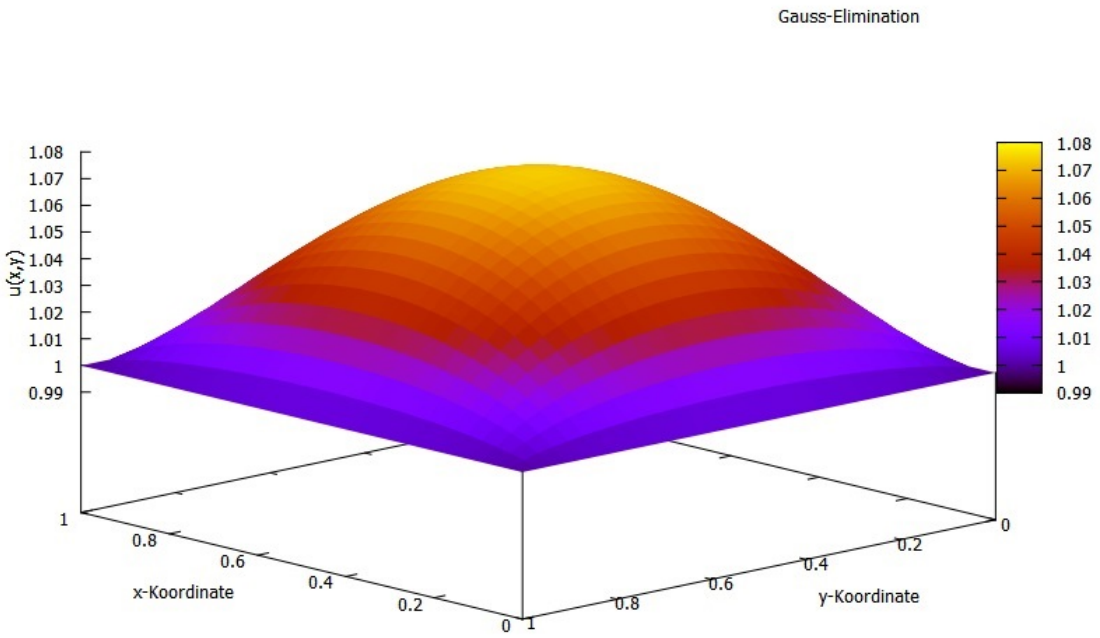

Abbildung 3.3.2: Lösung mittels Eliminationsverfahren von Gauß bei $N = 32$

Mit wenigen Stützstellen (wie in Abb. 3.3.1), zum Beispiel $N = 8$ können lediglich 81 Punkte aufgenommen werden, wobei 32 Punkte davon auf den Rand fallen. Dieser Umstand führt zu hohen Diskretisierungsfehlern (siehe 2.2, Taylorreihen-Entwicklung).

Die Gauß-Elimination berechnet den numerisch gesehen optimalen Wert zu jedem Punkt am Gitter. Anhand der Schrittweite h kann die Auflösung erhöht werden (zu sehen im Vergleich zwischen Abb. 3.3.1 und Abb. 3.3.2). Ab wie vielen Iterationsschritten die Lösung der Algorithmen in 3.2 jener Lösung in 3.1 genügt, wird anhand des bereits erwähnten Konvergenzbegriffes erörtert. Zu diesem Zweck führen wir das Residuum ein:

$$r^m := f - A \cdot u^m \tag{3.3.1}$$

Ziel soll es sein, nach jedem Iterationsschritt (m) das Residuum r^m zu berechnen. Der Betrag der Maximumsnorm $\|r\|_{max} := \underbrace{\max(r_i)}_{\text{für } 0 \leq i \leq (N+1)^2}$ soll Aufschluss darüber geben, ob ein weiterer Iterationsschritt erforderlich ist, oder ob die Lösung dieser Norm entspricht. Um Vergleiche zwischen den Verfahren anstellen zu können, werden wir die Konvergenz untersuchen. Zwei Aspekte sind zu berücksichtigen, die Konvergenzgeschwindigkeit anhand der Anzahl an Iterationsschritten[14] und der absoluten Berechnungszeit. Diese zwei Parameter werden bei Variation der Norm und in Abhängigkeit der Stützstellenanzahl aufgezeichnet.[15]

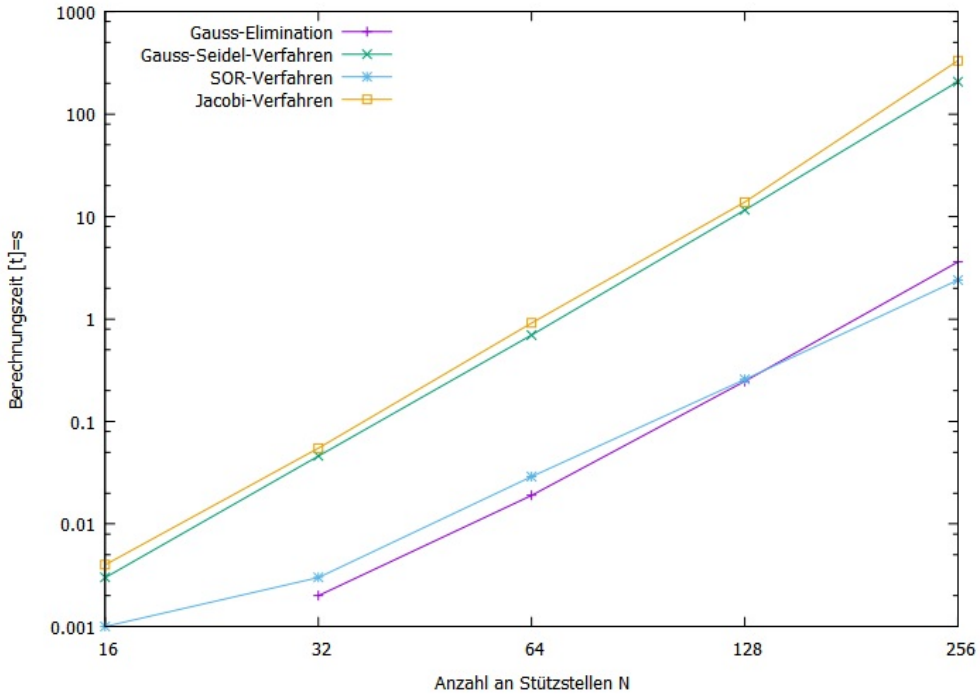

Abbildung 3.3.3: Konvergenzgeschwindigkeit der Zeit/Stützstellenanzahl

[14] Die Gauß-Elimination ist davon ausgenommen.

[15] Die Graphen wurden mithilfe eines Intel Core i7-6500U Prozessors, mit einem 64-bit-basiertem Betriebssystem aufgenommen.

