Karl-Franzens Universität Graz
Institut für Physik

Projekte für Computational Physics II

Computermethoden der linearen Algebra

Andreas Windisch

1. Oktober 2010

Erklärung

Diese Ausarbeitung entstand im Rahmen der Lehrveranstaltung Computational Physics II. Dabei sind die ersten Kapiteln aus der Vorlesungsmitschrift aufgebaut, wie sie Prof. Gattringer präsentiert hat. Im zweiten Abschnitt befinden sich die ausgearbeiteten Projekte, welche ebenfalls unter der Anleitung von Prof. Gattringer entwickelt wurden.

Inhaltsverzeichnis

1 Drei Beispiele für Anwendungen der linearen Algebra

1.1 Ebenes Fachwerk

Man denke sich folgendes ebene Fachwerk.

Idealisierung: Die Stäbe können nur auf Zug und Druck belastet werden.

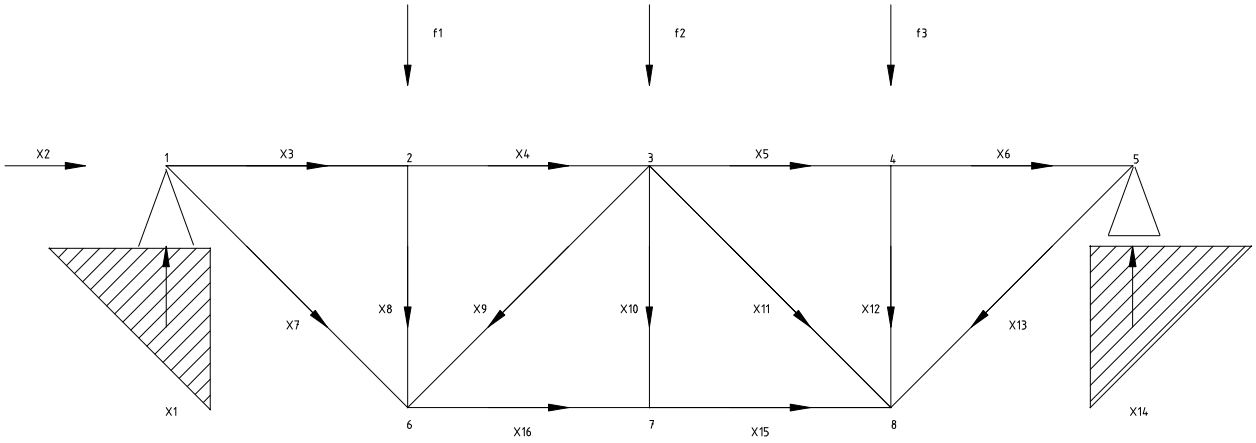


Abbildung 1: Ebenes Fachwerk mit Kräfteverteilung

Zweiwertige Lager: Diese können Kräfte in x und z Richtung aufnehmen.
Einwertige Lager: Diese können nur Kräfte in z Richtung aufnehmen.

Das Problem ist statisch bestimmt falls gilt:

$$S + b = 2 \cdot K \tag{1.1}$$

Dabei ist S die Anzahl der Stäbe ($= 13$), K die Anzahl der Knoten ($= 8$) und b die Summe der Wertigkeiten der Lager ($= 3$). Unser Problem ist also berechenbar. Der Faktor 2 in der Gleichung kommt von der Dimensionalität des Problemes.

Gesucht: $x_i = |\vec{x_i}| + Vorzeichen.$
An jedem Knoten müssen die Summen der Kräfte verschwinden. Dazu betrachten wir die Knoten. Bei der Erstellung des Gleichungssystemes setzt man also Knoten für Knoten an.

Knoten 1:

$$\begin{aligned} \vec{x_1} + \vec{x_2} + \vec{x_3} + \vec{x_7} &= 0 \\ x_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} + x_2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_3 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_7 \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} &= 0 \\ x_2 + x_3 + \frac{1}{\sqrt{2}} x_7 &= 0 \\ x_1 - \frac{1}{\sqrt{2}} x_7 &= 0 \end{aligned} \tag{1.2}$$

Knoten 2:

$$\begin{aligned} \vec{x_3} + \vec{x_4} + \vec{x_8} + \vec{f_1} &= 0 \\ x_3 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_4 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_8 \begin{pmatrix} 0 \\ -1 \end{pmatrix} + f_1 \begin{pmatrix} 0 \\ -1 \end{pmatrix} &= 0 \\ x_3 + x_4 &= 0 \\ -x_8 &= f_1 \end{aligned} \tag{1.3}$$

In dieser Weise fährt man nun fort um das Gleichungssystem zu formulieren (siehe Übung 1). So erhalten wir 16 Gleichungen für $x_1, x_2, \ldots, x_{16}$. Die Gleichungen bilden ein lineares Gleichungssystem:

$$A\vec{x} = \vec{F}, \tag{1.4}$$

wobei $\vec{x}$ Vektor mit den Komponenten $x_1, \ldots, x_{16}$, $\vec{F}$ Vektor mit Komponenten f_i und A eine 16×16-Matrix ist. Sie beinhaltet die Geometrie der Struktur.
Anmerkung: Diese Methode kann auch verallgemeinert werden um etwa die Kräfteverteilung in einem Kranhaken, also einem kontinuierlichen Teil, zu bestimmen. Dazu wird das betreffende Bauteil durch Simplices diskretisiert (z.B. Dreiecke, Tetraeder und dgl.), an deren Knotenpunkten wieder die betreffenden Gleichungen formuliert werden. So führt auch dies auf ein (großes) lineares Gleichungssystem.

1.2 Ein quantenmechanisches Problem

$$\begin{aligned} \psi(\vec{x}, t) = \psi(\vec{x}) \cdot e^{\frac{iE \cdot t}{\hbar}} &\Rightarrow \\ \hat{H}\psi(\vec{x}) = E\psi(\vec{x}) & \\ \psi(\vec{x}) \rightarrow |\psi\rangle & \\ \hat{H}|\psi\rangle = E|\psi\rangle & \end{aligned} \tag{1.5}$$

Dabei ist $\hat{H}$ Hamiltonoperator (z.B. $\hat{H} = -\frac{\Delta}{2m} + V(\vec{x})$),
$|\psi\rangle$ Wellenfunktion und E Energieeigenwert.

