Fakultät für Mathematik
Universität Regensburg

Ein Vergleich des Verfahrens der konjugierten Gradienten und Mehrgittermethoden, angewandt auf die diskretisierte Poisson-Gleichung

Bachelor-Arbeit

Michael Bauer
Matrikel-Nummer 1528558

Erstprüfer Prof. Garcke
Zweitprüfer Prof. Blank

Inhaltsverzeichnis

1 Einleitung

Viele Prozesse in den Naturwissenschaften, wie Biologie, Chemie und Physik, aber auch der Medizin, Technik und Wirtschaft lassen sich auf partielle Differentialgleichungen (PDG) zurückführen. Das Lösen solcher Gleichungen ist allerdings nicht immer möglich, oder aufwendig.
Eine PDG, die vor Allem in der Physik häufige Verwendung findet, ist die Poisson-Gleichung – eine elliptische partielle Differentialgleichung zweiter Ordnung. So genügt diese Gleichung beispielsweise dem elektrostatischen Potential u zu gegebener Ladungsdichte f, aber auch dem Gravitationspotential u zu gegebener Massendichte f.
Methoden aus der numerischen Mathematik ermöglichen uns nun das Lösen von partiellen Differentialgleichungen mittels computerbasierten Algorithmen. Hierbei wird jedoch nicht die Lösung direkt bestimmt, sondern versucht eine exakte Approximation der Lösung zu erhalten. Dabei ist es wichtig, dass der zugrunde liegende Algorithmus effizient ist.
Um nun die Lösung einer partiellen Differentialgleichung mittels effizienten Algorithmus bestimmen zu können, müssen wir uns im Vorfeld Gedanken darüber machen, wie wir diese am besten erhalten. Eine der zentralen Methoden der Numerik sind Finite Differenzen. Hierbei diskretisiert man das Gebiet, auf dem die PDG definiert ist und führt die Gleichung auf ein lineares Gleichungssystem zurück.
Eine Möglichkeit ein solches lineares Gleichungssystem zu lösen sind iterative Verfahren. Somit hat man ein großartiges Werkzeug, dass eine Lösung innerhalb weniger Iterationsschritte berechnet und auch mit sehr großen Systemen gut zurecht kommt. Natürlich gilt das nicht für jedes Verfahren, weshalb wir über die Verfahren sprechen möchten, die diese Kriterien erfüllen.
Abschließend wollen wir uns dann noch mit Mehrgittermethoden beschäftigen. Sie stellen die wohl effizienteste und modernste Methode dar, ein lineares Gleichungssystem zu lösen.

2 Diskretisierung der Poisson-Gleichung im $\mathbb{R}^2$

2.1 Definition (Poisson-Gleichung)

Sei $\Omega = (0,1) \times (0,1) \in \mathbb{R}^2$ ein beschränktes, offenes Gebiet. Gesucht wird eine Funktion $u(x,y)$, die das Randwertproblem

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) \text{ in } \Omega & (2.1)\\ u(x,y) &= g(x,y) \text{ in } \partial\Omega & (2.2) \end{aligned}$$

löst. Dabei seien $f : \Omega \to \mathbb{R}$ und $g : \partial\Omega \to \mathbb{R}$ stetige Funktionen und es bezeichnet $\Delta := \sum_{k=1}^{n} \frac{\partial^2}{\partial x_k^2}$ den Laplace-Operator. Für die Poisson-Gleichung im $\mathbb{R}^2$ gilt dann:

$$\begin{aligned} -\Delta u(x,y) &= \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y) \text{ in } \Omega & (2.3)\\ u(x,y) &= g(x,y) \text{ in } \partial\Omega & (2.4) \end{aligned}$$

(2.2) und (2.4) nennt man Dirichlet-Randbedingung.
Beachte: Es ist äquivalent: $\partial_{xx} u(x,y) = \frac{\partial^2 u(x,y)}{\partial x^1}$

Um diese (elliptische) partielle Differentialgleichung nun in Ω zu diskretisieren, bedarf es der Hilfe der Finiten Differenzen Methode.

2.2 Finite Differenzen-Methode für die Poisson-Gleichung

2.2.1 Zentraler Differenzenquotient zweiter Ordnung

Wir betrachten ein $(x,y) \in \Omega$ beliebig. Dann gilt für $h > 0$ mit der Taylorformel

$$u(x+h,y) = \sum_{k=0}^{n} h^k \frac{\partial^k u(x,y)}{\partial x^k} \approx u(x,y) + h\partial_x u(x,y) + \frac{h^2}{2!}\partial_{xx} u(x,y) + \mathcal{O}(h^3) \qquad (2.5)$$

$$u(x-h,y) = \sum_{k=0}^{n} (-1)^k h^k \frac{\partial^k u(x,y)}{\partial y^k} \approx u(x,y) - h\partial_x u(x,y) + \frac{h^2}{2!}\partial_{xx} u(x,y) - \mathcal{O}(h^3) \quad (2.6)$$

Analog können wir diese Betrachtung für $u(x, y+h)$ und $u(x, y-h)$ machen.
Löst man nun (2.5) und (2.6) jeweils nach $\partial_{xx} u(x,y)$ auf und addiert die zwei Gleichungen, so erhält man:

$$\partial_{xx} u(x,y) + O(h^2) = \frac{u(x-h,y) - 2u(x,y) + u(x+h,y)}{h^2} \quad (2.7)$$

Ebenso lösen wir nach $\partial_{yy} u(x,y)$ auf und erhalten:

$$\partial_{yy} u(x,y) + O(h^2) = \frac{u(x,y-h) - 2u(x,y) + u(x,y+h)}{h^2} \quad (2.8)$$

Wobei hier $\partial_{xx} u(x,y) = \frac{\partial^2 u(x,y)}{\partial x^2}$ und $\partial_{yy} u(x,y) = \frac{\partial^2 u(x,y)}{\partial y^2}$ gemeint ist. Diese Näherungen nennt man auch (zentralen) Differenzenquotienten der zweiten Ableitung. $O(h^2)$ ist ein Term zweiter Ordnung und wird vernachlässigt.

Somit erhalten wir für $\Delta u(x,y)$ die Näherung

$$\begin{aligned} \Delta u(x,y) &= \partial_{xx} u(x,y) + \partial_{yy} u(x,y) \\ &\approx \frac{u(x-h,y) + u(x+h,y) - 4u(x,y) + u(x,y-h) + u(x,y+h)}{h^2} \end{aligned} \quad (2.9)$$

2.2.2 Diskretisierung von Ω

Mit einem zweidimensionalen Gitter, der Gitterweite h, wobei $h \in \mathbb{Q}$ mit $h = \frac{1}{m}$ und $m \in \mathbb{N}_{>1}$, wird nun das Gebiet Ω diskretisiert. Die Zahl $N := (m-1)$ gibt uns an, wie viele Gitterpunkte es jeweils in x- bzw. y-Richtung gibt.

Für $i, j = 1, ..., N$ kann man dann $u(x,y)$ auch schreiben als:

$$u(x_i, y_j) = u(ih, jh) \quad (2.10)$$

und Ω fassen wir als Ω_h auf, so dass:

$$\Omega_h := \{u(ih, jh) | 1 \leq i, j \leq N\} \quad (2.11)$$

Betrachten wir nun noch den Abschluss von Ω_h:

$$\overline{\Omega}_h := \{u(ih, jh) | 0 \leq i, j \leq m\} \quad (2.12)$$

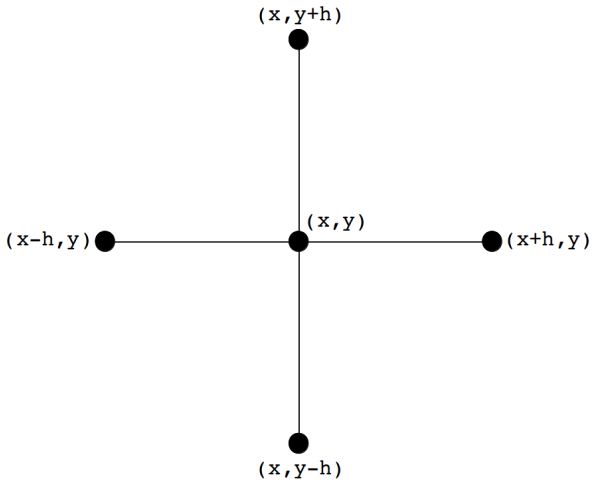

Abbildung 2.1: 5-Punkt-Differenzenstern im Gitter

Mit der Formel (2.9) ergibt sich nun für $\Delta u(x,y) \approx \Delta_h u(x,y)$ die diskretisierte Form:

$$\Delta_h u(x,y) = \frac{u(x-h,y) + u(x+h,y) - 4u(x,y) + u(x,y-h) + u(x,y+h)}{h^2} \tag{2.13}$$

für alle $(x,y) \in \Omega_h$.
Man stellt $\Delta_h u(x,y)$ häufig auch als 5-Punkt-Differenzenstern (Abbildung 2.2.2) in der Form

$$[-\Delta_h]_\xi = \frac{1}{h^2} \begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}, \xi \in \Omega_h \tag{2.14}$$

dar. (Dahmen Reusken)