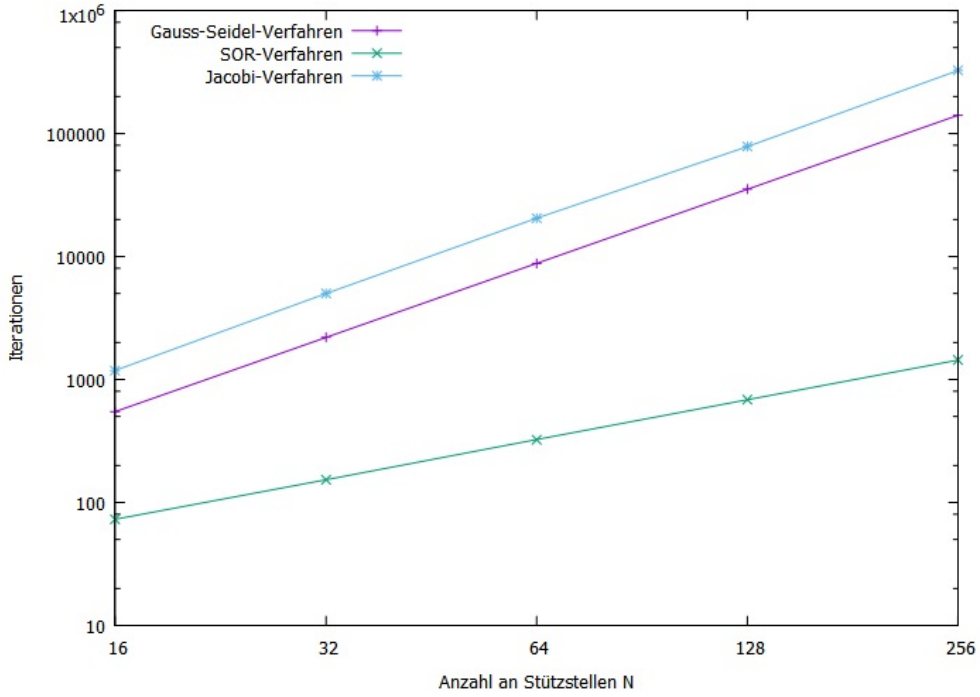

Abbildung 3.3.4: Konvergenzgeschwindigkeit der Iterationen/Stützstellenanzahl

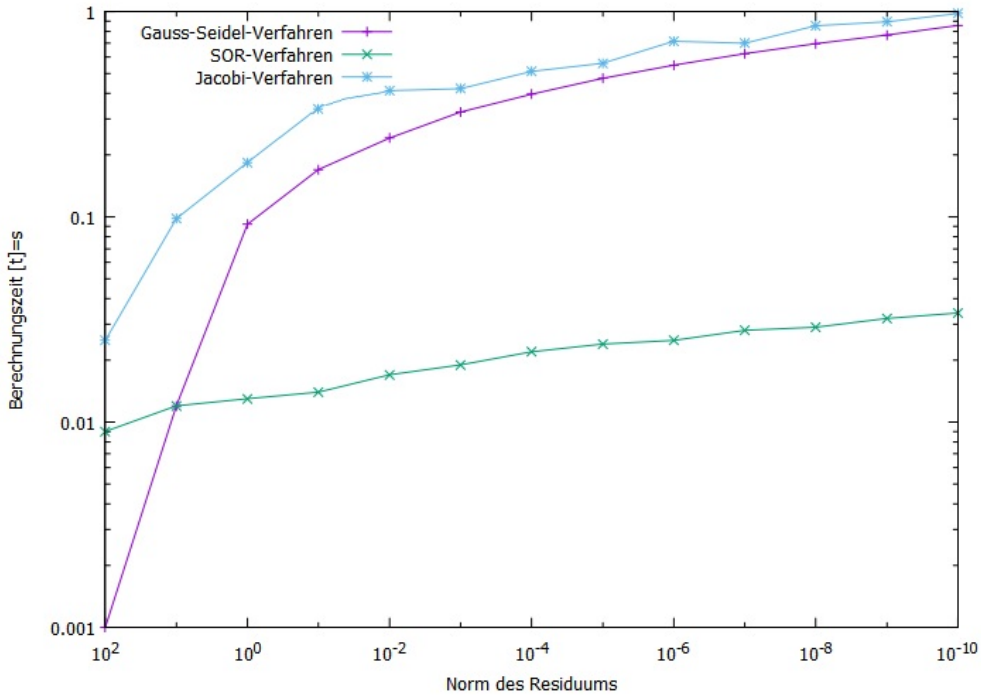

Abbildung 3.3.5: Konvergenzgeschwindigkeit der Zeit/Norm

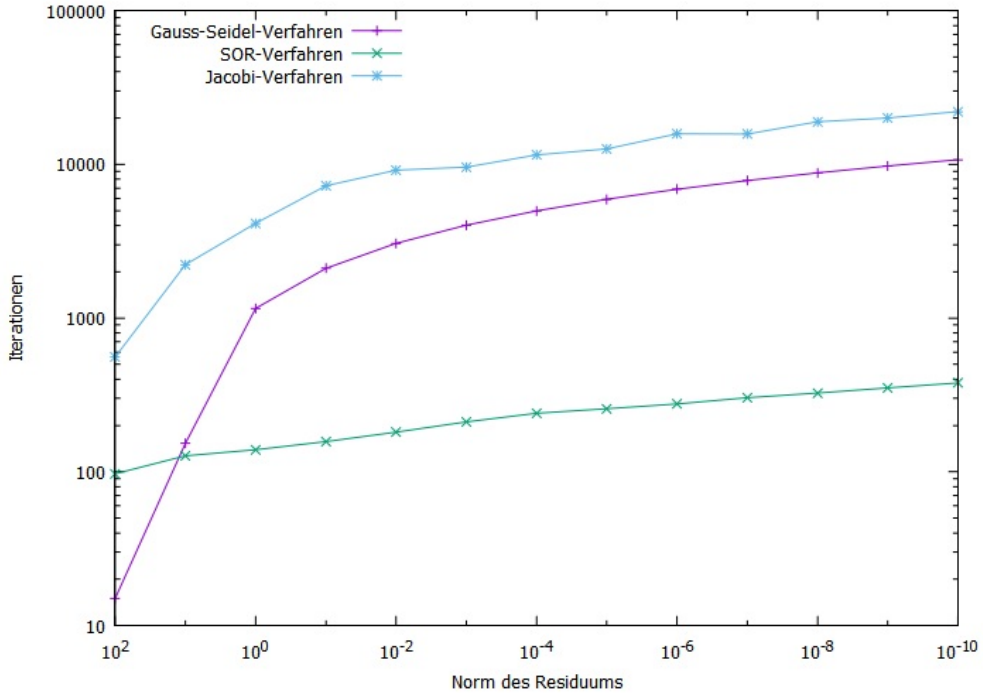

Abbildung 3.3.6: Konvergenzgeschwindigkeit der Iterationen/Norm

Wie wir anhand der Graphen 3.3.3, 3.3.4, 3.3.5, 3.3.6 sehen können, ist das SOR-Verfahren im Rahmen dieser Untersuchungen den anderen Verfahren überlegen[16]. Die Jacobi-Iteration bietet keine zufriedenstellende Lösungsmöglichkeit, die Berechnungszeit und die Anzahl an Iterationen sind wesentlich höher als bei anderen Verfahren. Die Speicherallozierung des mehrdimensionalen Arrays ist beim Eliminationsverfahren von Gauß zeitlich aufwändig, weswegen das SOR-Verfahren bei größerer Anzahl an Stützstellen schneller vorankommt.

[16]Bei Anwendung der Formel 3.2.3

4 Mehrgitterverfahren

Es wurden bereits früh Wege gesucht, die Ausführungszeiten und die Anzahl an Iterationen mit anderen Verfahren (wie zum Beispiel Nested Iteration) erheblich zu senken. In den 1960er Jahren ist schließlich die Idee des Mehrgitterverfahrens zum ersten Mal aufgegriffen worden [7, S. 23f]. Die Grundidee besteht darin, das Problem auf einem gröberen Gitter mit weniger Aufwand und Rechenzeit mithilfe herkömmlicher direkter oder indirekter Verfahren zu lösen und anschließend die Lösung wieder auf das feinere Gitter zu interpolieren, ohne signifikante Fehler zu machen. Die dafür benötigten Konzepte werden in den folgenden Abschnitten Schritt für Schritt erarbeitet.

4.1 Fehlerglättung

Bereits im Kapitel 3.2 ist im Rahmen der Gauß-Seidel Iteration der Begriff der Fehlerglättung aufgegriffen worden. Für den Fehler v^m gilt:

$$v^m := u - u^m \tag{4.1.1}$$

$$A \cdot v^m = r^m \tag{4.1.2}$$

Der Index m gibt die Zahl des aktuellen Iterationsschrittes an. Der Parameter u soll die numerisch exakte Lösung sein, u^m jene, nach dem m-ten Iterationsschritt, sowie r^m das aus u^m resultierende Residuum. Der Algorithmus zur Berechnung des Residuums r lässt sich für unser Problem folgendermaßen implementieren:

Funktionsname : Residuum
Daten : u_h, f_h
Ergebnis : r
initialisiere Residuum r
solange $i < (N+1)^2$ **tue**
  wenn i *ein innerer Punkt ist;* `/* laut Abb.: 2.2.3 */`
  dann
    $r_i = f_i - u_i \cdot 4N^2 + u_{i-1} \cdot N^2 + u_{i+1} \cdot N^2 + u_{i-N-1} \cdot N^2 + u_{i+N+1} \cdot N^2$
  inkrementiere i; `/* rücke um einen Punkt weiter */`
Ende

Algorithmus 4.1.1 : Berechnung des Residuums

Da die Gleichung 4.1.2 die gleiche Form wie unsere Ausgangsgleichung $Au = f$ hat, kann der Fehler mit denselben Algorithmen berechnet und dargestellt werden, wie die Lösung selbst. Diese Eigenschaft ist die wichtigste Zutat für das Mehrgitterverfahren.

Wir werden im Folgenden die Glättungseigenschaften unserer ausgewählten Verfahren (Gauß-Seidel, SOR, Jacobi) untersuchen, um eine qualitative Wahl aus diesen treffen zu können. Wir teilen unseren diskreten Differentialoperator in die Teile $m+1$ und m auf, in jene Anteile, die bereits im Zuge des $m+1$-ten Iterationsschrittes berechnet wurden,

und jene m, die noch zu berechnen sind.

$$L_h = -\Delta_h = L_h^{m+1} + L_h^m = -\Delta_h^{m+1} - \Delta_h^m \tag{4.1.3}$$

Eingesetzt in unsere Differentialgleichung erhalten wir:

$$-\Delta_h^{m+1} u_h^{m+1} - \Delta_h^m u_h^m = f_h \tag{4.1.4}$$

Subtrahiert man Gleichung 4.1.4 von der Gleichung $A_h u_h = f_h$, so erhält man:

$$-\Delta_h^{m+1} v_h^{m+1} - \Delta_h^m v_h^m = 0 \tag{4.1.5}$$

Es wird der Glättungsoperator S_h eingeführt:

$$S_h = \frac{v_h^m}{v_h^{m+1}} \tag{4.1.6}$$

Wir können für die Operatoren L_h die Eigenwertprobleme anschreiben, wobei $e^{j(\varphi,\psi)\cdot\frac{(x,y)}{h}}$ bereits als Eigenvektor laut [7, S. 99] bestimmt ist[17]:

$$L_h^{m+1} e^{j(\varphi,\psi)\cdot\frac{(x,y)}{h}} = \widetilde{L}_h^{m+1}(\varphi,\psi) e^{j(\varphi,\psi)\cdot\frac{(x,y)}{h}} \tag{4.1.7}$$

$$L_h^{m} e^{j(\varphi,\psi)\cdot\frac{(x,y)}{h}} = \widetilde{L}_h^{m}(\varphi,\psi) e^{j(\varphi,\psi)\cdot\frac{(x,y)}{h}} \tag{4.1.8}$$

$$\widetilde{S}_h(\varphi,\psi) := -\frac{\widetilde{L}_h^m(\varphi,\psi)}{\widetilde{L}_h^{m+1}(\varphi,\psi)} \tag{4.1.9}$$

Die Parameter φ, ψ sind als Frequenzen der lokalen Fourier Analyse zu interpretieren. Analog zum Glättungsoperator bezeichnen wir den Quotienten aus den Eigenwerten $\widetilde{L}_h$ im Folgenden als Verstärkungsfaktor $\widetilde{S}_h(\varphi,\psi)$. Um anschließend die Glättungseigenschaft bezüglich φ, ψ zu untersuchen, wird der Glättungsfaktor μ eingeführt:

$$\mu := \sup\left\{\left|\widetilde{S}_h(\varphi,\psi)\right|\right\} \tag{4.1.10}$$

Mit diesem Werkzeug lassen sich für unsere drei Verfahren die Glättungsfaktoren bestimmen.

Das Einzelschrittverfahren (Gauß-Seidel-Verfahren):

Nach Auflösung des 5-Punkte Sterns für das Gauß-Seidel Verfahren mit der Gleichung 3.2.1, erhält man für den gesplitteten Differentialoperator L_h:

$$\begin{aligned} L_h^{m+1} &= \frac{1}{h^2}\begin{bmatrix} & 0 & \\ -1 & 4 & 0 \\ & -1 & \end{bmatrix} \\ L_h^{m} &= \frac{1}{h^2}\begin{bmatrix} & -1 & \\ 0 & 0 & -1 \\ & 0 & \end{bmatrix} \end{aligned} \tag{4.1.11}$$

[17] Die $\widetilde{L}_h$ gekennzeichneten Operatoren sind die dazugehörigen Eigenwerte.