Mit Basiswellenfunktionen $\psi_i(\vec{x})$ und Koeffizienten $\alpha_i \in \mathbb{C}$ kann man schreiben:

$$\psi(\vec{x}) = \sum_i \alpha_i \psi_i(\vec{x}) \tag{1.6}$$
$$|\psi\rangle = \sum_i \alpha_i |i\rangle.$$

So folgt:

$$\hat{H}|\psi\rangle = \hat{H} \sum_i \alpha_i |i\rangle = \sum_i \alpha_i \hat{H}|i\rangle = E|\psi\rangle = E \sum_i \alpha_i |i\rangle. \tag{1.7}$$

Eine weitere, wichtige Gleichung ist:

$$\langle i|j\rangle = \delta_{ij}, \quad vgl: \quad \int d^3x \ \psi_i(\vec{x})^* \psi_j(\vec{x}) = \delta_{ij}. \tag{1.8}$$

Nimmt man nun $\hat{H}$ zwischen i und j findet man:

$$\underbrace{\langle j|\hat{H}|i\rangle}_{H_{ji} \in \mathbb{C}} = \int d^3x \psi_j(\vec{x})^* \hat{H} \psi_i(\vec{x})$$
$$\sum_i \alpha_i \langle j|\hat{H}|i\rangle = E \sum_i \alpha_i \underbrace{\langle j|i\rangle}_{\delta_{ji}} \tag{1.9}$$
$$\sum_i H_{ji} \alpha_i = E \alpha_j$$

$|i\rangle, i = 1, \ldots, \infty$. Trunkiert: $i = 1, \ldots, N$.

$$\sum_{i=1}^{N} H_{ji} \alpha_i = E \alpha_j, \quad \forall j = 1, \ldots, N. \tag{1.10}$$
$$H\vec{\alpha} = E\vec{\alpha}, \quad H \ldots N \times N - Matrix.$$

Wir haben also ein Eigenwertproblem. Dabei finden wir Eigenvektoren $\vec{\alpha}^{(I)}, I = 1, \ldots, N$ und Eigenwerte $E^{(I)}, I = 1, \ldots, N$.

$$\psi^{(I)}(\vec{x}) = \sum_{n=1}^{N} \alpha_n^{(I)} \psi_n(\vec{x}), \quad E^{(I)}.$$

1.3 Vibrationsspektrum eines Moleküls

Um dieses zu berechnen führen wir Koordinaten $q_i, \quad i = 1, \ldots, f$ ein. Wir setzen an:

$$q_i = \Theta_i^{in\ Schwingung} - \Theta_i^{in\ Ruhe} \tag{1.11}$$

In Ruhelage: $q_i = 0, \quad \forall i = 1, \ldots, f$, dh. q_i entspricht einer Auslenkung.
Als nächstes benötigen wir die potentielle Energie:

$$U(q_1, \ldots, q_f), \quad U(0, \ldots, 0) = 0, \tag{1.12}$$

dh. die Ruhelage ist auf Null gesetzt. Weiters stellt die Ruhelage ein Minimum dar. Wir setzen nun eine Potenzreihenentwicklung um die Ruhelage an:

$$\begin{aligned} U(q_1,\ldots,q_f) &= \underbrace{U(0,\ldots,0)}_{=0} + \underbrace{\sum_{i=1}^{f}(\frac{\partial}{\partial q_i}U)\Big|_{q_i=0} q_i}_{=0,\ weil\ Minimum} + \frac{1}{2}\sum_{i,j=1}^{f}(\frac{\partial}{\partial q_i}\frac{\partial}{\partial q_j}U)\Big|_{q=0} q_i q_j + \ldots \\ &= \frac{1}{2}\sum_{i,j} A_{ij} q_i q_j + \ldots \end{aligned} \tag{1.13}$$

Die kinetische Energie ist:

$$T = \frac{1}{2}\sum_{j,k}^{f} M_{jk}\dot{q_j}\dot{q_k}. \tag{1.14}$$

Dabei ist M_{jk} verallgemeinerte Massenmatrix. Da wir nun die potentielle und die kinetische Energie kennen, schreiben wir die Lagrangefunktion an:

$$\mathcal{L} = T - U = \frac{1}{2}\sum_{i,j=1}^{f}[M_{ij}\dot{q_i}\dot{q_j} - A_{ij}q_i q_j]. \tag{1.15}$$

Ohne Beweis: $A,\ M$ sind symmetrische Matrizen. Wir bilden die Euler-Lagrange-Gleichungen.

$$\frac{\partial\mathcal{L}}{\partial q_k} - \frac{d}{dt}\frac{\partial\mathcal{L}}{\dot{q_k}} = 0 \Rightarrow \sum_{j=1}^{f}[A_{kj}q_j + M_{kj}\ddot{q_j}] = 0, \quad \forall k = 1,\ldots,f. \tag{1.16}$$

Ansatz:

$$\begin{aligned} q_j &= x_j e^{i\omega t} \\ \ddot{q_j} &= -\omega^2 x_j e^{i\omega t}. \end{aligned} \tag{1.17}$$

Einsetzen:

$$\begin{aligned} \sum_{j=1}^{f}[A_{kj} - \omega^2 M_{kj}]x_j e^{i\omega t} &= 0 \quad \forall k = 1\ldots,f \\ \Rightarrow [A - \omega^2 M]\vec{x} &= 0. \end{aligned} \tag{1.18}$$

Wir sind nur an nichttrivialen Lösungen interessiert. Deshalb müssen wir fordern:

$$\begin{aligned} det[A - \omega^2 M] &= 0 \\ \Leftrightarrow det[M(M^{-1}A - \omega^2\mathbb{1})] &= 0 \\ \Leftrightarrow det[M]det[M^{-1}A - \omega^2\mathbb{1}] &= 0, \end{aligned} \tag{1.19}$$

bzw.

$$P_{M^{-1}A}\cdot(\omega^2) = 0. \tag{1.20}$$

Die Nullstellen des charakteristischen Polynoms $P_{M^{-1}A}$ sind die Eigenfrequenzen des Moleküls.