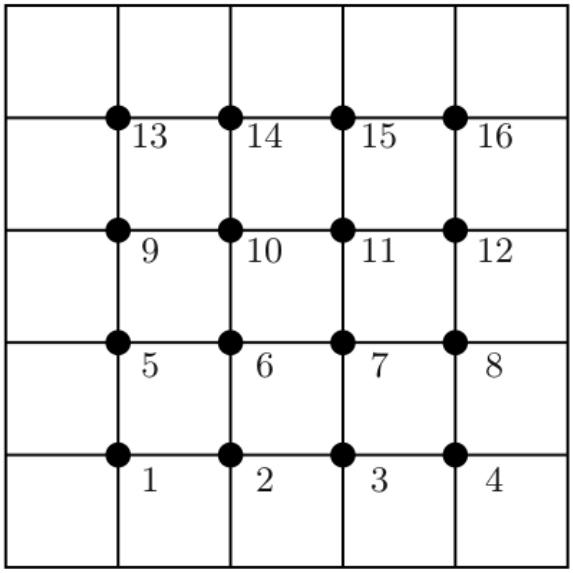

Abbildung 2.2: (Lexikographische) Nummerierung von $\Omega = (0,1)^2$ mit $n = 5$

Nummeriert man nun alle Gitterpunkte des Gitters fortlaufend von links unten nach rechts oben durch (Abbildung 2.2.2) und stellt für jeden dieser Punkte Gleichung 2.13 auf, so führt dies auf eine $N^2 \times N^2$-Matrix der Form:

$$\mathbf{A} = \begin{pmatrix} A_1 & -Id & & \\ -Id & A_2 & \ddots & \\ & \ddots & \ddots & -Id \\ & & -Id & A_n \end{pmatrix} \tag{2.15}$$

wobei $\mathbf{Id} \in \mathbb{R}^{N \times N}$ die Identität meint und für alle $i = 0, .., n$ gilt:

$$A_i = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \tag{2.16}$$

$A_i \in \mathbb{R}^{N \times N}$. Man kann zeigen (???), dass A symmetrisch positiv definit (s.p.d) ist.

Um nun auf ein lineares Gleichungssystem der Form $\mathbf{A}u = f$ zu kommen, muss natürlich noch die rechte Seite, also das f aufgestellt werden. Zu jeder Komponente von f, die einen Randpunkt als Nachbarn hat, wird dieser dazu addiert. Dies führt uns auf folgende rechte Seite:

$$f = h^2 \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix} \tag{2.17}$$

wobei gilt

$$f_1 = \begin{pmatrix} f(h,h) + h^{-2}(g(h,0) + g(0,h)) \\ f(2h,h) + h^{-2}(g(2h,0)) \\ \vdots \\ f(1-2h,h) + h^{-2}(g(1-2h,0)) \\ f(1-h,h) + h^{-2}(g(1-h,0) + g(0,1-h)) \end{pmatrix} \tag{2.18}$$

$$f_j = \begin{pmatrix} f(h,jh) + h^{-2}(g(0,jh)) \\ f(2h,jh) \\ \vdots \\ f(1-2h,jh) \\ f(1-h,jh) + h^{-2}(g(1,jh)) \end{pmatrix} 2 \leq j \leq N-1, \tag{2.19}$$

$$f_N = \begin{pmatrix} f(h,1-h) + h^{-2}(g(h,1) + g(0,1-h)) \\ f(2h,1-h) + h^{-2}(g(2h,1)) \\ \vdots \\ f(1-2h,1-h) + h^{-2}(g(1-2h,1)) \\ f(1-h,1-h) + h^{-2}(g(1-h,1) + g(1,1-h)) \end{pmatrix} \tag{2.20}$$

Wir erhalten das lineare Gleichungssystem der Form $\mathbf{A}u = f$, wobei $\mathbf{A}$ und f bekannt sind und u der approximierte Lösungsvektor ist, der die Lösung der partielle Differentialgleichung enthält. Wir bezeichnet ab jetzt die diskrete 2D Poisson Matrix mit $\mathbf{A}_{2D}$

2.3 Eigenschaften der Matrix $\mathbf{A}_{2D}$

Da $\mathbf{A}_{2D} \in \mathbb{R}^{n \times n}$ s.p.d. ist, existiert eine Orthogonalbasis aus Eigenvektoren für die gilt:

$$\mathbf{A}_{2D} v = \lambda v, \tag{2.21}$$

wobei $\lambda_1, ..., \lambda_n \in \mathbb{R}$ und $v \in \mathbb{R}^n$.

2.3.1 Eigenvektoren von $\mathbf{A}_{2D}$

Sei $x_k := k \cdot h$, wobei $k \in \mathbb{N}$ und $x_k \in \Omega_h$. Dann gilt für die Eigenvektoren der Matrix $\mathbf{A}_{2D}$:

$$v_{k,l}(x_i, y_j) = \sin(k\pi x_i)\sin(l\pi y_j) \tag{2.22}$$

2.3.2 Eigenwerte von $\mathbf{A}_{2D}$

Sei $\theta_k := k\pi h \in \mathbb{R}$. Dann gilt für die Eigenwerte von $\mathbf{A}_{2D}$:

$$\lambda_{ij} = 4\left(\sin^2(\frac{\theta_i}{2}) + \sin^2(\frac{\theta_j}{2})\right) \text{ für } 1 \leq i, j \leq N \tag{2.23}$$

wobei N die Anzahl der Gitterpunkte in x- und in y-Richtung definiert.

2.3.3 Definition (Kondition einer symmetrischen Matrix)

Sei $\mathbf{A}$ eine symmetrische Matrix des $\mathbb{R}^{n\times n}$. Dann gilt für die Kondition der Matrix:

$$\kappa_2(A) := \frac{\lambda_{max}}{\lambda_{min}} \tag{2.24}$$

2.3.4 Lemma (Kondition von $\mathbf{A}_{2D}$)

Für die Matrix $\mathbf{A}_{2D}$ gilt für ihre Kondition:

$$\kappa_2(A_{2D}) = \frac{\cos^2(\frac{\pi h}{2})}{\sin^2(\frac{\pi h}{2})} = \left(\frac{2}{\pi h}\right)^2 (1 + \mathcal{O}(h^2)) \tag{2.25}$$

Beweis:

Da $\mathbf{A}_{2D}$ s.p.d. ist, gilt mit Gleichung 2.24: $\kappa_2(A_{2D}) = \frac{\lambda_{max}}{\lambda_{min}}$. Für die Eigenwerte gilt aus Unterabschnitt 2.3.2:

$$\lambda_{ij} = 4\left(\sin^2(\frac{\theta_i}{2}) + \sin^2(\frac{\theta_j}{2})\right) \text{ für } 1 \leq i, j \leq N$$

Dann folgt:

$$\begin{aligned}\kappa_2(A_{2D}) &= \frac{\lambda_{N,N}}{\lambda_{1,1}} = \frac{4(2\sin^2(\frac{N\pi h}{2}))}{4(2\sin^2(\frac{\pi h}{2}))} \overset{h=\frac{1}{m},\ N=m-1}{=} \frac{\sin^2(\frac{(m-1)\pi}{2m})}{\sin^2(\frac{\pi h}{2})} \\ &= \frac{\sin^2(\frac{\pi}{2}-\frac{\pi h}{2})}{\sin^2(\frac{\pi h}{2})} \overset{Additionstheoreme}{=} \frac{\cos^2(\frac{\pi h}{2})}{\sin^2(\frac{\pi h}{2})}\end{aligned}$$

In [DahmenReusken] wird außerdem gezeigt, dass sich $\kappa(\mathbf{A}_{2D})$ wie folgt nähern lässt:

$$\frac{\cos^2(\frac{\pi h}{2})}{\sin^2(\frac{\pi h}{2})} = \left(\frac{2}{\pi h}\right)^2 (1 + \mathcal{O}(h^2)) \tag{2.26}$$

Natürlich wollen wir die Funktion $u(x, y)$ so gut wie möglich in Ω_h approximieren und sind daher bestrebt das Gitter so fein als möglich zu wählen. Daraus ergibt sich jedoch die negative Eigenschaft von $\mathbf{A}_{2D}$
Zur Erinnerung: $h := \frac{1}{m}$. Je größer m gewählt wird, desto schlechter wird die Kondition der Matrix.

$$\left(\frac{2}{\pi h}\right)^2 (1 + \mathcal{O}(h^2)) = \left(\frac{2}{\pi \frac{1}{m}}\right)^2 (1 + \mathcal{O}(\frac{1}{m^2})) \approx 8m^2 \tag{2.27}$$

Für ein großes m, also ein feines Gitter, wird die Kondition sehr groß.