Die dazugehörigen Eigenwerte ergeben sich zu:

$$\begin{aligned} \widetilde{L}_h^{m+1}(\varphi,\psi) &= \frac{1}{h^2}\left(4 - e^{-j\varphi} - e^{-j\psi}\right) \\ \widetilde{L}_h^{m}(\varphi,\psi) &= -\frac{1}{h^2}\left(e^{j\varphi} + e^{j\psi}\right) \end{aligned} \tag{4.1.12}$$

Aus Gleichung 4.1.12 bestimmen wir den Glättungsfaktor:

$$\mu = \sup\left\{\left|\frac{e^{j\varphi} + e^{j\psi}}{e^{-j\varphi} + e^{-j\psi-4}}\right|\right\} = \frac{1}{2} \text{ für } (\varphi,\psi) = \left(\frac{\pi}{2}, \arccos\left(\frac{4}{5}\right)\right) \text{ [7, S. 105]} \tag{4.1.13}$$

Der Betrag des Verstärkungsfaktors $\left|\widetilde{S}_h(\varphi,\psi)\right|$ lässt sich außerdem über die Frequenzen φ und ψ darstellen:

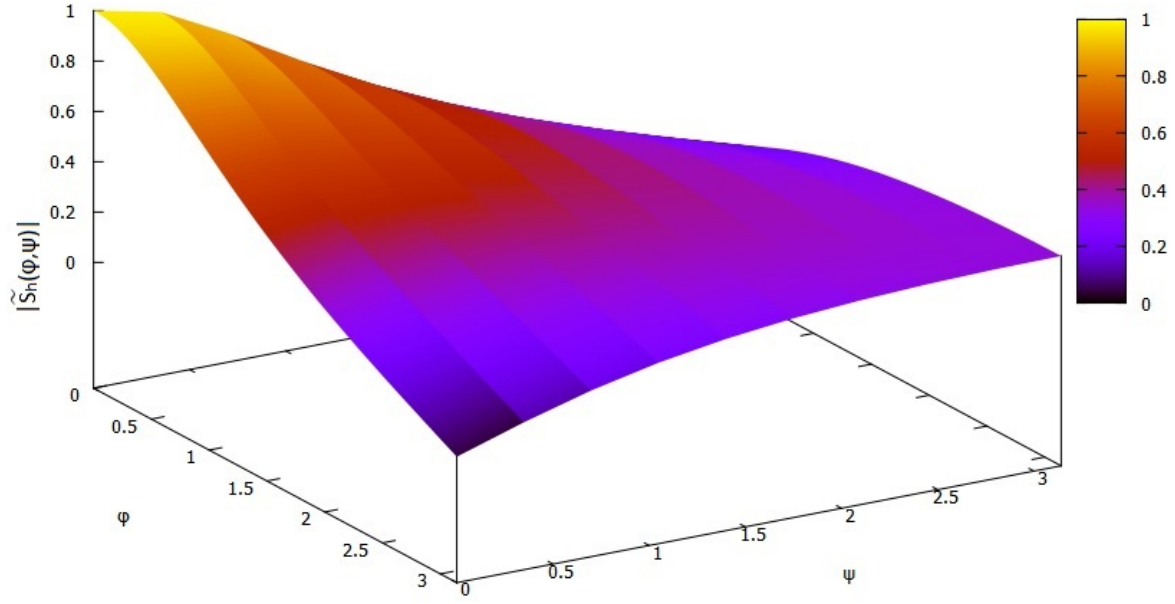

Abbildung 4.1.1: Glättungseigenschaften der Gauß-Seidel Iteration

In Abb. 4.1.1 sieht man, dass höhere Frequenzen besser gedämpft werden als niedrigere. Das bedeutet, dass das Gauß-Seidel Verfahren für unsere Anwendungen als Glättungsverfahren geeignet ist. Wir wollen die anderen Iterationsverfahren ebenfalls dahingehend untersuchen.

Das SOR-Verfahren (Gauß-Seidel-Relaxation):

Auch für dieses Verfahren werden die Matrizen der Differentialoperatoren aufgestellt:

$$\begin{aligned} L_h^{m+1} &= \frac{1}{h^2}\begin{bmatrix} & 0 & \\ -1 & \frac{4}{\omega} & 0 \\ & -1 & \end{bmatrix} \\ L_h^{m} &= \frac{1}{h^2}\begin{bmatrix} & -1 & \\ 0 & 4\left(1-\frac{1}{\omega}\right) & -1 \\ & 0 & \end{bmatrix} \end{aligned} \tag{4.1.14}$$

Die dazugehörigen Eigenwerte ergeben sich zu:

$$\begin{aligned} \widetilde{L}_h^{m+1}(\varphi,\psi) &= \frac{1}{h^2}\left(\frac{4}{\omega} - e^{-j\varphi} - e^{-j\psi}\right) \\ \widetilde{L}_h^{m}(\varphi,\psi) &= -\frac{1}{h^2}\left(e^{j\varphi} + e^{j\psi} - 4\left(1-\frac{1}{\omega}\right)\right) \end{aligned} \tag{4.1.15}$$

Bei diesem Verfahren ist eine Abhängigkeit des Glättungsfaktors vom Relaxationsparameter ω zu bemerken, bestenfalls wird bei $\omega = 1$ der Wert $\mu = \frac{1}{2}$ erreicht [7, S. 106, Figure 4.2]. Bezüglich der Glättungseigenschaften bringt es keinen Vorteil, dieses Verfahren der Gauß-Seidel Iteration vorzuziehen und somit den Relaxationsparameter miteinzubeziehen.

Das Gesamtschrittverfahren (Jacobiverfahren):

Das Jacobiverfahren werden wir hinsichtlich der Glättungseigenschaften folgendermaßen analysieren:

$$\begin{aligned} L_h^{m+1} &= \frac{1}{h^2}\begin{bmatrix} & 0 & \\ 0 & 4 & 0 \\ & 0 & \end{bmatrix} \\ L_h^{m} &= \frac{1}{h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \end{aligned} \tag{4.1.16}$$

Wir können den Verstärkungsfaktor anschreiben:

$$\widetilde{S}_h(\varphi,\psi) = \frac{\cos(\varphi)+\cos(\psi)}{2} \tag{4.1.17}$$

Beim Jacobiverfahren wird ein Glättungsfaktor von $\mu = 1$ erreicht. Es lässt sich die Glättungseigenschaft über die Frequenzen laut Abb. 4.1.2 darstellen.

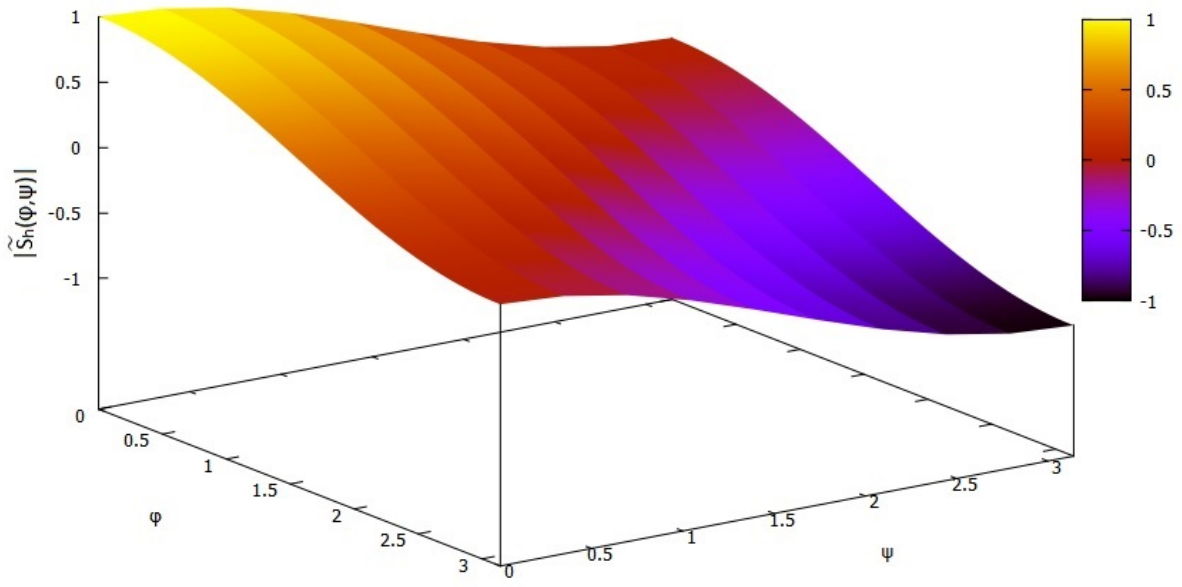

Abbildung 4.1.2: Glättungseigenschaften der Jacobi Iteration

Das Einzelschrittverfahren ist gemäß dieser Überlegungen eine sinnvolle Wahl als Glätter. Wir werden das Gauß-Seidel Verfahren auf unser Gleichungssystem anwenden und beobachten, was mit dem Fehler nach m Iterationsschritten passiert. Anhand der Abbildungen 4.1.3, 4.1.4, 4.1.5 erkennt man, dass der Fehler (in diesem Fall für $N = 32$ Stützstellen) nach m Iterationsschritten immer glatter wird. Ohne dabei große Fehler zu machen, sind wir in der Lage, Berechnungen auf einem gröberen Gitter durchzuführen, da sich der Fehler (zu sehen anhand Abbildung 4.1.5) bei $m = 10$ von Punkt zu Punkt nicht mehr so stark ändert wie bei $m = 0$ (Abb. 4.1.3).

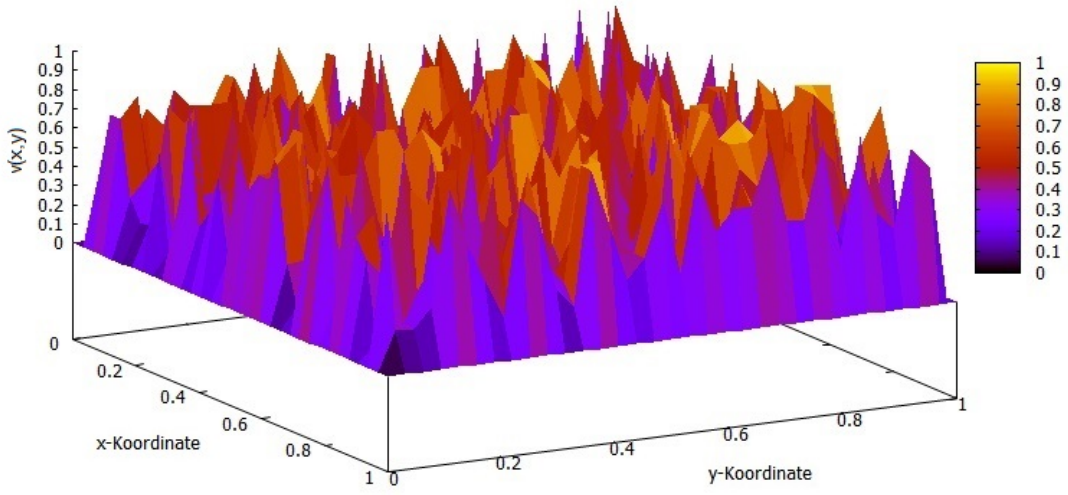

Abbildung 4.1.3: Fehler v ohne Glättung des Startvektors

Der Startlösung wurden willkürlich Werte zwischen 0 und 1 zugewiesen, der Rand ist davon nicht betroffen, daher gilt dort $v_{\partial G_h} = 0$. An dieser Stelle ist es noch nicht sinnvoll, zwischen den Punkten zu interpolieren, die hohen Frequenzen müssen erst geglättet werden.