2 Das Gauß'sche Eliminationsverfahren

Wir beginnen mit einem Beispiel.

$$\left.\begin{array}{ccc|cc} 1 & 2 & 3 & 2 & Z_1 \\ 7 & -1 & 4 & 9 & Z_2 \\ 1 & 1 & 1 & 4 & Z_3 \end{array}\right. \tag{2.1}$$

Dabei ist der Block links der Linie eine Matrix M, rechts der Linie finden wir den Vektor b. Dies ist unsere Notation für die geplanten Manipulationen, bei denen wir die Freiheit der Linearkombinationen nutzen, um die Matrix auf Dreiecksform zu bringen. Wir behandeln also ein Gleichungssystem der Form $M.x = b$.
Wir führen dies nun explizit für dieses Beispiel aus:

$$\begin{array}{rcl} Z_2 \leftarrow Z_2 & - & 7 \cdot Z_1 \\ Z_3 \leftarrow Z_3 & - & Z_1 \end{array}$$

$$\left.\begin{array}{ccc|cc} 1 & 2 & 3 & 2 & \tilde{Z}_1 \\ 0 & -15 & -17 & -5 & \tilde{Z}_2 \\ 0 & -1 & -2 & 2 & \tilde{Z}_3 \end{array}\right. \tag{2.2}$$

Wir haben das Problem nun um eine Dimension reduziert. Wir fahren fort:

$$\tilde{Z}_3 \leftarrow \tilde{Z}_3 - \frac{\tilde{Z}_2}{(-15)}$$

$$\left.\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 0 & -15 & -17 & -5 \\ 0 & 0 & -2+\frac{17}{15} & 2+\frac{5}{15} \end{array}\right. \tag{2.3}$$

Wir schreiben dies nocheinmal an:

$$\left.\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 0 & -15 & -17 & -5 \\ 0 & 0 & -\frac{13}{15} & \frac{7}{3} \end{array}\right. \tag{2.4}$$

Nun wechseln wir die Notation und sehen uns das Gleichungssystem an:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 2 \\ -15x_2 - 17x_3 &= -5 \\ -\frac{13}{15}x_3 &= \frac{7}{3} \end{aligned} \tag{2.5}$$

Wir können nun, von unten beginnend, immer in die darüberliegende Zeile einsetzen:

$$\begin{aligned} x_3 = -\frac{7}{13} \cdot \frac{15}{3} &= -\frac{35}{13} \\ \Rightarrow x_2 &= \frac{44}{13} \\ \Rightarrow x_1 &= \frac{43}{13} \end{aligned} \tag{2.6}$$

2.1 Gauß'sche Elimination - naive Implementierung

Wir arbeiten mit folgendem Gleichungssystem:

$$M.x = b \tag{2.7}$$

Dabei treten folgende Objekte auf:

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1L} \\ \vdots & \vdots & \ddots & \vdots \\ m_{L1} & \cdots & \cdots & m_{LL} \end{pmatrix} \tag{2.8}$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_L \end{pmatrix},\ b = \begin{pmatrix} b_1 \\ \vdots \\ b_L \end{pmatrix}. \tag{2.9}$$

Allgemein können wir schreiben:

$$\begin{aligned} Z_1 &\rightsquigarrow Z_1' = Z_1 \\ Z_2 &\rightsquigarrow Z_2' = Z_2 - m_{21}\frac{Z_1}{m_{11}} \\ Z_3 &\rightsquigarrow Z_3' = Z_3 - m_{31}\frac{Z_1}{m_{11}} \\ \vdots &\rightsquigarrow \vdots \\ Z_L &\rightsquigarrow Z_L' = Z_L - m_{L1}\frac{Z_1}{m_{11}} \end{aligned} \tag{2.10}$$

Nach $(L-1)$-fachem Anwenden ist die Matrix M eine obere Dreiecksmatrix:

$$M^{(L-1)} = \begin{pmatrix} \tilde{m_{11}} & \cdots & \cdots & \cdots & \tilde{m_{1L}} \\ 0 & \tilde{m_{22}} & & & \vdots \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \tilde{m_{LL}} \end{pmatrix} \tag{2.11}$$

$$\Rightarrow \tilde{M}.x = \tilde{b} \tag{2.12}$$

Rücksubstitution:

$$x_L = \frac{\tilde{b_L}}{\tilde{m_{LL}}} \tag{2.13}$$

$$\begin{aligned} \tilde{m}_{11}x_1 + \tilde{m}_{12}x_2 + \ldots + \tilde{m}_{1L}x_L &= \tilde{b_1} \\ \tilde{m}_{22}x_2 + \ldots + \tilde{m}_{2L}x_L &= \tilde{b_2} \\ \vdots &= \vdots \\ \tilde{m}_{L-1L-1}x_{L-1} + \tilde{m}_{L-1L}x_L &= \tilde{b}_{L-1} \\ \tilde{m}_{LL}x_L &= \tilde{b}_L \end{aligned} \tag{2.14}$$

$$\Rightarrow x_L = \frac{\tilde{b}_L}{\tilde{m}_{LL}} \tag{2.15}$$

Dies setzen wir nun in die darüberliegende Zeile ein und kommen so rekursiv zu der gesamten Lösung:

$$\begin{aligned} x_{L-1} &= (\tilde{b}_{L-1} - \tilde{m}_{L-1L}\frac{\tilde{b}_L}{\tilde{m}_{LL}})\frac{1}{\tilde{m}_{L-1L-1}} \\ \vdots &= \vdots \\ x_1 &= (\tilde{b}_1 - \sum_{j=2}^{L} \tilde{m}_{1j}x_j)\frac{1}{\tilde{m}_{11}} \end{aligned} \tag{2.16}$$

2.2 Weitere Anwendungen der Gauß'schen Elimination

$$M.x^{(i)} = b^{(i)}, \quad i = 1, \ldots, L_B \tag{2.17}$$

Triviale Verallgemeinerung auf mehrere Gleichungssysteme:

$$M.X = B,\ X = (x^{(1)}, x^{(2)}, \ldots, x^{(L_B)}),\ B = (b^{(1)}, b^{(2)}, \ldots, b^{(L_B)}). \tag{2.18}$$

Sei nun $L_B = L$ und $B = \mathbb{1}$, dh. $M.X = \mathbb{1}$.
Aus der Lösung des Systems mit Gauß'scher Elimination folgt so auch die inverse Matrix:

$$\Rightarrow X = M^{-1} \tag{2.19}$$

Weiters:

- Die Determinante einer Matrix ändert sich nicht wenn zu Zeilen (Spalten) das Vielfache von anderen Zeilen (Spalten) addiert wird

- Die Determinante einer Dreiecksmatrix ist das Produkt ihrer Diagonalelemente, also

$$M \overset{G.E.}{\rightarrow} \tilde{M} \ \Rightarrow det(M) = det(\tilde{M}) = \prod_{i=1}^{L} \tilde{m}_{ii}, \tag{2.20}$$

  das Gauß'sche Eliminationsverfahren liefert also auch die Determinante.