3 Iterative Lösungsvefahren für lineare Gleichungssysteme

Gleichungssysteme, die partielle Differentialgleichungen lösen, können sehr groß werden. Aus diesem Grund sind direkte Verfahren, wie z.B. der Gauß-Algorithmus oder die LR-Zerlegung nicht geeignet. Ihr Rechenaufwand beläuft sich im Allgemeinen auf $\mathcal{O}(n^3)$ und ist zu langsam.
Wesentlich besser geeignet für diese Problemstellung sind iterative Verfahren. Sie zeichnen sich durch eine schnelle Konvergenz und einen geringeren Rechenaufwand aus - falls sie konvergieren.

3.1 Grundbegriffe

3.1.1 Definition (Spektralradius)

Sei $\mathbf{A} \in \mathbb{R}^{n\times n}$. Und seien für alle $i = 1, ..., n$ $\lambda_i \in \mathbb{R}$. Dann gilt:

$$\rho(A) := \max_{1\leq i\leq n} |\lambda_i| \tag{3.1}$$

Ist $\mathbf{A}$ symmetrisch so gilt auch $\rho(A) = \|A\|_2$ und $\lambda_i \in \mathbb{R}_{>0}$ für alle $i = 1, ..., n$.

3.1.2 Definition (Iterationsmatrix)

Sei $\mathbf{A} \in \mathbb{R}^{n\times n}$. Sei außerdem $\mathbf{C} \in \mathbb{R}^{n\times n}$ eine nichtsinguläre Matrix. Für die iterative Lösung eines linearen Gleichungssystems der Form $Au = f$ ist die Iterationsmatrix $\mathbf{T} \in \mathbb{R}^{n\times n}$ definiert als:

$$\mathbf{T} := (\mathbf{Id} - \mathbf{CA}) \tag{3.2}$$

wobei die Iterationsvorschrift für $k = 1, ..., n$ gegeben ist durch:

$$u^{k+1} := (\mathbf{Id} - \mathbf{CA})u^k + \mathbf{C}f \tag{3.3}$$

3.1.3 Satz (Konvergenz iterativer Verfahren)

Ein iteratives Verfahren mit beliebigen Startvektor $x^0 \in \mathbb{R}^n$ konvergiert genau dann gegen die exakte Lösung $x^* \in \mathbb{R}^n$, wenn gilt:

$$\rho(\mathbf{T}) = \rho(Id - CA) < 1 \tag{3.4}$$

Einen Beweis hierzu findet man z.B. in (Dahmen/Reusken und Verweis).

3.1.4 Definition (Residuum und Fehler)

Sei $u^* \in \mathbb{R}^n$ die exakte Lösung des linearen Gleichungssystems $Au = f$. Sei außerdem für $u^k \in \mathbb{R}^n$ die Approximation der Lösung im $k - ten$ Iterationsschritt. Dann gilt für das Residuum:

$$r^k := f - \mathbf{A}u^k \tag{3.5}$$

Der Fehler, also die Diskrepanz zwischen exakter und approximierter Lösung, ist definiert als:

$$e^k := u^* - u^k \tag{3.6}$$

Durch Multiplikation mit der Matrix $\mathbf{A}$ ergibt sich:

$$e = u^* - u \Leftrightarrow \mathbf{A}e = \mathbf{A}(u^* - u) \Leftrightarrow \mathbf{A}e = \mathbf{A}u^* - \mathbf{A}u \Leftrightarrow \mathbf{A}e = b - \mathbf{A}u \Leftrightarrow \mathbf{A}e = r \tag{3.7}$$

$\mathbf{A}e = r$ nennen wir Residuumsgleichung.

3.2 Das Jacobi-Verfahren (Gesamtschrittverfahren)

Für die Beispiele gilt stets: $n := N \cdot N$, wobei N die Gitterweite in x- und y-Richtung darstellt.

3.2.1 Das allgemeine Jacobi-Iterationsverfahren

Sei $\mathbf{A} \in \mathbb{R}^{n \times n}$ und $f, u \in \mathbb{R}^n$, wobei u die Lösung des linearen Gleichungssystems $\mathbf{A}u = f$ ist. Dann lässt sich $\mathbf{A}$ wie folgt zerlegen:

$$A = D - L - U \tag{3.8}$$

Dabei sind $\mathbf{D}, \mathbf{L}, \mathbf{U} \in \mathbb{R}^{n \times n}$, wobei $\mathbf{D}$ die Diagonalelemente von $\mathbf{A}$ enthält, $\mathbf{L}$ eine strikte untere und $\mathbf{U}$ eine strikte obere Dreiecksmatrix sind.
Somit ergibt sich für $\mathbf{A}u = f$:

$$Au = f \Leftrightarrow (D - L - U)u = f \Leftrightarrow Du = (L + U)u + f \tag{3.9}$$

Ist nun **D** nicht singulär, so gilt für das Jacobi-Verfahren folgende Iterationsvorschrift:

$$Du^{k+1} = (L + U)u^k + f \Leftrightarrow u^{k+1} = D^{-1}(L + U)u^k + D^{-1}f \tag{3.10}$$

In Komponentenschreibweise:
Mit einem Startvektor $u^0 \in \mathbb{R}^n$ *beliebig* und $k = 1, 2, ...$ berechne für $i = 1, ..., n$:

$$u_i^{k+1} = \frac{1}{a_{ii}}(f_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}u_i^k) \tag{3.11}$$

In jedem Schritt zur Berechnung von u^{k+1} muss hier die Information seines Vorgängers u^k bekannt sein. Der Rechenaufwand pro Iterationsschritt beträgt $\mathcal{O}(n^2)$ und entspricht somit einer Matrix-Vektor-Multiplikation.

3.2.2 Satz (Iterationsmatrix des Jacobi-Verfahrens)

Für die Iterationsmatrix des Jacobi-Verfahrens gilt:

$$\mathbf{T}_J := (\mathbf{Id} - \mathbf{D}^{-1}\mathbf{A}) \tag{3.12}$$

Hier ist also $\mathbf{C} = \mathbf{D}^{-1}$

Beweis:

Mit der Iterationsvorschrift folgt:

$$\begin{aligned} u^{k+1} &= D^{-1}(L + U)u^k + D^{-1}f \overset{(L+U)=(D-A)}{=} D^{-1}(D - A)u^k + D^{-1}f \\ &= (Id - D^{-1}A)u^k + D^{-1}f \end{aligned}$$

$\Longrightarrow$ Beh.

3.2.3 Satz (Eigenwerte der Jacobi-Iterationsmatrix bzgl. $\mathbf{A}_{2D}$)

Man sieht leicht ein, dass die Eigenvektoren von $\mathbf{T}_J$ gleich denen von $\mathbf{A}_{2D}$ sind. Dann gilt für die Eigenwerte der Iterationsmatrix $\mathbf{T}_J := (\mathbf{Id} - \mathbf{D}^{-1}\mathbf{A}_{2D})$:

$$\lambda_{i,j}(\mathbf{T}_J) = 1 - \left(\sin^2(\frac{\theta_i}{2}) + \sin^2(\frac{\theta_j}{2})\right) \tag{3.13}$$

für $1 \leq i, j \leq N$ und θ_i, θ_j wie in Unterabschnitt 2.3.2.