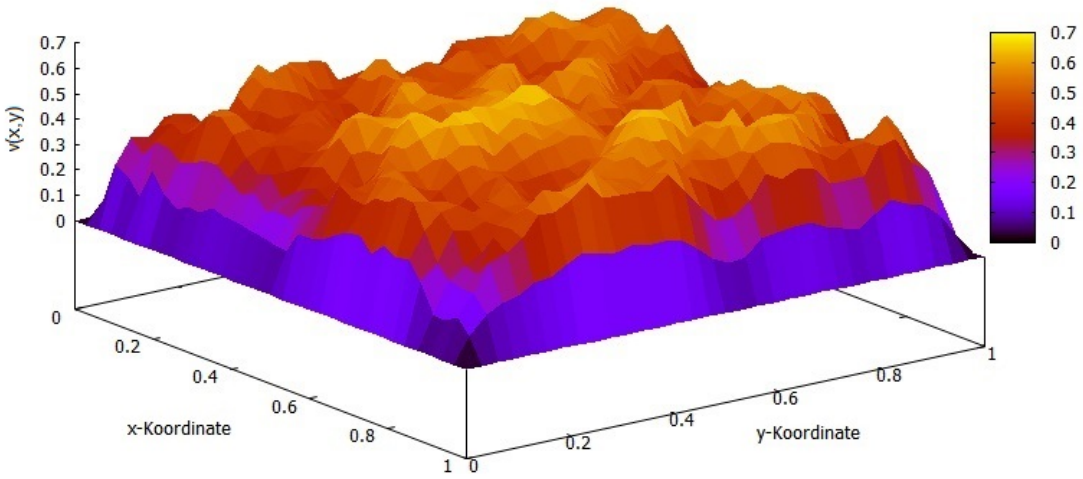

Abbildung 4.1.4: Fehler v nach $m = 2$ Iterationen

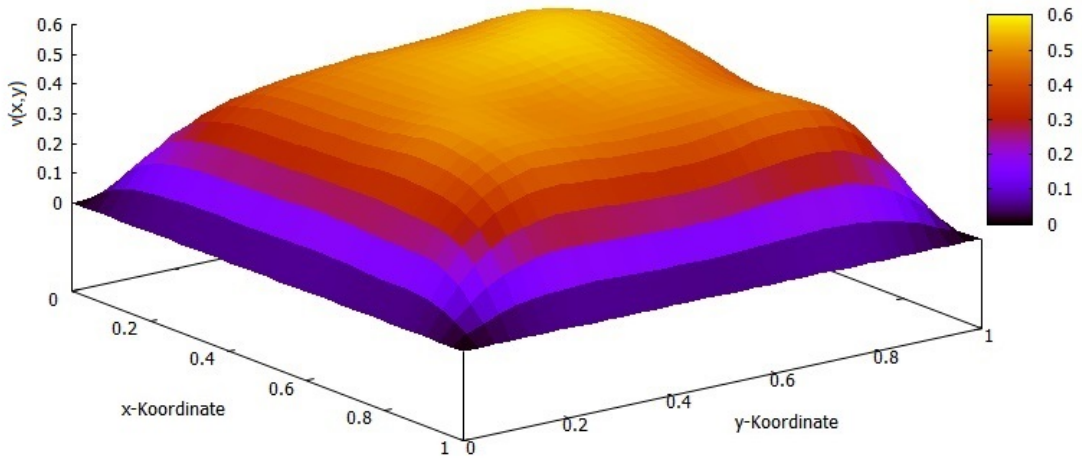

Abbildung 4.1.5: Fehler v nach $m = 10$ Iterationen

Nach $m = 10$ Iterationen ist der Fehler weitgehend geglättet, die Restriktion auf ein gröberes Gitter kann vorgenommen werden.

4.2 Gitterstrukturen

Bisher sind wir von einem homogenen Gitter mit äquidistantem Abstand h in jede Raumrichtung ausgegangen. Es gibt auch andere Methoden, um den Raum zu diskretisieren, nachzulesen in [7, S.4f]. Um auf ein gröberes Gitter zu gelangen, benötigt man den sogenannten Restriktionsoperator. Im einfachsten Fall wird die Gitterweite h auf jeder Achse verdoppelt, es wird mit jedem zweiten Punkt weiter gerechnet[18]. Umgekehrt ist man in der Lage, zwischen zwei Gitterpunkten zu interpolieren, man verwendet den Prolongationsoperator, um wieder auf einen feineren Gitterlevel zurückzurechnen.

4.2.1 Restriktion

Damit jeder zweite Gitterpunkt aus dem Gitter „übergangen“ wird, ist die Vorschrift $\log_2(N) \in \mathbb{N}$ einzuhalten, da sonst bei der Halbierung der Gitterweite Probleme auftreten. Die Anzahl der Stützstellen N muss durch eine Zahl 2^x $x \in \mathbb{N}$ teilbar sein.

[18] Auch an dieser Stelle gibt es wieder weitere Möglichkeiten, siehe [7, S.41].

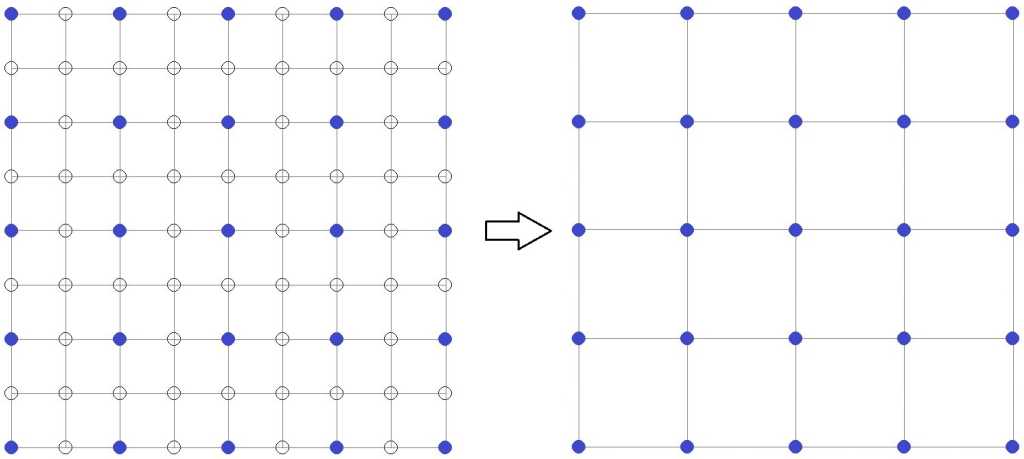

Abbildung 4.2.1: Halbierung des Gitters

Die einfachste Methode, um laut Abb. 4.2.1 von einem G_h,- auf ein G_{2h} Gitter zu restringieren, ist die Injektion. Bei diesem Verfahren wird jeder zweite Punkt übernommen, alle anderen Gitterpunkte sind dabei zu verwerfen, dieser Umstand bewirkt intuitiv einen Informationsverlust (wie viel Information verloren geht, hängt stark von der Anzahl an Glättungsschritten ab). Sollte das Gitter nicht zureichend vorgeglättet worden sein, oder es durch verschiedene Gründe mehrere „Ausreißer“ geben, so stellt diese Möglichkeit keine guten Konvergenzeigenschaften dar. Eine verbesserte Möglichkeit ist die Restriktion via Halbgewichtung, die direkt benachbarten Punkte werden für Berechnungen einbezogen. Die Vollgewichtung bietet intuitiv die beste Möglichkeit, um die Restriktion mit den wenigsten Fehlern durchzuführen, die Werte aller direkt umliegenden Punkte (auch diagonale Punkte) werden berücksichtigt. Welchen Restriktionsoperator wir zu wählen haben, hängt von verschiedenen Faktoren ab, wir werden diese Thematik an späterer Stelle (Abschnitt 5.1) wieder aufgreifen. Betrachten wir den mittleren Punkt der Abb. 4.2.2, so werden für die jeweiligen Verfahren folgende umliegende Punkte berücksichtigt:

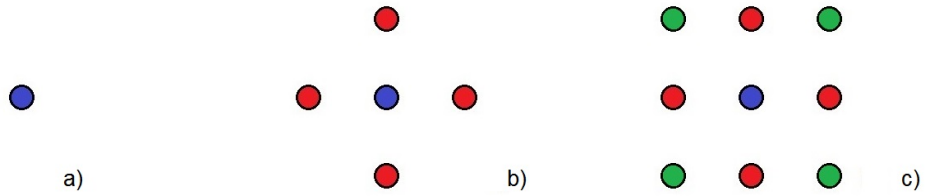

Abbildung 4.2.2: a) Injektion b) Halbgewichtung c) Vollgewichtung

Anhand der Abb. 4.2.2 lässt sich der Restriktionsoperator als geometrische Schablonennotation anschreiben:

$$v_{2h}^{Injektion} = \begin{bmatrix} & & \\ & 1 & \\ & & \end{bmatrix}_{h}^{2h} \cdot v_h \tag{4.2.1}$$

$$v_{2h}^{Halbgewichtung} = \frac{1}{8} \begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}_{h}^{2h} \cdot v_h \tag{4.2.2}$$

$$v_{2h}^{Vollgewichtung} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{h}^{2h} \cdot v_h \tag{4.2.3}$$

Unter Verwendung der Vollgewichtung werden wir die Restriktion für unsere Problemstellung laut Algorithmus 4.2.1 implementieren.

Funktionsname : Restriktion
Daten : Residuum auf feinem Gitter $r_h \in G_h$
Ergebnis : Residuum auf grobem Gitter $r_{2h} \in G_{2h}$
solange $i < (N+1)^2$ **tue**
  Zeilenindex x der aktuellen Position i
  Spaltenindex y der aktuellen Position i
  wenn *i ein innerer Punkt ist;* `/* laut Abb.: 2.2.3 */`
  dann
    ermittle von i die aktuelle Zeile/Spalte (x/y)
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x-1)\cdot(N+1)+(y-1)}}{16}$; `/* x-h,y-h */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x-1)\cdot(N+1)+(y)}}{8}$; `/* x-h,y */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x-1)\cdot(N+1)+(y+1)}}{16}$; `/* x-h,y+h */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x)\cdot(N+1)+(y-1)}}{8}$; `/* x,y-1 */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x)\cdot(N+1)+(y)}}{4}$; `/* x,y */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x)\cdot(N+1)+(y+1)}}{8}$; `/* x,y+1 */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x+1)\cdot(N+1)+(y-1)}}{16}$; `/* x+h,y-h */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x+1)\cdot(N+1)+(y)}}{8}$; `/* x+h,y */`
    $r_{2h,i} = r_{2h,i} + \frac{r_{h,(x+1)\cdot(N+1)+(y+1)}}{16}$; `/* x+h,y+h */`
  inkrementiere i; `/* rücke um einen Punkt weiter */`
Ende

Algorithmus 4.2.1 : Restriktion mittels Vollgewichtung

4.2.2 Prolongation

Um nach Berechnungen am groben Gitter wieder auf das feinere Gitter zu interpolieren, wird die Prolongation ausgeführt. Wir werden die Vorgehensweise für die bilineare Interpolation beschreiben. Mithilfe dieser Methode der Interpolation wird der Wert an einem Punkt aus dem gewichteten Mittel der Werte aller umliegenden Punkte berechnet. Auch an dieser Stelle gibt es wieder (analog zur Restriktion) Alternativen, indem zum Beispiel nur die vier direkt umliegenden Punkte für die Berechnungen berücksichtigt werden. Außerdem gibt es die lineare Interpolation, die auf einem Dreieck, anstatt auf einem Viereck definiert ist. Näheres dazu ist in [2, S. 146] zu finden.