2.3 Pivoting

$$M.x = b$$

$$M = \left(\begin{array}{ccc|c} 0 & 7 & 3 & 1 \\ 1 & 4 & 9 & 2 \\ -12 & 1 & -2 & 3 \end{array}\right) \Leftrightarrow \left(\begin{array}{ccc|c} -12 & 1 & -2 & 3 \\ 1 & 4 & 9 & 2 \\ 0 & 7 & 3 & 1 \end{array}\right) \tag{2.21}$$

Pivoting:
Vertausche Zeilen so, dass die Zeile bei der das größte erste Element auftritt ganz oben steht.
Dabei ist es notwendig im Programm 'Umspeicherungen' vorzunehmen. Dies ist jedoch aufwendig und kann durch eine elegantere Methode vermieden werden: Man verwendet ein Indexfeld P, welches sich merkt welche Zeile mit welcher vertauscht wurde, also letztlich eine Permutation von Zeilennummern beinhaltet. Das tatsächliche Matrixelement wird dann mit $M(P(i), j)$ angesprochen.

$$P = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} \overset{Vertausche\ Z_1,\ Z_3}{\longrightarrow} P = \begin{pmatrix} 3 \\ 2 \\ 1 \\ 4 \\ 5 \\ 6 \end{pmatrix} \tag{2.22}$$

Ansprechen von Zeile 1:

$$M(P(1), j) = M(3, j) \tag{2.23}$$

3 LU-Zerlegung

3.1 LU-Zerlegung

Man spricht von einer LU-Zerlegung einer Matrix A, wenn diese als Produkt

$$A = LU \tag{3.1}$$

geschrieben werden kann. Dabei haben L und U folgende Form:

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{pmatrix}, \; U = \begin{pmatrix} u_{11} & \cdots & \cdots & u_{1n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}. \tag{3.2}$$

Dabei ist L eine lower triangular matrix, U upper triangular matrix, mit der Einschränkung dass die Elemente $u_{ii} \neq 0$ sein müßen.
Eine LU-Zerlegung ist nicht immer möglich. Falls eine Zerlegung möglich ist, so liefert dies oft ein schnelleres, numerisch stabileres Verfahren für lineare Gleichungssysteme, Matrixinversion und Determinantenberechnung als etwa das Gauß'sche Eliminationsverfahren.

z.B.:

$A.x = b, \; A = LU.$
$\Rightarrow L \underbrace{Ux}_{y} = b, \; \Rightarrow$
Schritt 1: $Ly = b, \; \Rightarrow y$
Schritt 2: $Ux = y, \; \Rightarrow x$

In beiden Fällen haben wir Gleichungssysteme mit Dreiecksform. Diese können also durch Rücksubstitution gelöst werden.
Determinante:

$$det[A] = det[LU] = \underbrace{det[L]}_{=1} \cdot det[U] = \prod_{i=1}^{N} u_{ii} \tag{3.3}$$

Eine LU-Zerlegung ist nicht immer möglich. Dazu ein Beispiel:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{21} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{21} \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} \\ u_{11}l_{21} & l_{21}u_{12} + u_{21} \end{pmatrix} \tag{3.4}$$

Durch Vergleichen finden wir: $u_{11} = 0$, und so folgt

$$det[A] = -1 = det[LU] = det[L]det[U] = 0 \tag{3.5}$$

Diese Matrix hat also keine LU-Zerlegung.

Beispiel: Lösung eines Gleichungssystemes mit LU:

$$A = \begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix}; \; b = \begin{pmatrix} 1 \\ 3 \end{pmatrix}; \; Ax = b \tag{3.6}$$

$$A = \begin{pmatrix} 1 & 0 \\ -\frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} 3 & -1 \\ 0 & \frac{5}{3} \end{pmatrix} = LU$$

$$y = Ux \Rightarrow Ly = b$$

$$\begin{pmatrix} 1 & 0 \\ -\frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix} \tag{3.7}$$

$$\Rightarrow y_1 = 1;$$

$$-\frac{1}{3}y_1 + y_2 = 3$$

$$\Rightarrow y_2 = \frac{10}{3}$$

Und weiters:

$$Ux = y$$

$$\begin{pmatrix} 3 & -1 \\ 0 & \frac{5}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{10}{3} \end{pmatrix}$$

$$\Rightarrow x_2 = 2 \tag{3.8}$$

$$3x_1 - x_2 = 1 \Rightarrow x_1 = \frac{3}{3} = 1$$

$$\Rightarrow x = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{3.9}$$

Ohne Beweis betrachten wir folgendes Theorem.

Satz 1. *Sei A eine $(N \times N)$-Matrix. A hat eine LU-Zerlegung* $\Leftrightarrow$

$a_{11} \neq 0,\ det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \neq 0,\ det \begin{pmatrix} a_{11} & \cdots & a_{13} \\ \vdots & & \vdots \\ a_{31} & \cdots & a_{33} \end{pmatrix} \neq 0, \ldots, detA \neq 0.$

Wenn die Zerlegung existiert, so ist sie eindeutig.