Beweis:

Für $\mathbf{D} = 4\mathbf{Id}$ folgt $\mathbf{D}^{-1} = \frac{1}{4}\mathbf{Id}$. Für die Iterationsmatrix $\mathbf{T}_J$ angewandt auf einen Vektor u gilt:

$$Tu = (Id - D^{-1}A)u = Idu - D^{-1} \underbrace{Au}_{=\lambda(A)u} = Idu - \frac{1}{4}IduA = u(1 - \frac{1}{4}\lambda(A))$$

Die Eigenwerte von $\mathbf{T}$ lassen sich also einfach durch $(1 - \frac{h^2}{4}\lambda(A))$ berechen:

$$\lambda_{i,j}(\mathbf{T}_J) = 1 - \left(\sin^2(\frac{\theta_i}{2}) + \sin^2(\frac{\theta_j}{2})\right) \text{ für } 1 \leq i, j \leq N$$

3.2.4 Lemma (Spektralradius der Jacobi-Iterationsmatrix bzgl. $\mathbf{A}_{2D}$)

Das Jacobi-Verfahren konvergiert für die diskrete Poisson Gleichung und es gilt für den Spektralradius:

$$\rho(\mathbf{T}_J) = \cos(\pi h) < 1 \tag{3.14}$$

Beweis:

Mit Gleichung 3.22 folgt:

$$\begin{aligned}
\rho(Id - D^{-1}A_{2D}) &= \max_{1 \leq i \leq n} |\lambda_i| = \max_{1 \leq i \leq n} |1 - \left(\sin^2(\frac{\theta_i}{2}) + \sin^2(\frac{\theta}{2})\right)| \\
&= 1 - 2\sin^2(\frac{\pi h}{2}) = 1 - 2(\frac{1}{2}(1 - \cos(\pi h)) \\
&= \cos(\pi h) \overset{\text{Taylorformel}}{\approx} 1 - \frac{1}{2}\pi^2 h^2
\end{aligned}$$

Das Jacobi-Verfahren konvergiert also für $\mathbf{A}_{2D}$. Allerdings nimmt mit feinerem Gitter, also kleinere Schrittweite h, die Konvergenzgeschwindigkeit stark ab, da der Spektralradius nahe bei 1 liegt.

3.2.5 Das Jacobi-Iterationsverfahren für die Poisson-Gleichung

Um den Rechenaufwand zu verbessern, nutzen wir nun noch die Dünnbesetztheit von $\mathbf{A}_{2D}$ aus. Es sind pro Zeile maximal 5 Einträge ungleich Null.

Berechne für $k = 1, 2, ...$ mit Startvektor $u^0 \in \mathbb{R}^n$ *beliebig*
Für $i = 1, ..., N$ und für $j = 1, ..., N$:

$$u_{i,j}^{k+1} = \frac{1}{a_{i,i}}(f_{i,j} - u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k) \tag{3.15}$$

(Werte, bei denen eine Null im Index steht, werden ignoriert!)
Der Rechenaufwand pro Iterationsschritt beträgt lediglich $\mathcal{O}(N \cdot N) = \mathcal{O}(n)$ Schritte.

3.3 Das Jacobi-Relaxationsverfahren

Wir wollen nochmal das Jacobi-Verfahren betrachten:

$$u^{k+1} = (Id - D^{-1}A)u^k + D^{-1}f \tag{3.16}$$

Wir wollen zunächst eine Umformung vornehmen:

$$\begin{aligned} u^{k+1} &= (Id - D^{-1}A)u^k + D^{-1}f \\ &= u^k - D^{-1}Au^k + D^{-1}f \\ &= u^k + D^{-1}\underbrace{(f - Au^k)}_{=r^k} \end{aligned} \tag{3.17}$$

Wir addieren also zu u^k das Residuum. Die Idee ist nun dieses mit einem Parameter $\omega \in \mathbb{R}$ zu multiplizieren, um bessere Konvergenzeigenschaften zu erhalten:

$$\begin{aligned} u^{k+1} &= u^k + D^{-1}\omega r^k = u^k + \omega D^{-1}(f - Au^k) \\ &= u^k - \omega D^{-1}Au^k + \omega D^{-1}f = (Id - \omega D^{-1}A)u^k + \omega D^{-1}f \\ &= (Id - \omega Id + \omega Id - \omega D^{-1}A)u^k + \omega D^{-1}f \\ &= (1 - \omega)Id + \omega(Id - D^{-1}A)u^k + \omega D^{-1}f \end{aligned} \tag{3.18}$$

Gleichung 3.18 ist die Iterationsvorschrift für das Jacobi-Relaxationsverfahren. Die Iterationsmatrix ist offensichtlich gegeben durch:

$$\mathbf{T}_{J_\omega} := (\mathbf{Id} - \omega\mathbf{D}^{-1}\mathbf{A}) = (1 - \omega)\mathbf{Id} + \omega(\mathbf{Id} - \mathbf{D}^{-1}\mathbf{A}) \tag{3.19}$$

Dies führt auf folgenden Algorithmus:

3.3.1 Algorithmus des Jacobi-Relaxations-Verfahrens

Sei $u^0 \in \mathbb{R}^n$ ein *beliebiger* Startvektor und $k = 1, 2, ...$ berechne für $i = 1, ..., n$:

$$u_i^{k+1} = (1-\omega)u_i^k + \frac{\omega}{a_{ii}}(f_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}u_i^k) \tag{3.20}$$

Beachte: Für $\omega = 1$ erhalten wir das Jacobi-Verfahren. Auch hier beträgt der Rechenaufwand $\mathcal{O}(n^2)$. Angewandt auf die 2D Poisson Gleichung gilt:

3.3.2 Algorithmus des Jacobi-Relaxations-Verfahrens für $\mathbf{A}_{2D}$

Berechne für $k = 1, 2, ...$ mit Startvektor $u^0 \in \mathbb{R}^n$ *beliebig*
Für $i = 1, ..., N$ und für $j = 1, ..., N$:

$$u_{i,j}^{k+1} = (1-\omega)u^k + \frac{\omega}{a_{i,i}}(f_{i,j} - u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k) \tag{3.21}$$

mit Rechenaufwand $\mathcal{O}(n)$.

3.3.3 Satz (Eigenwerte des Jacobi-Relaxationsverfahren bzgl. $\mathbf{A}_{2D}$)

Auch hier entsprechen die Eigenvektoren denen von $\mathbf{A}_{2D}$ und für die Eigenwerte von $\mathbf{T}_{J_\omega}$ gilt:

$$\lambda_{i,j}(\mathbf{T}_J) = 1 - 4\omega\left(\sin^2(\frac{\theta_i}{2}) + \sin^2(\frac{\theta_j}{2})\right) \text{ für } 1 \leq i, j \leq N \tag{3.22}$$

Beweis:

Analog zu Gleichung 3.2.3.

3.3.4 Lemma (Spektralradius der Jacobi-Relaxations-Matrix bzgl. $\mathbf{A}_{2D}$)

Das Jacobi-Relaxations-Verfahren konvergiert für die diskrete Poisson Gleichung und es gilt für den Spektralradius:

$$\rho(\mathbf{T}_J) = \omega\cos(\pi h) < 1 \tag{3.23}$$

Beweis: Ebenfalls analog zu Gleichung 3.2.4, wobei nun gilt:

$$1 - \omega 2 \sin^2(\frac{\pi h}{2}) = \omega \cos(\pi h) \overset{\text{Taylorformel}}{\approx} 1 - \frac{\omega}{2}\pi^2 h^2 \tag{3.24}$$

Das Jacobi-Relaxations-Verfahren konvergiert für $\mathbf{A}_{2D}$ sogar schneller für ein $\omega \in (0,1)$. Häufig verwendete Werte für ω sind $\frac{1}{2}$ und $\frac{2}{3}$. Leider ist für kleines h die Konvergenz immer noch sehr langsam.