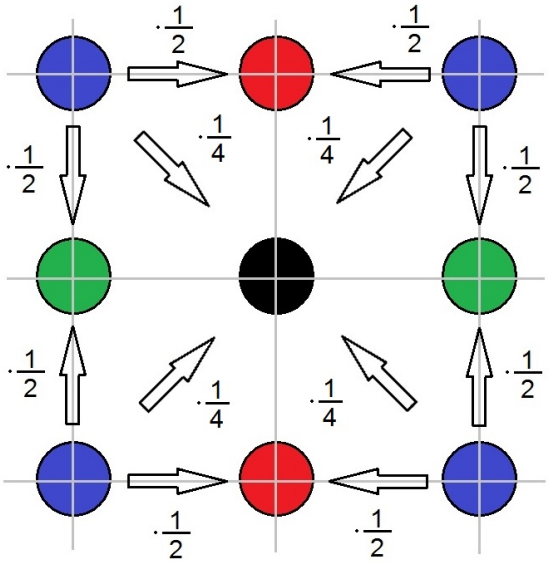

Abbildung 4.2.3: Bilineare Interpolation

Anhand der Abb. 4.2.3 lässt sich der Prolongationsoperator für die bilineare Interpolation ebenfalls als Schablonennotation anschreiben:

$$v_h^{bilinearesInterpolation} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2h}^{h} \cdot v_{2h} \tag{4.2.4}$$

Es ist für die Berechnung zwischen vier Punkten zu unterscheiden (tatsächlich vorhandene Punkte: a, Interpolation zwischen zwei waagrechten Punkten: b, Interpolation zwischen zwei senkrechten Punkten: c, Interpolation in der Mitte vierer Punkte: d).

Funktionsname : Prolongation
Daten : Fehler $v_{2h} \in G_{2h}$ des groben Gitters
Ergebnis : Fehler $v_h \in G_h$ des feinen Gitters

$x = 1$; /* Zeilenindex */
$y = 1$; /* Spaltenindex */
$z = 0$; /* zusätzlicher Spaltenindex */
solange $x < N$ **tue**
  $y = 1$
  $z = 0$
  solange $y < N$ **tue**
    wenn x *eine gerade Zahl ist* **dann**
      wenn y *eine gerade Zahl ist* **dann**
        $v_{h,x\cdot(N+1)+y} = v_{2h,\frac{x}{2}\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}}$; /* Punkt a */
      sonst
        inkrementiere z
        $v_{h,x\cdot(N+1)+y} = \frac{v_{2h,\frac{x}{2}\cdot\left(1+\frac{N}{2}\right)+y-z} + v_{2h,\frac{x}{2}\cdot\left(1+\frac{N}{2}\right)+y-z+1}}{2}$; /* Punkt b */
    sonst
      wenn y *eine gerade Zahl ist* **dann**
        $v_{h,x\cdot(N+1)+y} = \frac{v_{2h,\lfloor\frac{x}{2}\rfloor\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}} + v_{2h,1+\lfloor\frac{x}{2}\rfloor\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}}}{2}$; /* Punkt c */
      sonst
        $v_{h,x\cdot(N+1)+y} = \frac{v_{2h,\lfloor\frac{x}{2}\rfloor\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}} + v_{2h,\lfloor\frac{x}{2}\rfloor\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}+1} + v_{2h,\left(1+\lfloor\frac{x}{2}\rfloor\right)\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}} + v_{2h,\left(1+\lfloor\frac{x}{2}\rfloor\right)\cdot\left(1+\frac{N}{2}\right)+\frac{y}{2}+1}}{4}$; /* Punkt d */
    inkrementiere y; /* rücke zur nächsten Spalte */
  inkrementiere x; /* rücke zur nächsten Zeile */
Ende

Algorithmus 4.2.2 : Bilineare Interpolation

4.3 Zweigitterverfahren

Um dem Konzept des Mehrgitterverfahrens näher zu kommen, bietet sich als ersten Schritt das Zweigitterverfahren an. Die Lösung des diskretisierten Gleichungssystems $Av = r$ soll auf einem gröberen Gitter (G_{2h}) mit doppelter Schrittweite $2h$ berechnet werden. Die Zahl ν_1 ist eine Kennzeichnung für die Anzahl der Iterationen zur Vorglättung und ν_2 für die Anzahl der Iterationen zur Nachglättung. Das Vorgehen des Zweigitter-Algorithmus ist anhand eines $G_{\frac{1}{8}}$-Gitters in der Abbildung 4.3.1 beschrieben. Diese Idee wird anschließend im Algorithmus 4.3.1 für allgemein große Gitter implementiert.

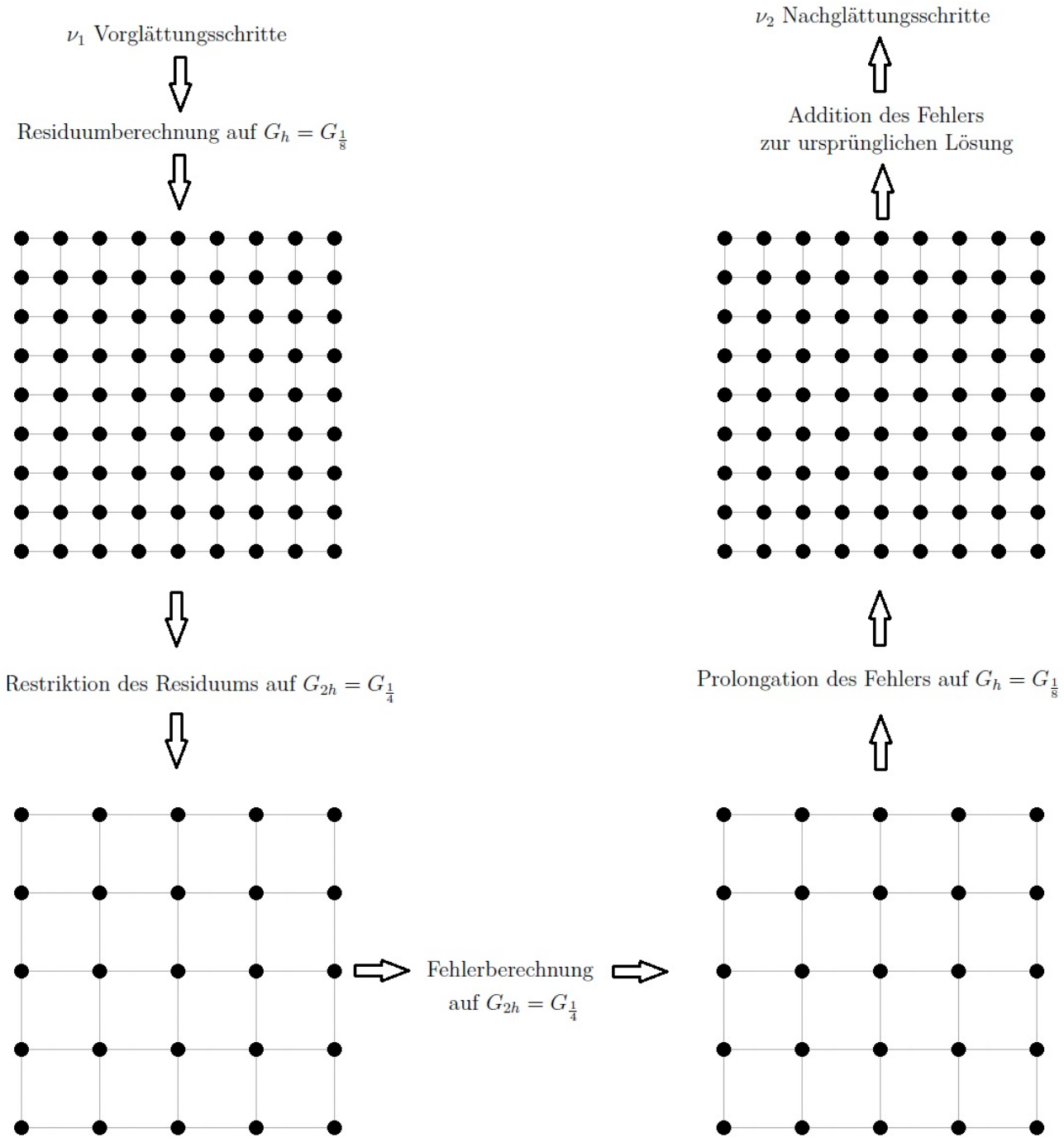

Abbildung 4.3.1: Zweigitterverfahren

Funktionsname : Zweigitterverfahren
Daten : u_h^m, f_h, ν_1, ν_2
Ergebnis : u_h^{m+1}
$i = 0$
solange $i < \nu_1$ **tue**
  u_h^{m,ν_1} =GaußSeidel(u_h^m, f_h); `/* Vorglättung(3.2.1) */`
  inkrementiere i

r_h =Residuum(u_h^{m,ν_1}, f_h); `/* Berechnung des Residuums(4.1.1) auf` G_h `*/`
r_{2h} =Restriktion(r_h); `/* Restriktion des Residuums(4.2.1) auf` G_{2h} `*/`

v_{2h} =Gauß(r_{2h}); `/* Berechne den Fehler (3.1.2) auf` G_{2h}`, dieser */`
; `/* Schritt kann auch mit den Iterationsverfahren */`
; `/* (3.2.1, 3.2.2, 3.2.3) ausgeführt werden. */`

v_h =Prolongation(v_{2h}); `/* Prolongation des Fehlers(4.2.2) auf` G_h `*/`
$u_h^{m+1} = u_h^{m,\nu_1} + v_h$; `/* addiere den Fehler zur ursprünglichen Lösung */`

$i = 0$
solange $i < \nu_2$ **tue**
  u_h^{m+1,ν_2} =GaußSeidel$\left(u_h^{m+1}, f_h\right)$; `/* Nachglättung(3.2.1) */`
  inkrementiere i
Ende

Algorithmus 4.3.1 : Zweigitterverfahren

Wie wir in Abschnitt 5.1 sehen werden, weist dieses Verfahren bezüglich der Anzahl an Iterationen zwar bessere Konvergenzeigenschaften als alle anderen bisher vorgestellten Verfahren auf, dennoch wird die Berechnung bei einem großen Gitter zu viel Zeit und Iterationen in Anspruch nehmen. Die Gauß-Elimination ist trotzdem auf dem halb so großen Gitter G_{2h} auszuführen. Letztlich soll das Ziel, unabhängig von der Anzahl der Stützstellen N, sein, die Berechnung $Av = r$ auf einem immer gleich groben Gitter - zum Beispiel $G_{\frac{1}{16}}$ - auszuführen. Um dieses Ziel zu erreichen, ist es notwendig, mehrmals zu restringieren und anschließend wieder genauso oft zu prolongieren. Dieses Verfahren nennen wir iteratives, geometrisches Mehrgitterverfahren.