Explizite Konstruktion

$$\begin{aligned} u_{(i)} &\quad \ldots \quad i-te\ Zeile\ von\ U \\ l^j &\quad \ldots \quad j-te\ Spalte\ von\ L \\ a_{(i)} &\quad \ldots \quad i-te\ Zeile\ von\ A \\ a^j &\quad \ldots \quad j-te\ Spalte\ von\ A \end{aligned}$$

$$U = \begin{pmatrix} u_{(1)} \\ u_{(2)} \\ \vdots \\ u_{(n)} \end{pmatrix};\ L = (l^1, l^2, \ldots, l^N);\ A = \begin{pmatrix} a_{(1)} \\ a_{(2)} \\ \vdots \\ a_{(n)} \end{pmatrix} = (a^1, a^2, \ldots, a^n). \tag{3.10}$$

$$A = LU$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{N1} & \cdots & l_{N,N-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & \cdots & \cdots & u_{1N} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{NN} \end{pmatrix} \tag{3.11}$$

Zeilen von A:

$$\begin{aligned} a_{(1)} &= u_{(1)} \\ a_{(2)} &= l_{21}u_{(1)} + u_{(2)} \\ \vdots &= \vdots \\ a_{(n)} &= \sum_{k=1}^{n-1} l_{nk}u_{(k)} + u_{(n)} \\ \vdots &= \vdots \\ a_{(N)} &= \sum_{k=1}^{N-1} l_{Nk}u_{(k)} + u_{(N)} \end{aligned} \tag{3.12}$$

Spalten von A:

$$\begin{aligned} a^1 &= u_{11}l^1 \\ a^2 &= u_{12}l^1 + u_{22}l^2 \\ \vdots &= \vdots \\ a^n &= \sum_{k=1}^{n} u_{kn}l^k \\ \vdots &= \vdots \\ a^N &= \sum_{k=1}^{N} u_{kN}l^k \end{aligned} \tag{3.13}$$

Die beiden Gleichungssysteme können nun wechselweise ineinander eingesetzt werden:

$$\begin{aligned} u_{(1)} &= a_{(1)} \\ \Rightarrow l^1 &= \frac{a^1}{u_{11}} \\ u_{(2)} &= a_{(2)} - l_{21}u_{(1)} \\ l^2 &= (a^2 - u_{12}l^1)/u_{22} \\ \vdots &= \vdots \end{aligned} \tag{3.14}$$

In Pseudocode geschrieben sieht das dann so aus:

$$
\begin{aligned}
&for\ i = 1 \to N \\
&\qquad u_{ij} \;=\; a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}; \quad j = 1, \ldots, N \\
&\qquad l_{ji} \;=\; (a_{ji} - \sum_{k=1}^{i-1} u_{ki} l_{jk})/u_{ii}; \quad j = i+1, \ldots, N \\
&endfor
\end{aligned}
$$

3.2 Anwendungen

- Determinantenberechnung
- Gleichungssysteme lösen

Allgemein, zur Lösung des Gleichungssystemes:

$$
\begin{aligned}
Ax = b \Rightarrow Ly = b \;&\to\; y \\
Ux = y \;&\to\; x
\end{aligned}
$$

$$
y_1 = b_1;\ y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j, \quad i = 1, \ldots, N \tag{3.15}
$$

$$
x_N = \frac{y_N}{u_{NN}}, \quad x_i = (y_i - \sum_{j=i+1}^{N} u_{ij} x_j)/u_{ii}.
$$

Spezialfälle:

A hermite'sch, dh. $A^\dagger = A$.

$$
A = LU \Rightarrow A^\dagger = (LU)^\dagger = U^\dagger L^\dagger = A = LU \tag{3.16}
$$

Allerdings kann nicht gefolgert werden: $U^\dagger = L$, $L^\dagger = U$, da i.A. $u_{ii}^\dagger \neq 1$.
Definiere:

$$
\begin{aligned}
D = diag(u_{11}, \ldots, u_{nn}); \quad U &= DU' \\
u'_{ij} &= \frac{u_{ij}}{u_{ii}}
\end{aligned} \tag{3.17}
$$

Dann:

$$
\begin{aligned}
A = LU &= LDU' \\
A^\dagger = (LDU')^\dagger &= U'^\dagger D^\dagger L^\dagger = A = LU.
\end{aligned} \tag{3.18}
$$

Durch Vergleich und aus der Eindeutigkeit der LU-Zerlegung folgt dann, dass $L = U'^\dagger$ und $U = D^\dagger L^\dagger$ ist. Weiters:

$$
\begin{aligned}
A = LU = LD^\dagger L^\dagger; \quad A^\dagger = (LD^\dagger L^\dagger)^\dagger = LDL^\dagger = A = LD^\dagger L^\dagger \\
\Rightarrow D = D^\dagger \Rightarrow u_{ii} \in \mathbb{R}.
\end{aligned} \tag{3.19}
$$

A hermite'sch und positiv definit:
positiv definit: $v^\dagger A v > 0 \quad \forall v$
$v = e_i$ (kartesischer Einheitsvektor) $\Rightarrow e_i^\dagger A e_i = a_{ii} > 0$.

$$\begin{aligned} A &= LDL^\dagger \\ A\ positiv\ definit &\Rightarrow D = diag(u_{11}, \ldots, u_{nn})\ positiv\ definit \\ &\Rightarrow u_{ii} > 0 \end{aligned} \tag{3.20}$$

Weil: $v^\dagger A v = v^\dagger L D L^\dagger v = w^\dagger D w > 0 \quad \forall v$, und $L^\dagger v = w$. Dann gibt es folgende Zerlegung:

$$\begin{aligned} D = PP, \quad P &= diag(\sqrt{u_{11}}, \sqrt{u_{22}}, \ldots, \sqrt{u_{nn}}) \\ A &= LPPL^\dagger = \underbrace{LP}_{T}(LP)^\dagger = TT^\dagger \end{aligned} \tag{3.21}$$

Diese Zerlegung heißt Cholesky Decomposition.

4 Iterative Methoden für Gleichungssysteme

4.1 Einfaches Beispiel

Sei A eine $N \times N$-Matrix der Form $A = \mathbb{1} - B$. Die geometrische Reihe für eine Matrix M:

$$s_n = \sum_{i=0}^{n} M^i, \quad M^0 = \mathbb{1}. \tag{4.1}$$

$$s_{n+1} = \left\{ \begin{array}{c} s_n + M^{n+1} \\ \mathbb{1} + s_n M \end{array} \right\} = s_n + M^{n+1} = \mathbb{1} + s_n M \tag{4.2}$$

$$\begin{aligned} \Leftrightarrow s_n - s_n M &= \mathbb{1} - M^{n+1} \\ s_n(\mathbb{1} - M) &= \mathbb{1} - M^{n+1} \\ s_n &= (\mathbb{1} - M^{n+1})(\mathbb{1} - M)^{-1} \end{aligned} \tag{4.3}$$

$$Falls \lim_{n\to\infty} M^n = 0 \; so \; gilt \lim_{n\to\infty} s_n = (\mathbb{1} - M)^{-1}.$$

$M = UDU^\dagger = UDU^{-1}$, bzw. $M^n = UD^nU^{-1}$ dh. alle Eigenwerte kleiner 1.
$A = \mathbb{1} - B$

Iterative Vorschrift:

$$\begin{aligned} s_0 &= \mathbb{1} \\ s_1 &= \mathbb{1} + B = \mathbb{1} + s_0 B \\ s_2 &= \mathbb{1} + B + B^2 = \mathbb{1} + s_1 B \\ \vdots &= \vdots \\ s_n &= \mathbb{1} + s_{n-1} B, \end{aligned} \tag{4.4}$$

und s_n geht für $n \to \infty$ gegen $(\mathbb{1} - B)^{-1} = A^{-1}$.
Dies ist kein effizienter Algorithmus, stellt aber ein anschauliches Beispiel dar.