3.4 Glättungseigenschaft

Die Iterationsvorschrift des Jacobi-Relaxations-Verfahrens war gegeben durch:

$$u^{k+1} = (Id - \omega D^{-1}A)u^k + \omega D^{-1} f \tag{3.25}$$

Die Eigenvektoren sind wegen Unterabschnitt 3.2.3 und Unterabschnitt 3.3.3 gegeben durch:

$$v_{i,j} = sin(k\pi x_i) sin(l\pi y_j) \text{ für } 1 \leq k, l \leq N \tag{3.26}$$

und $x_i, y_j \in \Omega_h$.
Als Eigenvektoren der Matrizen $\mathbf{T}_J$ bzw. $\mathbf{T}_{J_\omega}$ bilden diese Vektoren eine Basis. Für den Fehlerterm im $k-ten$ Iterationsschritt gilt:

$$e^k = u^* - u^k \tag{3.27}$$

Betrachten wir nun den $(k+1)-ten$ Fehler und formen geschickt um, dann gilt:

$$\begin{aligned} e^{k+1} &= u^* - u^{k+1} = u^* - \Big((Id - \omega D^{-1}A)u^k + \omega D^{-1} f\Big) \\ &= \underbrace{u^* - u^k}_{=e^k} + \omega D^{-1} A u^k - D^{-1} f = e^k + \omega D^{-1}(Au^k - f) \\ &= e^k + \omega D^{-1}(Au^k - Au^*) = e^k + \omega D^{-1} A(u^k - u^*) \\ &= e^k + \omega D^{-1} A e^k = (Id - \omega D^{-1} A)e^k \end{aligned} \tag{3.28}$$

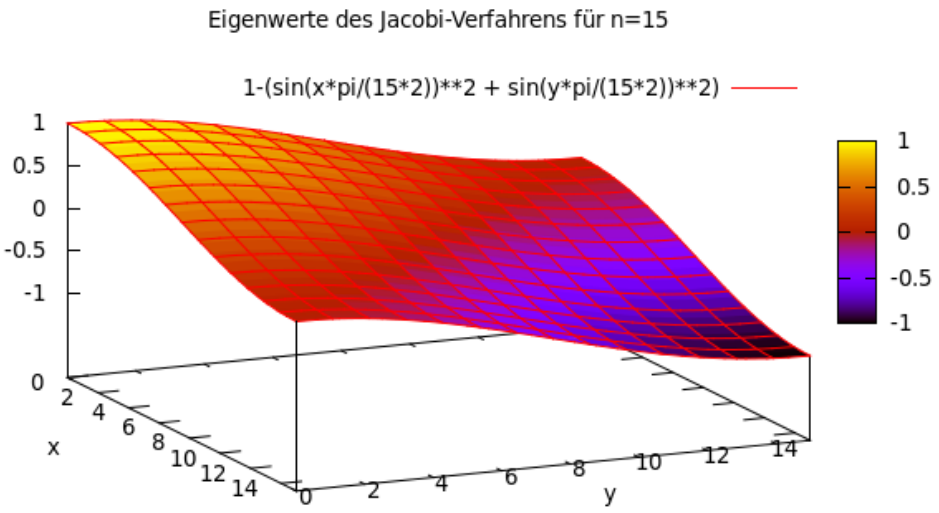

Abbildung 3.1: Plot für ein Omega blablalba

Da die n (zur Erinnerung: $n = N^2$) Eigenvektoren $v_{i,j}$ eine Basis bilden, lässt sich der Fehler e als Linearkombination der $v_{i,j}$ darstellen:

$$
\begin{aligned}
e^{k+1} &= \sum_{i=1}^{n} \xi_{(k+1)}^{(i)} v^{(i)} = (Id - \omega D^{-1} A) \left(\sum_{i=1}^{n} \xi_{(k)}^{(i)} v^{(i)} \right) \\
&= \left(\sum_{i=1}^{n} \xi_{(k)}^{(i)} (Id - \omega D^{-1} A) v^{(i)} \right) = \left(\sum_{i=1}^{n} \xi_{(k)}^{(i)} (v^{(i)} - \frac{\omega}{4} \underbrace{A v^{(i)}}_{=\lambda(A) v^{(i)}}) \right) \\
&= \left(\sum_{i=1}^{n} \xi_{(k)}^{(i)} \underbrace{(1 - \frac{\omega}{4} \lambda(A))}_{=\lambda(\mathbf{T}_{J_\omega})} v^{(i)} \right) = \left(\sum_{i=1}^{n} \xi_{(k)}^{(i)} \lambda(\mathbf{T}_{J_\omega}) v^{(i)} \right)
\end{aligned}
\tag{3.29}
$$

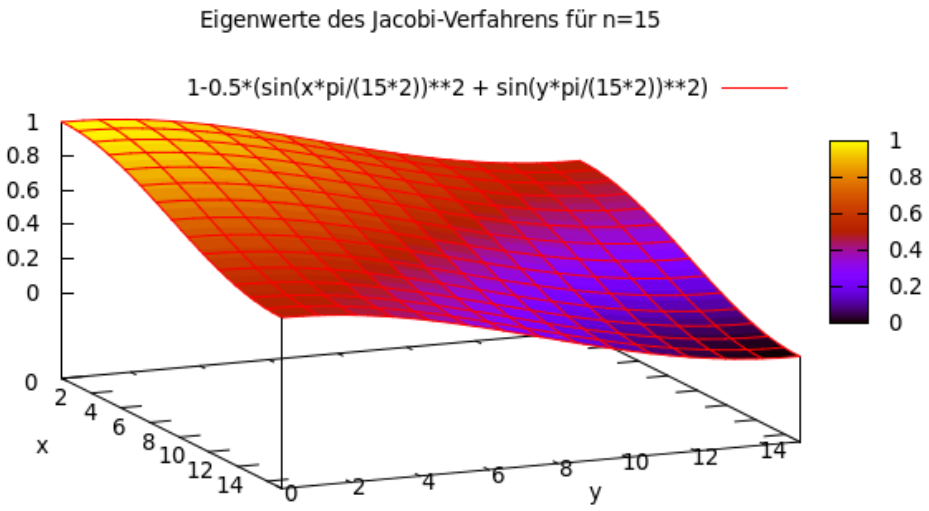

Abbildung 3.2: Plot für ein Omega blablalba

Betrachten wir nun die Eigenwerte der Iterationsmatrix für $\omega = 1$ (Jacobi-Verfahren), sieht man, dass die Eigenwerte für i, j nahe Null oder i, j nahe N den Fehler schlecht bis gar nicht dämpfen (Abbildung 3.1). Dies erklärt auch das langsame Konvergenzverhalten des Jacobi-Verfahren (siehe Kapitel 5). Interessanter weiße ist der optimale Wert $\omega = 1$, wenn man das Ganze als iteratives Verfahren verwendet [Saad]. Das Jacobi-Verfahren besitzt aber keine Glättungseigenschaft.
Da wir aber eine Fehlerglättung erreichen wollen, wählen wir nun $\omega = \frac{1}{2}$ (Abbildung 3.2). Wir stellen fest, dass die Eigenwerte der Iterationsmatrix zwischen 0 und 1 liegen. Sind $i, j > \frac{N}{2}$ so werden die Fehleranteile wesentlich besser gedämpft, da die Eigenwerte hier nahe Null liegen. Diese Eigenschaft nennt man die *Glättungseigenschaft*.
Es stellt sich z.B. heraus [Saad], dass der optimale Relaxationsparameter, der unabhängig von der Schrittweite h gewählt werden kann, $\omega = \frac{4}{5}$ ist. In Abbildung 3.3 ist gut zu sehen, dass viele der Eigenwerte nahe Null liegen.

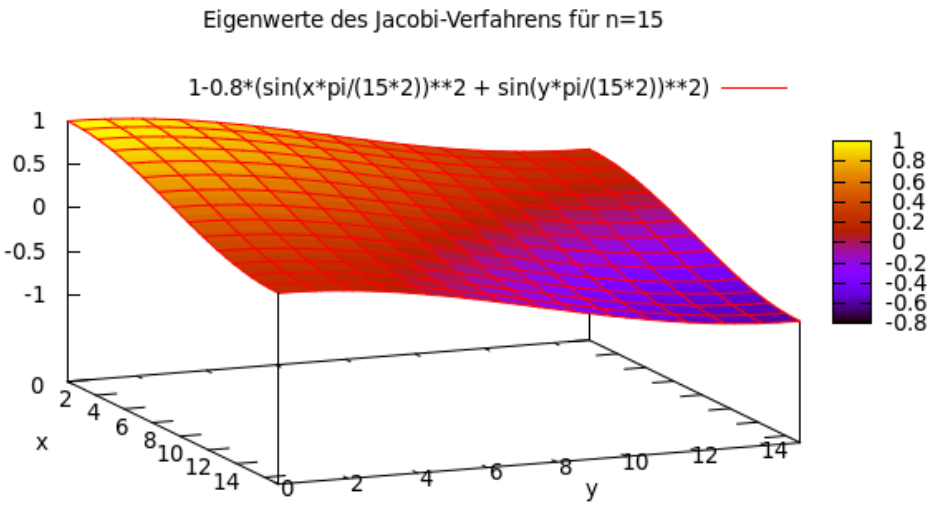

Abbildung 3.3: Plot für ein Omega blablalba

3.5 Das Verfahren der konjugierten Gradienten

Das Verfahren der konjugierten Gradienten wurde 1952 von Heestens und Stiefel erstmals vorgestellt. Das Verfahren zeichnet sich durch Stabilität und schnelle Konvergenz aus. Das CG-Verfahren (conjugate gradient) - wie es auch genannt wird - ist eine Projektionsmethode, bei der jedoch die Matrix **A** eine wesentliche Rolle spielt.