4.4 Iteratives Mehrgitterverfahren

Es gibt mehrere Möglichkeiten (Abb. 4.4.1), um zwischen den Gittern zu restringieren und zu prolongieren.

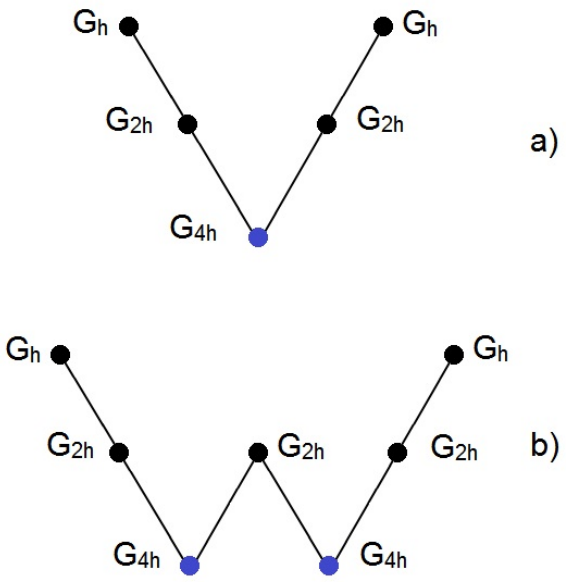

Abbildung 4.4.1: Struktur verschiedener Gitterverfahren: a) V-Zyklus, b)W-Zyklus

In der Literatur [7] findet sich die Bezeichnung V(ν_1,ν_2) für einen V-Zyklus mit ν_1 Vorglättungsiterationen pro Restriktionsschritt, sowie ν_2 Nachglättungsiterationen pro Prolongationsschritt, analog dazu gibt es den W(ν_1,ν_2)-Zyklus. Da die Gleichung $Av = r$ vom selben Typ der Gleichung $Au = f$ ist, kann $Av = r$ rekursiv aufgerufen werden. Mit diesem Vorgehen werden Fehler von Fehlern berechnet, der feinste Fehler wird nach der letzten Prolongation wieder zur ursprünglichen Lösung u_h addiert. Einen Nachteil bietet dieses Verfahren hinsichtlich der Speichernutzung. Sämtliche berechneten Fehler v_{xh}, sowie deren Residuen r_{xh} müssen temporär abgespeichert werden, da sie an späterer Stelle Gebrauch finden. Aufgrund der Tatsache, dass die Gitter pro Restriktion immer gröber werden (somit wird auch der erforderliche Speicher geringer), wird dieser Umstand vorerst außer Acht gelassen. Die Implementierung eines V(ν_1,ν_2)-Zyklus ist im Algorithmus 4.4.1 angeführt. Bei jedem Restriktions/Prolongationsschritt wird geglättet, die absolute Anzahl an Glättungsschritten ist daher von ν_1,ν_2 verschieden und hängt direkt von der Anzahl an Restriktionen/Prolongationen ab.

Funktionsname : Mehrgitterverfahren
Daten : u_h^m, f_h, ν_1, ν_2
Ergebnis : u_h^{m+1}
$i = 0$
solange $i < \nu_1$ **tue**
  u_h^{m,ν_1} =GaußSeidel(u_h^m, f_h); /* Vorglättung (3.2.1) */
  inkrementiere i

r_h =Residuum(u_h^{m,ν_1}, f_h); /* Berechnung des Residuums (4.1.1) auf G_h */
r_{2h} =Restriktion(r_h); /* Restriktion des Residuums (4.2.1) auf G_{2h} */
$x = 4$

solange $\frac{1}{xh} > 16$ **tue**
  $i = 0$
  $v_{xh} = 0$
  solange $i < \nu_1$ **tue**
    $v_{xh}^{\nu_1}$ =GaußSeidel$\left(v_{xh}, r_{2(x-1)h}\right)$; /* Vorglättung (3.2.1) */
    inkrementiere i

  r_{xh} =Residuum$\left(v_{xh}^{\nu_1}, r_{(2(x-1))h}\right)$; /* Residuumberechnung (4.1.1) auf G_{xh} */
  $x_{neu} = 2 \cdot x_{alt}$
  r_{xh} =Restriktion(r_h); /* Restriktion des Residuums(4.2.1) auf $G_{2x_{alt}h}$ */

v_{xh} =Gauß(r_{xh}); /* Berechne den Fehler (3.1.2) auf G_{xh}, dieser */
; /* Schritt kann auch mit den Iterationsverfahren */
; /* (3.2.1, 3.2.2, 3.2.3) ausgeführt werden. */

solange $\frac{1}{2xh} < N$ **tue**
  $v_{xh,Prolongation}$ =Prolongation(v_{2xh}); /* Prolongation des Fehlers */
  ; /* mit dem Algorithmus (4.2.2) auf G_{xh} */
  $v_{xh} = v_{xh} + v_{xh,Prolongation}$; /* addiere die Prolongation */
  ; /* zum ursprünglichen Fehler */

  $i = 0$
  solange $i < \nu_1$ **tue**
    $v_{xh}^{\nu_2}$ =GaußSeidel(v_{xh}, r_{xh}); /* Nachglättung (3.2.1) */
    inkrementiere i
  $x_{neu} = \frac{x_{alt}}{2}$

v_h =Prolongation(v_{2h}); /* Prolongation des Fehlers (4.2.2) auf G_h */
$u_h^{m+1} = u_h^{m,\nu_1} + v_h$; /* addiere den Fehler zur ursprünglichen Lösung */

$i = 0$
solange $i < \nu_1$ **tue**
  u_h^{m+1,ν_2} =GaußSeidel$\left(u_h^{m+1}, f_h\right)$; /* Nachglättung (3.2.1) */
  inkrementiere i
Ende

Algorithmus 4.4.1 : Mehrgitterverfahren

5 Ergebnisse

5.1 Vergleiche

Wir werden direkte Vergleiche der in den vorangegangenen Kapiteln vorgestellten Verfahren anstellen und die Ergebnisse des Mehrgitterverfahrens V(1,1) mit dem Zweigitterverfahren, sowie dem effizientesten Iterationsverfahren (SOR) hinsichtlich der Konvergenzgeschwindigkeiten gegenüberstellen.

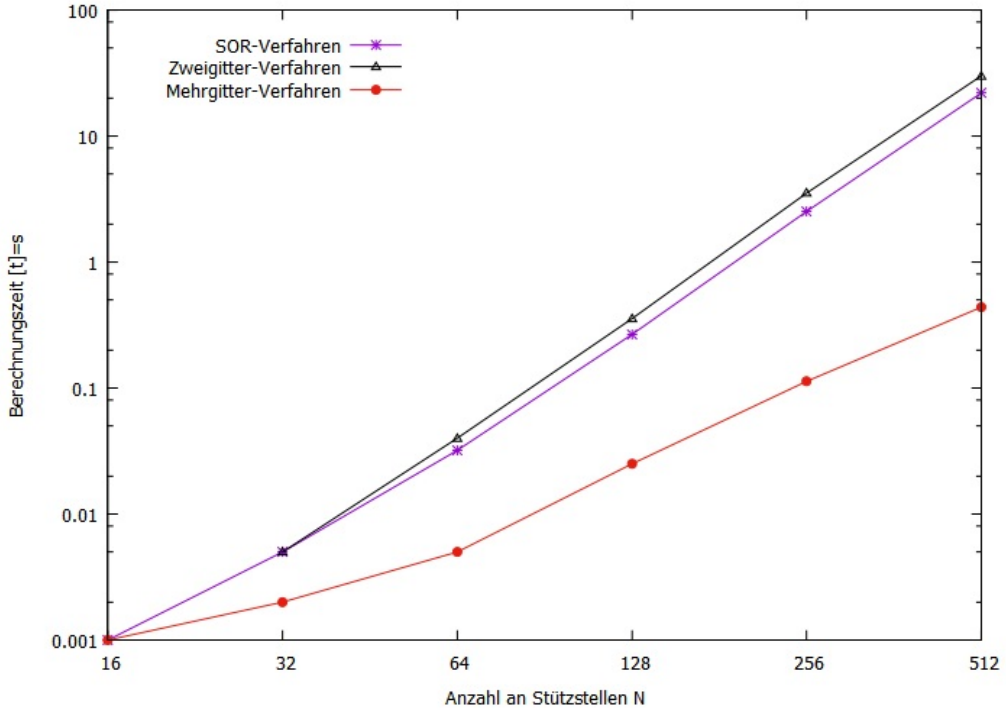

Abbildung 5.1.1: Konvergenzgeschwindigkeit der Zeit/Stützstellenanzahl

Anhand der Abb. 5.1.1 ist ersichtlich, dass sich bei einer geringen Anzahl an Stützstellen $N < 512$ die Restriktion und Prolongation, sowie die Glättungsschritte hinsichtlich der Ausführungszeit beim Zweigitterverfahren negativ auswirken. Insbesondere spielt das Gauß-Seidel Verfahren der Vor/Nachglättung eine Rolle, sowie die Berechnungen zwischen den Gittern. Dennoch ist laut Abb. 5.1.3 zu sehen, dass die Anzahl an Iterationen geringer ist als bei dem SOR-Verfahren.