4.2 Jacobi-, Gauss-Seidel- und SOR-Methoden

Attraktive Methode für bestimmte Arten von Gleichungssystemen.
Wir beginnen mit:

$$Ax = b \tag{4.5}$$

Zerlegen A in zwei Teile:

$$\begin{aligned} A &= B + A - B \; \Rightarrow \\ Ax &= Bx + (A - B)x = b \\ Bx &= (B - A)x + b \end{aligned} \tag{4.6}$$

Iterativer Ansatz:

$$Bx^{(n+1)} = (B - A)x^{(n)} + b \tag{4.7}$$

Wir beginnen mit einem Startvektor $x^{(0)}$. Die Iterationsgleichung (4.7) erzeugt uns dann $x^{(n)}$ mit $n = 1, 2, \ldots$. Dabei soll $x^{(n)}$ für $n \to \infty$ gleich $x^{(\infty)} \equiv x$ sein, wobei x Lösung von (4.6) ist.

Analyse des verbleibenden Fehlers:

$(4.6) - (4.7) \Rightarrow B(x - x^{(n+1)}) = (B - A)(x - x^{(n)})$ und $\Delta^{(n)} = x - x^{(n)}$ ist der verbleibende Fehler.
Mit $B\Delta^{(n+1)} = (B - A)\Delta^{(n)}$ folgt:

$$\begin{aligned} \Delta^{(n+1)} &= B^{-1}(B - A)\Delta^{(n)} \\ &= (\mathbb{1} - B^{-1}A)\Delta^{(n)} \\ &= (\mathbb{1} - B^{-1}A)(\mathbb{1} - B^{-1}A)\Delta^{(n-1)} \\ &= \ldots \\ \Delta^{(n+1)} &= (\mathbb{1} - B^{-1}A)^{n+1}\Delta^{(0)} \end{aligned} \tag{4.8}$$

Dabei ist $\Delta^{(0)}$ durch die Wahl des Startvektors $x^{(0)}$ fest vorgegeben. $\Delta^{(n+1)}$ konvergiert gegen 0, wenn $(\mathbb{1} - B^{-1}A)^{n+1}$ gegen die 0-Matrix konvergiert.

Satz 2. *Für eine Matrix M gilt $M^n \to 0$ $\Leftrightarrow$ alle Eigenwerte λ von M erfüllen $|\lambda| < 1$.*

Beweis:

Fall 1: M ist diagonalisierbar, dh. $\exists S: \; SMS^{-1} = D = diag(\lambda_1, \lambda_2, \ldots, \lambda_n) \Leftrightarrow M = S^{-1}DS$.
$M^n = S^{-1}D\underbrace{SS^{-1}}_{\mathbb{1}}D\underbrace{SS^{-1}}_{\mathbb{1}}\ldots\underbrace{SS^{-1}}_{\mathbb{1}}DS = S^{-1}D^nS = S^{-1}diag\underbrace{(\lambda_1^n, \lambda_2^n, \ldots, \lambda_n^n)}_{\to 0 \; wenn \; |\lambda_i|<1}S.$

Fall 2: M ist nicht diagonalisierbar. M kann jedoch immer auf Dreiecksform gebracht werden. Eine spezielle Dreiecksform ist die *Jordan'sche Normalform*. $\exists S$: $M = SJS^{-1}$. Dabei ist:

$$J = \begin{pmatrix} J_1 & 0 & \cdots & 0 \\ 0 & J_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & J_m \end{pmatrix}, \quad J_i = \begin{pmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & \cdots & 0 & \lambda_i \end{pmatrix}. \tag{4.9}$$

also J Blockmatrix, λ_i Eigenwert Nummer i und die Dimension der J_i wird durch den Entartungsgrad des jeweiligen Eigenwerts bestimmt.
$M^n = (SJS^{-1})^n = SJ^nS^{-1}$, und $J^n = diag(J_1^n, J_2^n, \ldots, J_m^n)$.

Für 2×2:

$$J_i^2 = \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix} \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix} = \begin{pmatrix} \lambda^2 & 2\lambda \\ 0 & \lambda^2 \end{pmatrix} \tag{4.10}$$

$$\begin{pmatrix} \lambda^2 & 2\lambda \\ 0 & \lambda^2 \end{pmatrix} \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix} = \begin{pmatrix} \lambda^3 & 3\lambda^2 \\ 0 & \lambda^3 \end{pmatrix} \tag{4.11}$$

$$(J_i)^n = \begin{pmatrix} \lambda_i & 1 \\ 0 & \lambda_i \end{pmatrix}^n = \begin{pmatrix} \lambda_i^n & n\lambda_i^{n-1} \\ 0 & \lambda_i^n \end{pmatrix} \to 0, \ |\lambda| < 1. \tag{4.12}$$

Für 3×3:

$$(J_i)^n = \begin{pmatrix} \lambda_i^n & n\lambda_i^{n-1} & \frac{n(n-1)}{2}\lambda_i^{n-2} \\ 0 & \lambda_i^n & n\lambda_i^{n-1} \\ 0 & 0 & \lambda_i^n \end{pmatrix}, \quad etc. \tag{4.13}$$

Dies kann durch Induktion gezeigt werden. q.e.d.

Die Iteration $Bx^{(n+1)} = (B - A)x^{(n)} + b$ konvergiert gegen Lösung x von $Ax = b$ wenn alle Eigenwerte von $\mathbb{1} - B^{-1}A$ kleiner als 1 sind. $x^{(0)}$ kann im Prinzip beliebig gewählt werden. Die Konvergenzrate ist durch den größten Eigenwert bestimmt.