3.5.1 Definition (A-orthogonal)

Sei **A** eine symmetrische, nicht singuläre Matrix. Zwei Vektoren $x, y \in \mathbb{R}^n$ heißen **konjugiert** oder **A-orthogonal**, wenn $x^T Ay = 0$ gilt.

3.5.2 Lemma - (A-orthogonaler) Projektionssatz

Sei U_k ein k-dimensionaler Teilraum des $\mathbb{R}^n$ $(k \leq n)$, und $p^0, p^1, ..., p^{k-1}$ eine *A-orthogonale Basis* dieses Teilraums, also $\langle p^i, p^j \rangle_A = 0 \;\; für \;\; i \neq j$. Sei $v \in \mathbb{R}^n$, dann gilt für $u^k \in U_k$:

$$\|u^k - v\|_A = \min_{u \in U_k} \|u - v\|_A \tag{3.30}$$

genau dann, wenn u^k die *A-orthogonale Projektion* von v auf $U_k = span\{p^0, ..., p^{k-1}\}$ ist. Außerdem hat $\mathbf{u}^k$ die Darstellung

$$P_{U_{k,\langle\cdot,\cdot\rangle}}(v) = \mathbf{u}^k = \sum_{j=0}^{k-1} \frac{\langle v, p^j\rangle_A}{\langle p^j, p^j\rangle_A} p^j \tag{3.31}$$

Der Beweis zu diesem Lemma folgt direkt aus dem Projektionssatz. (siehe auch Dahmen/Reusken)

3.5.3 Allgemeiner Algorithmus der konjugierten Gradienten

Zur Erzeugung der Lösung von x^* durch Näherungen $x^1, x^2, ...$ definieren wir folgende Teilschritte:

0. Definiere den Teilraum U_1 und bestimme das Residuum r^0 mit Startvektor x^0

$$U_1 := span\{r^0\}, \text{ wobei } r^0 = b - Ax^0 \tag{3.32}$$

1. Bestimme eine A-orthogonale Basis

$$p^0, ..., p^{k-1} \text{ von } U_k \tag{3.33}$$

2. Bestimme eine Näherungslösung x^k, so dass

$$\|x^k - x^*\|_A = \min_{u \in U_k} \|x - x^*\|_A \tag{3.34}$$

gilt. Mit dem (A-orthogonalen) Projektionssatz berechnen wir also:

$$x^k = \sum_{j=0}^{k-1} \frac{\langle x^*, p^j\rangle_A}{\langle p^j, p^j\rangle_A} p^j \tag{3.35}$$

3. Erweitere den Teilraum U_k und berechne das iterierte Residuum

$$U_{k+1} := span\{p^0, ..., p^{k-1}, r^k\} \text{ wobei } r^k := b - Ax^k \tag{3.36}$$

Natürlich ist das kein numerischer Algorithmus, den man in Programmcode umsetzen kann, allerdings sollte man sich die Schritte des CG-Algorithmus klar machen, um die Effizienz dahinter zu verstehen:

3.5.3.1 Erklärung zum CG-Algorithmus

zu 0. Initialisiere den Algorithmus und bestimme das Residuum. Hierfür ist ein Startvektor x^0 beliebig zu wählen.

zu 1. Um eine A-orthogonale Basis von den U_k zu bestimmen ist im wesentlichen ein (A-) Orthogonalisierungverfahren notwendig.

zu 2. Hier wird das Verfahren aus Unterabschnitt 3.5.2 angewandt, um eine neue Näherungslösung in U_k zu bestimmen.

zu 3. Erweiterung des Teilraums U_k zu U_{k+1} durch das Hinzufügen des $k-ten$ Resdiduums (berechne: $r^k := b - Ax^k$).

3.5.4 Numerischer Algorithmus der konjugierten Gradienten

Gegeben ist eine symmetrisch positiv definite Matrix $\mathbf{A} \in \mathbb{R}^n$. Bestimme die (Näherungs-) Lösung x^* mit Hilfe eines *beliebigen* Startvektors $x^0 \in \mathbb{R}^n$ zu einer gegebenen rechten Seite $b \in \mathbb{R}^n$. Setze $\beta_{-1} := 0$ und berechne das Residuum $r^0 = b - Ax^0$.
Für $k = 1, 2, ...$, falls $r^{k-1} \neq 0$ berechne:

$$\begin{aligned} p^{k-1} &= r^{k-1} + \beta_{k-2} p^{k-2}, \text{ wobei } \beta_{k-2} = \frac{\langle r^{k-1}, r^{k-1} \rangle}{\langle r^{k-2}, r^{k-2} \rangle} \text{ mit } (k \geq 2) \\ x^k &= x^{k-1} + \alpha_{k-1} p^{k-1}, \text{ wobei } \alpha_{k-1} = \frac{\langle r^{k-1}, r^{k-1} \rangle}{\langle p^{k-1}, Ap^{k-1} \rangle} \\ r^k &= r^{k-1} - \alpha_{k-1} A p^{k-1} \end{aligned}$$

Man muss in diesem Algorithmus pro Iterationsschritt lediglich zwei Skalarprodukte ausrechnen und eine Matrix-Vektor-Multiplikation durchführen. Somit erhält man einen Rechenaufwand von $\mathcal{O}(n^2)$. Angewandt auf $\mathbf{A}_{2D}$ kann man - durch das Ausnutzen der Dünnbesetztheit - einen Aufwand pro Schritt von $\mathcal{O}(n)$ erreichen.

An dieser Stelle soll noch ein kurzer Satz über die Konvergenz des Verfahrens folgen. Einen Beweis hierzu findet man z.B. in (Dahmen/Reusken 573):

3.5.5 Satz (Konvergenz des CG-Algorithmus) [DahmenReusken]

Sei $\mathbf{A} \in \mathbb{R}^{n\times n}$ symmetrisch, positiv definit und seien $u, f, u^*, u^k \in \mathbb{R}^n$, wobei u^* die exakte Lösung des Gleichungssystems $Au = b$ und x^k die approximierte Lösung durch das CG-Verfahren ist. Dann gilt für $k = 1, 2, ...$:

$$\|x^k - x^*\|_A \leq 2\left(\frac{\kappa_2(A)-1}{\kappa_2(A)+1}\right)^k \|x^0 - x^*\|_A \tag{3.37}$$

Da stets $\frac{\kappa_2(A)-1}{\kappa_2(A)+1} < 1$ gilt, sichert dieser Satz die Konvergenz des Algorithmus. Man sieht also, dass die Konvergenz des Verfahrens von der Kondition der Matrix $\mathbf{A}$ abhängt.

3.6 Vorkonditioniertes Verfahren der konjugierten Gradienten (PCG)

Das PCG-Verfahren (preconditioned conjugate gradient) ist eine optimierte Version des CG-Verfahrens. Wie wir in Unterabschnitt 2.3.4 gesehen haben, ist z.B. $\mathbf{A}_{2D}$ schlecht konditioniert. Die Idee ist nun, die bei der Iterations zu Grunde liegende Matrix $\mathbf{A}$ durch eine ähnliche Matrix mit besserer Kondition zu ersetzen, damit sich das Konvergenzverhalten verbessert.