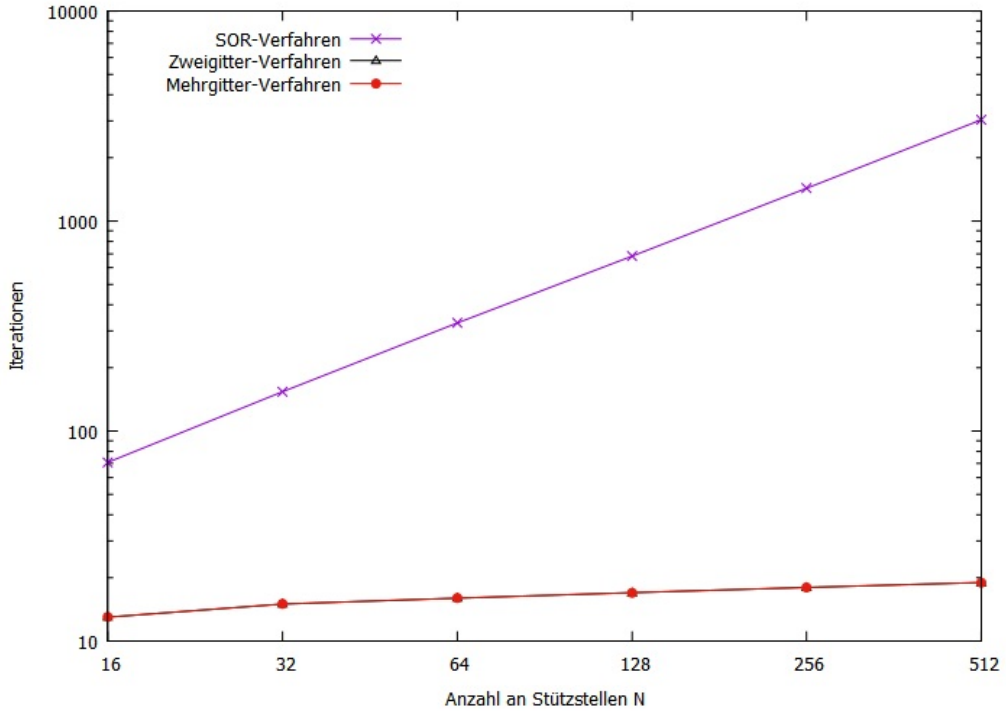

Abbildung 5.1.2: Konvergenzgeschwindigkeit der Iterationen/Stützstellenanzahl

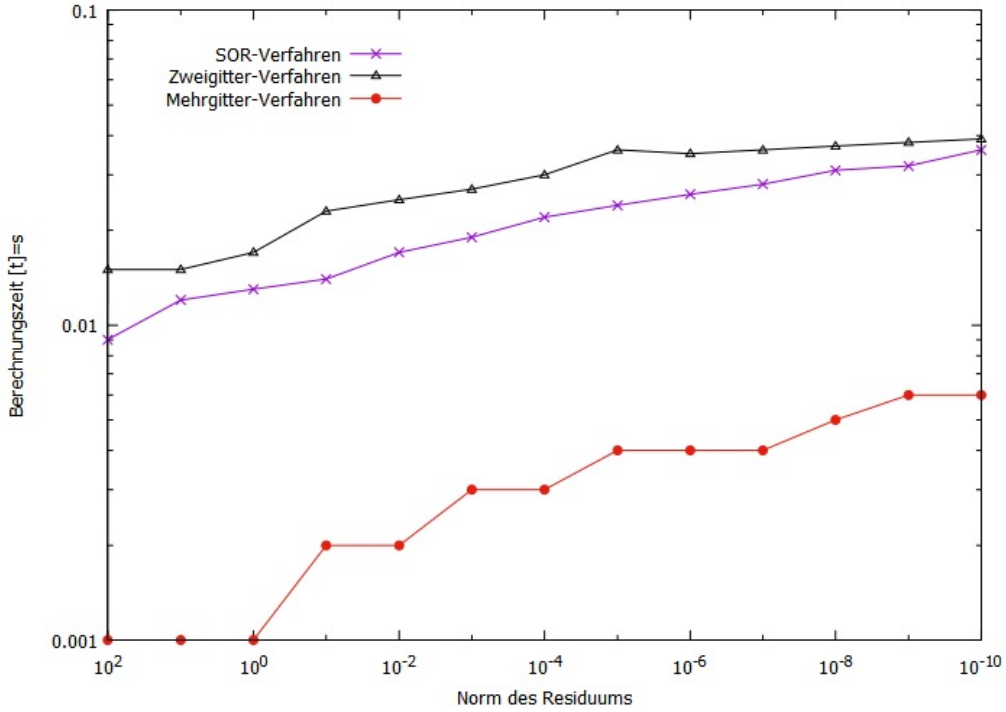

Abbildung 5.1.3: Konvergenzgeschwindigkeit der Zeit/Norm

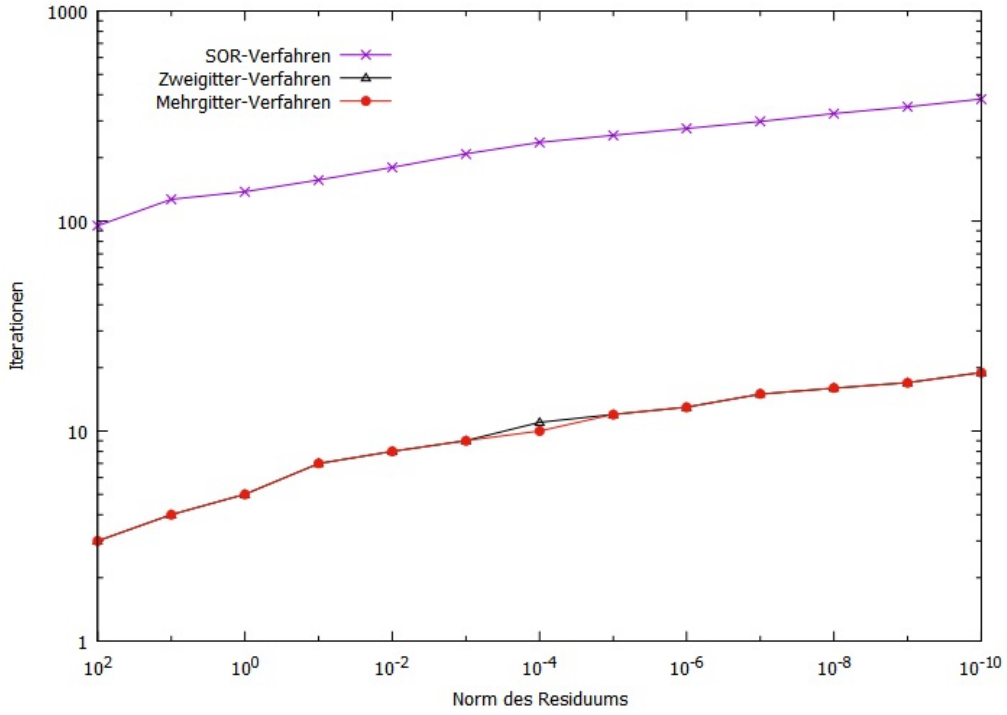

Abbildung 5.1.4: Konvergenzgeschwindigkeit der Iterationen/Norm

Das Mehrgitterverfahren V(1,1) erweist sich als sehr effektiv, es ist gemäß der Abb. 5.1.1 bezüglich der Berechnungszeit bei größeren Gittern wesentlich schneller als das SOR/Zweigitterverfahren.

Anhand der Abb. 3.3.4 lässt sich die Asymptotik einzelner Verfahren erkennen, die Anzahl an Iterationen des SOR-Verfahrens nimmt quadratisch[19] mit der Anzahl an Stützstellen zu, wohingegen die Iterationen des Zweigitter/Mehrgitterverfahrens mit steigenden N vergleichsweise geringfügig steigen. An dieser Stelle ist der Lösungsweg bezüglich der Anzahl an Iterationen lediglich auf Kosten der Berechnungszeit durch hinzufügen weiterer Glättungsschritte zu verbessern.

Wir werden außerdem den Einfluss der Vor/Nachglättung auf das gesamte Mehrgitterverfahren bezüglich der Ausführungszeit in Abhängigkeit der Stützstellen N untersuchen.

[19]Der Plot 3.3.4 ist doppelt logarithmisch dargestellt, die Abszisse mit der Basis 2, die Ordinate mit der Basis 10.

N	V(0,1)	V(1,0)	V(1,1)	V(2,1)	V(2,2)	V(3,3)
16	0.001s	0.001s	0.001s	0.001s	0.001s	0.001s
32	0.003s	0.004s	0.002s	0.001s	0.002s	0.002s
64	0.010s	0.011s	0.007s	0.008s	0.007s	0.008s
128	0.047s	0.046s	0.031s	0.030s	0.028s	0.029s
256	0.137s	0.145s	0.110s	0.089s	0.090s	0.108s
512	0.656s	0.585s	0.405s	0.391s	0.409s	0.434s
1024	2.789s	2.793s	1.937s	1.770s	1.843s	2.029s
2048	11.942s	12.156s	8.361s	8.045s	7.624s	8.273s

Tabelle 5.1.1: Auswertung der verschiedenen Zyklen des Mehrgitterverfahrens $V(\nu 1, \nu 2)$

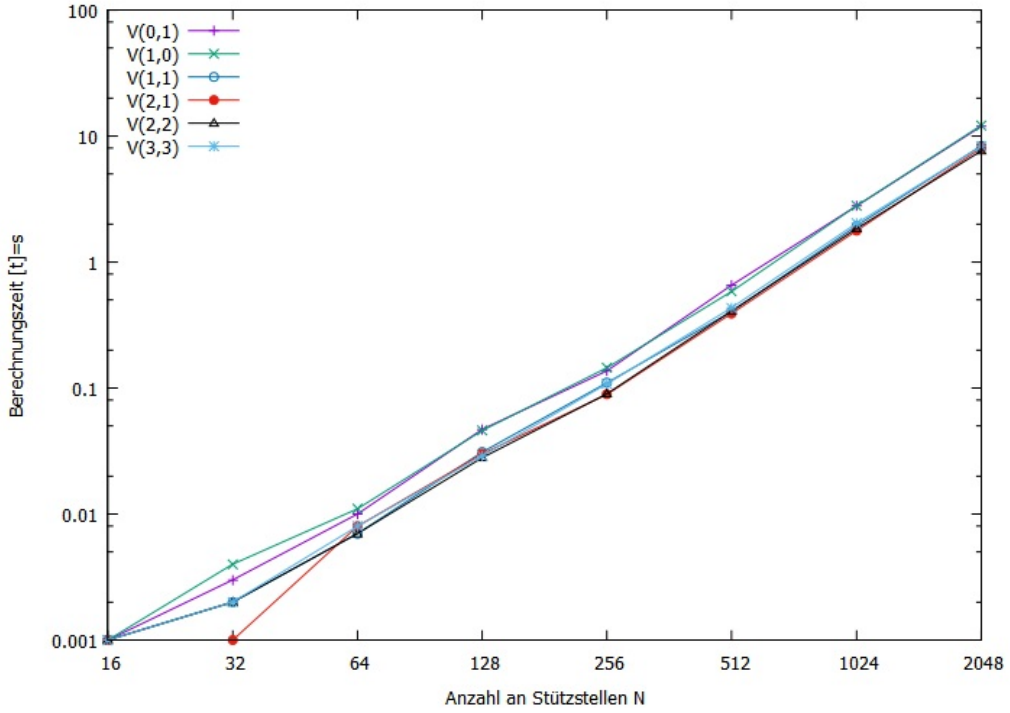

Abbildung 5.1.5: Plot der Tabelle: 5.1.1

Anhand der Tabelle 5.1.1 und dessen Visualisierung 5.1.5 ist zu erkennen, dass die optimale Anzahl an Vor/Nachglättungsschritten beim V(2,2) Zyklus zu finden ist. Eine weitere Erhöhung der Glättungsschritte resultiert in erhöhter Rechenzeit des Gauß-Seidel-Verfahrens, es verringert sich lediglich die Anzahl an Iterationsschritten. Um das

Problem $Au = f$ bei unseren Randbedingungen bei großen N zu lösen, empfiehlt sich bezüglich dieser Betrachtung das V(2,2) Mehrgitterverfahren.