Jacobi-Methode: $B = diag(a_{11}, a_{22}, \ldots, a_{NN})$. $A = diagA + Rest$.

$$\begin{aligned} Dx^{(n+1)} &= (D - A)x^{(n)} + b \\ x^{(n+1)} &= D^{-1}(D - A)x^{(n)} + D^{-1}b \\ x^{(n+1)} &= (\mathbb{1} - D^{-1}A)x^{(n)} + D^{-1}b \\ x_i^{(n+1)} &= x_i^{(n)} + \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{N} a_{ij}x_j^{(n)}), \quad i = 1, \ldots, N \\ x_i^{(n+1)} &= \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij}x_j^{(n)}) \end{aligned} \tag{4.14}$$

Ohne Beweis wollen wir folgenden Satz festhalten.

Satz 3. *Wenn $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ gilt, dann ist die Lösung konvergent.*

Gauss-Seidel-Methode:

$A = S + D + T$ mit $D = diag(a_{11}, a_{22}, \ldots, a_{NN})$, sowie

$$S = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ a_{21} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{pmatrix}, \quad T = \begin{pmatrix} 0 & a_{21} & \cdots & a_{1N} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & a_{N-1,N} \\ 0 & \cdots & \cdots & 0 \end{pmatrix}. \tag{4.15}$$

$B = S + D$. Wir finden dann:

$$\begin{aligned} Bx^{(n+1)} &= (B-A)x^{(n)} + b \\ (S+D)x^{(n+1)} &= -Tx^{(n)} + b \\ Dx^{(n+1)} &= b - Sx^{(n+1)} - Tx^{(n)} \\ x^{(n+1)} &= D^{-1}(b - Sx^{(n+1)} - Tx^{(n)}) \Leftrightarrow \\ x_i^{(n+1)} &= \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{(n+1)} - \sum_{j>i} a_{ij}x_j^{(n)}) \end{aligned} \tag{4.16}$$

Dabei benötigt $x_1^{(n+1)}$ kein $x_j^{(n+1)}$, $x_2^{(n+1)}$ benötigt genau das bis dahin bekannte $x_1^{(n+1)}$, und so weiter, bis schließlich $x_N^{(n+1)}$ die $x_j^{(n+1)}$ mit $j = 1, \ldots, N-1$ benötigt.

Sukzessive Überrelaxation (SOR-Successive Over Relaxation):

$B = S + \frac{D}{\omega}$, mit ω Streckungsfaktor, $\omega \in \mathbb{R}$ und $0 < \omega < 2$, da sonst nicht konvergent. Für den Fall $\omega = 1$ bekommen wir Gauss-Seidel, wählen wir $\omega > 1$ so konvergiert das Verfahren schneller als Gauss-Seidel. Wir setzen ein:

$$\begin{aligned} (S + \frac{1}{\omega}D)x^{(n+1)} &= (\frac{D}{\omega} - D - T)x^{(n)} + b \\ \Rightarrow x^{(n+1)} &= \omega D^{-1}(b - Sx^{(n+1)} - Tx^{(n)}) + (1-\omega)x^n \\ \Leftrightarrow x_i^{(n+1)} &= \frac{\omega}{a_{ii}}[b_i - \sum_{j<i} a_{ij}x_j^{(n+1)} - \sum_{j>i} a_{ij}x_j^{(n)}] + (1-\omega)x_i^{(n)} \end{aligned} \tag{4.17}$$

Das Verfahren konvergiert immer wenn A positiv definit ist.
Abbruchkriterium:

- Mittlere Genauigkeit ($||z|| = \sqrt{\sum_{i=1}^{N} z_i^2}$):

$$||x^{(n+1)} - x^{(n)}|| < \epsilon$$

- Mindestgenauigkeit an jedem Punkt:

$$\max_i |x_i^{(n+1)} - x_i^{(n)}| < \epsilon$$

4.3 Conjugate Gradient Methoden

Aufgabenstellung:

$$A\vec{x} = \vec{b} \tag{4.18}$$

Einfachster Fall:

A symmetrisch und positiv definit, dh.
$A^T = A, A_{ij} = A_{ji},\ \vec{x}^\dagger A\vec{v} \geq 0\ \forall \vec{v}$.

Definieren: $f(\vec{x}) = f(x_1, x_2, \ldots, x_n) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b} \cdot \vec{x}$.

Dabei ist der Ausdruck auf der rechten Seite eine Zahl.
$\vec{x}_0$ ist das Minimum von $f(\vec{x}) \Leftrightarrow \vec{\nabla} f(\vec{x}_0) = 0$.

$$\begin{aligned}
\vec{\nabla} f(\vec{x}) &= \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \vdots \\ \frac{\partial}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \vdots \\ \frac{\partial}{\partial x_n} \end{pmatrix} [\frac{1}{2} \sum_{i,j} A_{ij} x_i x_j - \sum_i b_i x_i] && (4.19) \\
\frac{\partial}{\partial x_1} f(\vec{x}) &= [A_{11} x_1 + \frac{1}{2} A_{12} x_2 + \frac{1}{2} A_{21} x_2 + \frac{1}{2} A_{13} x_3 + \frac{1}{2} A_{31} x_3 + \cdots - b_1] && (4.20) \\
&= \sum_j A_{1j} x_j - b_1,
\end{aligned}$$

wobei wir im letzten Schritte die Symmetrie von A, dh. $A_{ij} = A_{ji}$ ausgenutzt haben.

Allgemein:

$$\begin{aligned}
\frac{\partial}{\partial x_k} f(\vec{x}) &= \sum_{j=1}^{N} A_{kj} x_j - b_k \; \forall k && (4.21) \\
\vec{\nabla} f(\vec{x}) &= A\vec{x} - \vec{b} && (4.22) \\
\vec{\nabla} f(\vec{x}_0) &= 0 \Leftrightarrow A\vec{x}_0 - \vec{b} = 0, && (4.23)
\end{aligned}$$

dh. das Minimum $\vec{x}_0$ löst die Gleichung $A\vec{x} = \vec{b}$.