3.6.1 Satz

Sei $\mathbf{W} \in \mathbb{R}^{n\times n}$ s.p.d. dann gilt:

$$\mathbf{A}u = f \Longleftrightarrow \mathbf{W}^{-1}\mathbf{A}x = \mathbf{W}^{-1}b \tag{3.38}$$

3.6.1.1 Beweis:

$$\begin{aligned} \mathbf{A}u = f &\iff u = \mathbf{A}^{-1}\mathbf{E}f \Longleftrightarrow u = \mathbf{A}^{-1}\mathbf{W}\mathbf{W}^{-1}f \\ &\iff u = (\mathbf{W}^{-1}\mathbf{A})^{-1}f \Longleftrightarrow \mathbf{W}^{-1}\mathbf{A}x = \mathbf{W}^{-1}b \end{aligned}$$

Die Konditionszahl dieses Problem ist nun durch $\kappa_2(\mathbf{W}^{-1}\mathbf{A})$ bedingt. Das Ziel muss es also sein, $\mathbf{W}^{-1}$ so gut wie möglich zu wählen, damit die Kondition möglichst klein wird. Nun ist im Allgemeinen $\mathbf{W}^{-1}\mathbf{A}$ nicht s.p.d. Somit könnten wir zwar den CG-Alorithmus trotzdem darauf anwenden, werden aber wegen dieser Tatsache möglicherweise keine Konvergenz

erhalten. Um dies zu umgehen findet man z.B. in (Dahmen/Reusken 576) einen Lösungsansatz, bei dem mit der Cholesky-Zerlegung eine entsprechende Umformung gefunden werden kann.

3.6.2 Der Algorithmus des vorkonditionierten konjugierten Gradienten Verfahrens

Gegeben seien $\mathbf{A}, \mathbf{W} \in \mathbb{R}^n$ s.p.d. Bestimme die (Näherungs-) Lösung x^* mit Hilfe eines *beliebigen* Startvektors $x^0 \in \mathbb{R}^n$ zu einer gegebenen rechten Seite $b \in \mathbb{R}^n$. Setze $\beta_{-1} := 0$, berechne das Residuum $r^0 = b - Ax^0$ und $z^0 = \mathbf{W}^{-1}r^0$ (löse $\mathbf{W}z^0 = r^0$).
Für $k = 1, 2, ...$, falls $r^{k-1} \neq 0$ berechne:

$$\begin{aligned} p^{k-1} &= z^{k-1} + \beta_{k-2}p^{k-2}, \text{ wobei } \beta_{k-2} = \frac{\langle z^{k-1}, r^{k-1}\rangle}{\langle z^{k-2}, r^{k-2}\rangle} \text{ mit } (k \geq 2) \\ x^k &= x^{k-1} + \alpha_{k-1}p^{k-1}, \text{ wobei } \alpha_{k-1} = \frac{\langle z^{k-1}, r^{k-1}\rangle}{\langle p^{k-1}, Ap^{k-1}\rangle} \\ r^k &= r^{k-1} - \alpha_{k-1}Ap^{k-1} \\ z^k &= \mathbf{W}^{-1}r^k \text{ (löse } \mathbf{W}z^k = r^k) \end{aligned}$$

Wichtig hierbei ist, dass das Lösen von $\mathbf{W}z^k = r^k$ mit möglichst wenig Aufwand (ideal: $\mathcal{O}(n)$) berechnet werden soll.

3.6.3 Die unvollständige Cholesky-Zerlegung

Eine Matrix $\mathbf{A}$, die symmetrisch positiv definit ist, lässt sich durch eine Cholesky-Zerlegung in eine normierte untere Dreiecksmatrix $\mathbf{L}$ und eine rechte obere Dreicksmatrix $\mathbf{U}$ zerlegen, wobei gilt: $\mathbf{U} := \mathbf{D}\mathbf{L}^T$
Mit dieser Zerlegung möchten wir nun unser System vorkonditionieren. Allerdings würde eine vollständige Cholesky-Zerlegung viele Nulleinträge in einer Sparse-Matrix auslöschen. Darum greift man auf eine unvollständige Cholesky-Zerlegung zurück, bei der die Stellen, an denen $\mathbf{A}$ Nulleinträge besitzt, ebenfalls Null werden.

3.6.3.1 Definition (Das Muster E)

Sei das Muster $E \subseteq \{1, ..., n\} \times \{1, ..., n\}$, dann gilt:

$$E := \{(i, j) | 1 \leq i, j \leq n, a_{i,j} \neq 0\} \tag{3.39}$$

Dann lässt sich die Matrix $\mathbf{A}$ auch folgendermaßen schreiben:

$$\mathbf{A} = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T + E \approx \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T \tag{3.40}$$

wobei $\tilde{\mathbf{L}}, \tilde{\mathbf{L}}^T$ nicht die komplette Faktorisierung darstellt, sondern folgende Eigenschaften erfüllt:

3.6.3.2 Eigenschaften der Matrix $\tilde{\mathbf{L}}$

- $\tilde{\mathbf{L}}$ ist normierte untere Dreicksmatrix
- Es gilt: $l_{i,j} = 0$, falls $(i,j) \notin E$

Natürlich ist diese Faktorisierung ungenauer, als die vollständige Zerlegung, allerdings genügt sie, um die Kondition des Gleichungssystems in vielen Fällen zu verbessern. Um den folgenden Algorithmus effizient zu machen, werden Summen nur über Indizes aus dem Muster berechnet.

3.6.3.3 Der numerische Algorithmus der unvollständigen Cholesky Zerlegung

Seien $\mathbf{A} \in \mathbb{R}^{n\times n}$ s.p.d. und E das Muster zur Matrix $\mathbf{A}$. Berechne dann für $i = 1, 2, ..., n$:

$$l_{i,i} = \left(a_{i,i} - \sum_{j=1,(i,j)\in E}^{i-1} l_{i,j}^2 \right)^{\frac{1}{2}} \tag{3.41}$$

$$\text{for k} = i+1, ..., n : \text{ if } (k,i) \in E : \tag{3.42}$$

$$l_{k,i} = \left(a_{k,i} - \sum_{j=1,(k,j)\in E,(i,j)\in E}^{k-1} l_{k,j} l_{i,j} \right) / l_{i,i} \tag{3.43}$$

Bemerkungen:

- Die für den PCG-Algorithmus wichtige Matrix $\mathbf{W}$ wird nun definierte als: $\mathbf{W} := \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T$. Dadurch wird auch $\mathbf{W}z^k = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T z^k = r^k$ schnell durch Vorwärts- bzw. Rückwärtseinsetzen lösbar.
- Für viele Probleme zeigt sich, dass $\kappa_2(\mathbf{W}^{-1}\mathbf{A}) \ll \kappa_2(A)$ gilt.

Es gibt einige Varianten dieses Verfahrens. Wir wollen uns an dieser Stelle mit einer dieser auseinander setzen.

3.6.4 Die modifizierte unvollständige Cholesky-Zerlegung

Auch bei der modifizierten Methode des Verfahrens gehen wir vor, wie in Unterabschnitt 3.6.3. Jedoch werden die Vorschriften für die Matrix $\tilde{\mathbf{L}}$ abgeändert:

3.6.4.1 Eigenschaften der Matrix $\tilde{\mathbf{L}}$

Sei $e := (1,,1,...,1)^T$

- $a_{i,j} = (\tilde{\mathbf{L}}\tilde{\mathbf{L}}^T)_{i,j}$ für alle $(i,j) \in E, i \neq j$
- $\mathbf{A}e = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T e$, d.h. die Zeilensummen stimmen überein.
- $l_{i,j} = r_{i,j} = 0$ für alle $(i,j) \notin E$

3.6.4.2 Der numerische Algorithmus der modifizierten unvollständigen Cholesky-Zerlegung

Seien $\mathbf{A} \in \mathbb{R}^{n\times n}$ s.p.d. und E das Muster zur Matrix $\mathbf{A}$. Berechne dann für $i = 1,2,...,n$:

$$l_{i,i} = \left(a_{i,i} - \sum_{j=1,(i,j)\in E}^{i-1} l_{i,j}^2 \right)^{\frac{1}{2}} \tag{3.44}$$

$$\text{for k} = i+1,...,n : \tag{3.45}$$

$$\text{if} \quad (k,i) \in E : \tag{3.46}$$

$$l_{k,i} = \left(a_{k,i} - \sum_{j=1,(k,j)\in E,(i,j)\in E}^{k-1} l_{k,j} l_{i,j} \right) / l_{i,i} \tag{3.47}$$

$$\text{else} \quad : \tag{3.48}$$

$$a_{k,k} = a_{i,i} - \sum_{j=1,(k,j)\in E,(i,j)\in E}^{k-1} l_{k,j} l_{i,j} \tag{3.49}$$

Wir setzen wieder $\mathbf{W} := \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T$.
Es gibt noch weitere Verfahren (siehe z.B. Saad), wie beispielsweise das SSOR-Verfahren, die wir hier nicht weiter diskutieren wollen.