In Abschnitt 4.2.1 haben wir verschiedene Restriktionsoperatoren kennengelernt (Injektion, Halbgewichtung, Vollgewichtung). Welchen Operator wir zu wählen haben, hängt von der Anzahl an Vor/Nachglättungsiterationen ab.

V(0,1)		
Restriktionsoperator	Anzahl an Iterationen	Absolute Berechnungszeit
Injektion	2462	272,08s
Halbgewichtung	50	5,56s
Vollgewichtung	34	4,14s
V(1,1)		
Injektion	21	2,69s
Halbgewichtung	19	2,48s
Vollgewichtung	19	2,99s
V(2,2)		
Injektion	11	1,94s
Halbgewichtung	12	2,26s
Vollgewichtung	12	2,33s

Tabelle 5.1.2: Unterschiede der Restriktionsoperatoren bei verschiedenen Zyklen, mit $N = 1024$ und der Norm$= 10^{-8}$

Anhand der Tabelle 5.1.2 gibt es für jeden der drei Fälle (V(0,1), V(1,1), V(2,2)) geeignete Restriktionsoperatoren[20]. Im V(2,2)-Zyklus wurde das Residuum sehr gut vorgeglättet, weswegen die Injektion am geeignetsten ist. Im V(0,1) Fall finden wir noch Fehler gemäß Abb. 4.1.3 vor. Mit diesen hohen Frequenzen werden bei dieser Art der Restriktion zu viele Punkte verworfen, diesbezüglich liefert die Vollgewichtung die besten Werte. Ist das Residuum zureichend geglättet (V(1,1)), so reicht die Restriktion via Halbgewichtung aus. Zusammenfassend lässt sich sagen, dass wir für unser Problem ($N = 1024$ und der Norm$= 10^{-8}$) am besten den V(2,2)-Zyklus mit einer Restriktion via Injektion zu wählen haben, da hier die wenigsten Iterationen aufgewendet werden, und wir die geringste Berechnungszeit beobachten. Unter Vernachlässigung dieser Untersuchungen ist die Wahl der Vollgewichtung jedoch durchaus zulässig, da hier im Mittel (der Durchschnitt der drei Zyklen) die besten Werte erreicht werden.

[20]Die angegebenen Berechnungszeiten wurden aus dem Mittelwert einiger Daten ermittelt.

5.2 Implementierung

In diesem Abschnitt soll es um die Herangehensweise und Implementierung des Autors zu dieser Arbeit gehen. Die Implementierung wurde in C, mithilfe der Entwicklungsumgebung Code:Blocks unter Windows 10 (64-bit), mit dem GNU GCC Compiler vorgenommen. Sämtliche Werte sind mit einem ASUS X555UB Rechner, mit 8GB Arbeitsspeicher, und einem Intel Core I7-6500U Prozessor aufgenommen worden.

Als Einsteiger in diese Thematik empfiehlt es sich, zunächst die Implementierung der 1D-Poisson Gleichung auszuprobieren. Da die Größe der Matrix $A_{1D-PoissonGleichung}$[21] ein untergeordnetes Problem darstellt (die Probleme der Speichergröße werden anhand des 2D-Falles geschildert, nachzulesen in Abschnitt 2.2), kann man die komplette Tridiagonalmatrix abspeichern. In diesem einfachen Programm kann die Stützstellenanzahl N mit Präprozessoranweisungen angegeben werden, mit dem Nachteil, dass eine Veränderung dieses Parameters nach dem Kompilier-Vorgang nicht mehr möglich ist. Schließlich wurde in der zweiten Version, das Eliminationsverfahren von Gauß, das Gauß-Seidel Verfahren, das SOR Verfahren, das Zweigitterverfahren, sowie das Mehrgitterverfahren in 353 Codezeilen implementiert.

Da die Idee des Mehrgitterverfahrens bereits in einer Dimension umgesetzt ist, und alle Konzepte verstanden sein sollten, kann der nächste Schritt, nämlich die Programmierung des 2D-Falles, vorgenommen werden. Gleich zu Beginn treten Speicherprobleme auf, die laut Abschnitt 3.1 oder ähnlichen Überlegungen zu lösen sind. Außerdem empfiehlt es sich, die Systemmatrix A nicht mehr direkt abzuspeichern, es sind lediglich die Positionen der relevanten Einträge wichtig (beschrieben in Abschnitt 3.2). Des Weiteren wurde die 2D-Lösung um das Jacobi Verfahren erweitert, damit mehr Vergleiche angestellt werden können. Das Programm ist außerdem um eine Menüführung sowie einer automatischen Aufzeichnung der Konvergenzgeschwindigkeiten erweitert. Schlussendlich benötigt diese Implementierung des 2D-Problems in C, in seiner fünften Version (v3.2), inklusive einer Header-Datei 2280 Codezeilen.

Die Konvergenzgeschwindigkeiten sind mit dem Programm gnuplot5.0 patchlevel 1 visualisiert worden. Dazu empfiehlt sich, die Befehle in einer separaten Text-Datei abzuspeichern.

Für die schriftliche Arbeit wurde mithilfe der Umgebung TeXnicCenter und dem Compiler MiKTeX LaTeXverwendet.

6 Schlussfolgerungen/Diskussion

In dieser Arbeit haben wir die Poisson-Gleichung mit der finiten Differenzen Methode diskretisiert und verschiedene Lösungsmöglichkeiten für das daraus resultierende Gleichungssystem diskutiert. Dabei hat sich herausgestellt, dass die Anzahl an Iterationen und die Berechnungszeit mit den in den Abschnitten 3.1 und 3.2 vorgestellten Verfahren für steigende Stützstellenzahlen quadratisch ansteigen. Eine deutliche Verbesserung hinsichtlich der Anzahl an Iterationen wird anschließend durch das Zweigitterverfah-

[21] In diesem Fall ergibt sich anhand der finiten Differenzen Methode eine Tridiagonalmatrix.

ren erreicht. Der Mehrgitteralgorithmus erzielt auch bezüglich der Berechnungszeit sehr gute Werte. Die wichtigsten Zutaten für das Mehrgitterverfahren sind die Glättungseigenschaft und die anschließende Grobgitterkorrektur. Erfüllt ein allgemeines Gleichungssystem, das nichts mit partiellen Differentialgleichungen zu tun haben muss, die Glättungseigenschaft nicht, so treten für die anschließende Grobgitterkorrektur Probleme auf. Die im Kapitel 4.1 beschriebene Glättungseigenschaft ist zu erfüllen, wohingegen das Glättungsverfahren an sich nicht konvergieren muss. Außerdem sei gesagt, dass die Implementierung des Mehrgitterverfahrens, die aufwändig zu programmieren ist, bei kleinen Stützstellen $N \leq 256$ in unserem Fall nicht zweckmäßig scheint, da die anderen vorgestellten Lösungsverfahren auch in einer angemessenen Zeit konvergieren.

Die Frage, die sich erschließt, ist, in welchen Fällen eine hohe Auflösung überhaupt erforderlich sei. In dieser Arbeit ist stets von einem quadratischen Randgebiet ausgegangen worden, jedoch stößt man bei „komplizierteren“ Geometrien laut [1] schnell an die Grenzen der finiten Differenzen Methode. Um nicht auf die finite Elemente Methode umsteigen zu müssen, kann man das Gitter feiner auflösen und die „komplizierte“ Geometrie durch kleine rechteckige Gebiete annähern. Dieses Problem ist in Abb. 6.1 durch Annäherung eines unförmigen Gebietes durch rechteckige Gebiete verdeutlicht. Beim feineren Gitter (6.1-rechtes Bild) werden weniger Fehler gemacht. Die Matrix A bekommt in diesem Fall eine andere Besetzung als bei unserem Rechteckgebiet. Sie bleibt zwar eine Matrix mit maximal fünf Einträgen pro Zeile, jedoch sind die 1-Elemente (die Randpunkte) anders verteilt. Unter diesem Aspekt, dass man die Stützstellenanzahl erhöhen muss, stößt man bei herkömmlichen, direkten oder indirekten Verfahren schnell an Grenzen, das Mehrgitterverfahren ist eine sinnvolle Alternative.

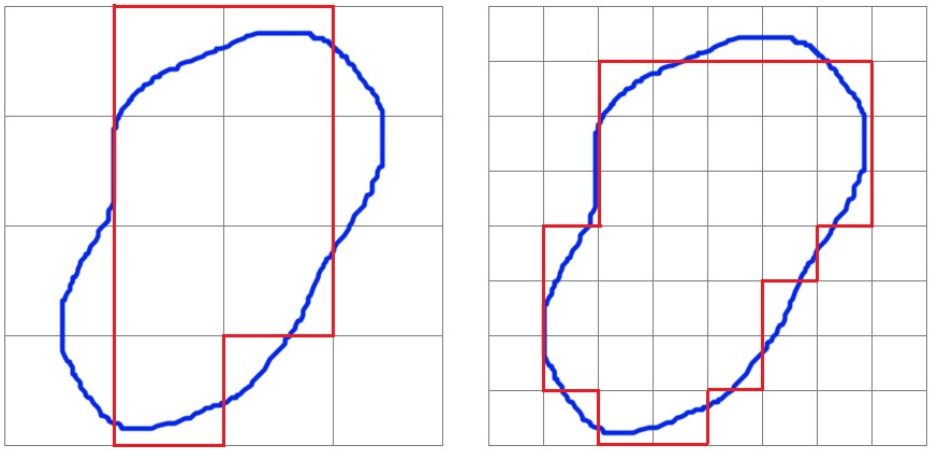

Abbildung 6.1: Komplexe Geometrien angenähert durch rechteckige Gebiete

Literatur

[1] Ceric, H. ; Langer, E.: *Fachvertiefungsunterlagen zu 'Numerische Aspekte der Mikrostruktursimulation'.* TU Wien, Institut für Mikroelektronik, WS2014/15

[2] Köckler, N.: *Mehrgittermethoden (Ein Lehr- und Übungsbuch).* Springer Spektrum, 2012

[3] Klima, R. ; Selberherr, S.: *Programmieren in C.* Springer Wien New York, 3. Auflage

[4] Knabner, P. ; Angermann, L.: *Numerik partieller Differentialgleichungen.* Springer-Verlag, 2005

[5] Prechtl, A.: *Vorlesungen über Elektrodynamik.* TU Wien, Institute of Electrodynamics, Mircowave and Circuit Engineering, 2010

[6] Schaback, R. ; Wendland, H.: *Numerische Mathematik.* Springer-Verlag, 2000

[7] Trottenberg, U. ; Oosterlee, C. ; Schüller, A.: *Multigrid.* Academic Press, 2001