4 Mehrgitterverfahren

In diesem Abschnitt sollen nun die Mehrgittermethoden genauer betrachtet werden. Bevor wir jedoch genauer auf dieses Verfahren eingehen, wollen wir uns nochmal einige Erkenntnisse klar machen:

4.1 Grundideen

1. Auslöschung hochfrequenter Fehler
Das Gauß-Seidel-Verfahren und das Jacobi-Verfahren löschen hochfrequente Fehler in den ersten Iterationsschritten aus. Niederfrequente Fehler werden nur sehr langsam beseitigt.(siehe Abschnitt 3.2 und **??**)

2. Grobe Fehler nach einer Gittertransformation
Niedrig frequente Fehler auf einem feinen Gitter werden zu hochfrequenten Fehlern, wenn sie auf ein gröberes Gitter überführt werden.

3. Residuumsgleichung
Die für diesen Algorithmus wichtige Residuumsgleichung lautet:

$$\mathbf{A}e^k = r^k \tag{4.1}$$

Die Lösung von $\mathbf{A}e^k = r^k$ ist äquivalent zur Lösung von $\mathbf{A}u = b$, wobei $e^k = 0$.

Beweis:

Das Residuum ist an der $k - ten$ Stelle definiert als

$$r^k = b - \mathbf{A}u^k \tag{4.2}$$

Der Fehler

$$e = u^* - u^k \tag{4.3}$$

wobei u^* die exakte Lösung darstellt, erfüllt ebenso folgende Gleichung:

$$\mathbf{A}e^k = \mathbf{A}(u^* - u^k) = \mathbf{A}u^* - \mathbf{A}u^k = b - \mathbf{A}u^k = r^k \tag{4.4}$$

Wir kennen zwar den Fehler e^k nicht, wissen aber, dass dieser 0 ist, falls $r^k = 0$. $\Longrightarrow$ Beh.

Gauss-Seidel- und Jacobi-Verfahren löschen also hochfrequente Fehler in den ersten Itertionsschritten aus. Um die nieder frequenten Fehler zu reduzieren sind allerdings wesentlich mehr Iterationsschritte notwendig. Auch aus diesem Grund finden beide Verfahren bei der Lösung großer, linearer Gleichungssysteme wenig Anwendung.
Auch wenn die hohe Anzahl an notwendigen Iterationen ein deutlicher Nachteil ist, wollen wir im folgenden die Vorteile dieser Methoden ausnutzen.
Wie inAbschnitt 2.2 gesehen befinden wir uns bei der Diskretisierung der Poisson-Gleichung auf einem Gebiet $\Omega_h = (0,1)^2$ der Schrittweite $h = \frac{1}{n}$. Nach der Ausführung von k-Iterationsschritten von Einzel- oder Gesamtschrittverfahren sind auf diesem Gitter die hochfrequenten Fehler $e^k = u^* - u^k$ verschwunden. Nun berechnet man im $k - ten$ Schritt das Residuum r^k und führt für das äquivalente lineare Gleichungssystem $\mathbf{A}e^k = r^k$, wobei $e^k = 0$ gilt, l Iterationsschritte aus. So erhalten wir eine Näherung des Fehlers e^k.
Stellt man Gleichung 4.3 um, berechnet also $e^k + u^k$, so erhält man eine neue Näherung der exakten Lösung.
Kombiniert man dieses Vorgehen nun mit dem Wechsel zwischen zwei Gittern der Gitterweite h und $2h$ so erhält man das Zweigitterverfahren:

4.2 Der Zweigitter-Algorithmus

Der Zweigitter-Algorithmus findet in der Praxis zwar wenig Verwendung, allerdings wird die Idee dahinter die Basis für das Mehrgitter-Verfahren sein.

while	$u^k \neq u^*$	
	pre-smooth	Jacobi-/Gauss-Seidel-Steps
	calculate residual	$r^k = b - Au^k$
	restrict	$r^k_{2h} = Rr^k_h$
		$\mathbf{A}^{2h} = R\mathbf{A}^h P$
	set error	$e^0_{2h} = 0$
	solve direct	$\mathbf{A}^{2h} e_{2h} = r^k_{2h}$
	prolongate	$e^k_h = Pe^k_{2h}$
	add error	$u^k_h = u^{k-1}_h + e^k_h$
	smooth (optional)	Jacobi-/Gauss-Seidel-Steps
end		

Zunächst sei erwähnt, dass das Nachglätten optional ist, allerdings einige Vorteile bietet, auf die wir hier nicht genauer eingehen möchten. Bei der Implementierung sollte also auf Nachglättung geachtet werden.

Der klare Nachteil dieser Methode liegt natürlich im direkten Lösen der Residuumsgleichung. Wählen wir ein sehr feines Gebiet Ω_h mit $n = 256$, also $h = \frac{1}{256}$, so liefert das Gebiet Ω_{2h} immer noch ein Gleichungssystem der Dimension 127^2. Ein System dieser Ordnung zu lösen erfordert Rechenaufwand, der hier nicht erwünscht ist.

4.3 Der Mehrgitter-Algorithmus

Eine bessere Methode ist, das Gitter immer feiner zu machen, bis das System direkt lösbar ist, um dann wieder auf das feinste Gitter zurück zu kehren. Genauer:

Multigrid (u, b)

if *(finest grid)*	return $u_{finestgrid} = \mathbf{A}^{-1}b$
else	
pre-smooth	Jacobi-/Gauss-Seidel-Steps
calculate residual	$r^k = b - Au^k$
restrict	$r^k_{2h} = Rr^k_h$
	$\mathbf{A}^{2h} = R\mathbf{A}^h P$
recursion	$e^k_{2h} =$ **Multigrid** $(0, r^k_{2h})$
prolongate	$e^k_h = Pe^k_{2h}$
add error	$u^k_h = u^{k-1}_h + e^k_h$
smooth	Jacobi-/Gauss-Seidel-Steps
	return u_h

end

Nun wollen wir die Idee, die wir in Abschnitt 4.2 entwickelt haben formulieren. Hier unterscheiden wir dann zwischen zwei Methoden: dem **V-Zyklus** und dem **W-Zyklus**. Bei einem V-Zyklus wird geht man pro Iterationsschritt vom feinsten zum gröbsten Gitter und zurück (siehe Bild bla). Bei einem W-Zyklus geht man vom feinsten auf das gröbste Gitter über, prolongiert

5 Ein Vergleich zwischen Iterativen- und Mehrgitter-Methoden

In diesem letzten Kapitel wollen wir die Iterativen-Verfahren, speziell das Verfahren der konjugierten Gradienten, und die Mehrgitterverfahren numerisch vergleichen. Dafür betrachten wir folgende Gleichung:

5.1 Beispiel einer Poisson Gleichung

Seien $f : \Omega \to \mathbb{R}$ und $g : \partial\Omega \to \mathbb{R}$ stetige Funktionen mit $f(x,y) = -4.$ und $g(x,y) = x^2 + y^2$. Sei außerdem $\Omega = (0,1) \times (0,1) \in \mathbb{R}^2$.Gegeben ist das Randwertproblem

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) = -4 \text{ in } \Omega & (5.1)\\ u(x,y) &= g(x,y) = x^2 + y^2 \text{ in } \partial\Omega & (5.2) \end{aligned}$$

Gesucht ist eine Funktion $u(x,y)$, die diese Gleichung löst.
Offensichtlich löst der elliptische Paraboloid $u(x,y) = x^2 + y^2$ die partielle Differentialgleichung, da $\partial_{xx}u(x,y) = \partial_{yy}u(x,y) = 2$. Allerdings wollen wir nun diese Lösung auch numerisch erhalten.

5.2 Jacobi-Verfahren angewandt auf das Beispiel

5.3 CG-Verfahren angewandt auf das Beispiel

5.4 PCG-Verfahren angewandt auf das Beispiel

5.5 Das Mehrgitterverfahren angewandt auf das Beispiel

5.5.1 V-Zyklus

5.5.2 W-Zyklus

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum Unterschrift