Komplexe Matrizen:

$$M = A + iC \tag{4.24}$$

wobei M die Matrix der $\Re e$-Teile und C die Matrix der $\Im m$-Teile ist.

$$\begin{aligned}
M\vec{x} &= \vec{b} \\
\Leftrightarrow (A + iC) \underbrace{(\vec{x} + i\vec{y})}_{\vec{\tilde{x}}} &= \underbrace{\vec{b} + i\vec{d}}_{\vec{\tilde{b}}} && (4.25)
\end{aligned}$$

Wir haben dann folgendes Gleichungssystem:

$$\begin{aligned}
A\vec{x} - C\vec{y} &= \vec{b} \quad (\Re e) \\
C\vec{x} + A\vec{y} &= \vec{d} \quad (\Im m)
\end{aligned} \tag{4.26}$$

Wenn A und C dabei jeweils eine $N \times N$-Matrix ist, so finden wir nun eine $2N \times 2N$-Matrix Q:

$$\underbrace{\begin{pmatrix} A & -C \\ C & A \end{pmatrix}}_{Q} \begin{pmatrix} \vec{x} \\ \vec{y} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{d} \end{pmatrix} \tag{4.27}$$

Wenn nun $H = A + iC$ eine hermite'sche Matrix ist, dh. falls gilt $H = H^\dagger$:

$$\begin{aligned}
H^\dagger = (A + iC)^\dagger &= A^T - iC^T = H = A + iC && (4.28) \\
\Leftrightarrow A^T = A, \quad C^T &= -C.
\end{aligned}$$

In diesem Falle finden wir also für A eine symmetrische, für C eine antisymmetrische Matrix vor. Ferner ist:

$$Q^T = \begin{pmatrix} A^T & C^T \\ -C^T & A^T \end{pmatrix} = \begin{pmatrix} A & -C \\ C & A \end{pmatrix} = Q. \tag{4.29}$$

Wir finden also: Ist H hermite'sch $\Leftrightarrow$ Q symmetrisch. H positiv definit und hermite'sch:
Vorraussetzung für conjugate gradient Methode gegeben.

Vorgehensweise beim Suchen des Minimums

- Es wird in diskreten Schritten vorgegangen: $n = 1, 2, \ldots$
- Ein Startvektor $\vec{x}^{(1)}$ wird vorgegeben
- Rekursiv werden erzeugt: Vektoren $\vec{x}^{(n)}$, Suchrichtungen $\vec{p}^{(n)}$, sowie Residuumsvektoren $\vec{r}^{(n)} = \vec{b} - A\vec{x}^{(n)}$, welche der Entfernung zur Lösung entsprechen
- Der Algorithmus bricht ab, wenn $|\vec{r}^{(n)}| < \epsilon$ ist
- $\vec{x}^{(n)} \in \vec{x}^{(1)} + span\{r^{(1)}, Ar^{(1)}, A^2r^{(1)}, \ldots, A^{n-1}r^{(1)}\}$ Krylov-Raum

Der Algorithmus ist von folgender Gestalt:

- Gib einen Startvektor $\vec{x}^{(1)}$ vor
- $\vec{r}^{(1)} = \vec{b} - A\vec{x}^{(1)}, \quad \vec{p}^{(1)} = \vec{r}^{(1)}$
- Rekursionsgleichungen:

$$\begin{aligned} \alpha_n = \frac{\vec{r}^{(n)^T}\vec{r}^{(n)}}{\vec{p}^{(n)^T}A\vec{p}^{(n)}} \quad &, \qquad \vec{r}^{(n+1)} = \vec{r}^{(n)} - \alpha_n A\vec{p}^{(n)} \\ \beta_n = \frac{\vec{r}^{(n+1)^T}\vec{r}^{(n+1)}}{\vec{r}^{(n)^T}\vec{r}^{(n)}} \quad &, \qquad \vec{p}^{(n+1)} = \vec{r}^{(n+1)} + \beta_n\vec{p}^{(n)} \\ \vec{x}^{(n+1)} &= \vec{x}^{(n)} + \alpha_n\vec{p}^{(n)} \end{aligned} \tag{4.30}$$

- Stop, wenn $|\vec{r}^{(n)}| < \epsilon$

Bemerkungen und Analyse:

Mittels Induktion kann gezeigt werden:

$$\begin{aligned} \vec{r}^{(i)^T}\vec{r}^{(j)} &= 0, \quad \forall i \neq j \\ \vec{p}^{(i)^T}A\vec{p}^{(j)} &= 0, \quad \forall i \neq j \\ \vec{r}^{(i)^T}\vec{p}^{(j)} &= 0, \quad \forall i \neq j \end{aligned} \tag{4.31}$$

Die Suchrichtungen sind orthogonal zu allen alten Residuumsvektoren. Man kann zeigen, dass α_n so gewählt ist, dass $f(\vec{x}^{(n)} + \alpha_n\vec{p}^{(n)})$ entlang des Strahles (der Geraden) $\vec{x}^{(n)} + \alpha_n\vec{p}^{(n)}$ minimal ist. Dies führt zu einer sukzessiven Minimierung in

allen Richtungen. Ein exaktes Ergebnis wird nach maximal N Schritten erhalten. Eine Verallgemeinerung auf komplexe, nichtnotwendigerweise symmetrische (oder hermite'sche) Matrizen ist möglich, allerdings geht dann die Interpretation der Minimierung verloren.

Preconditioner:

$A\vec{x} = \vec{b}$, A hat Eigenwerte λ_i. Die Konvergenzgeschwindigkeit hängt von der spektralen Konditionszahl $s = \frac{\lambda_{max}}{\lambda_{min}}$ ab. Dabei ist $s >> 1$ schlecht.
Das System $M^{-1}A\vec{x} = M^{-1}\vec{b}$ hat die gleiche Lösung $\vec{x}$. Ist M geeignet gebaut, und auch leicht invertierbar, so hat $M^{-1}A$ bessere spektrale Eigenschaften. Dies ist etwa gegeben, wenn M ähnlich kleine Eigenwerte hat wie A. M nennt man dann Preconditioner.

4.4 Einschub: Beschreibung von H_3^+ mit dem Hubbard Modell

4.4.1 Problemstellung, zweite Quantisierung

Wir wollen hier das Hubbard Modell auf eine (vereinfachte) Beschreibung des H_3^+ anwenden. Das Ion wird dabei durch drei an den Ecken eines gleichseitigen Dreieckes angeordnete Positionen (=Atomrümpfe) modelliert. Insgesamt gibt es 2 Elektronen die sich auf die drei Plätze verteilen können. Jedes Elektron hat zwei mögliche Spineinstellungen. Die folgende Abbildung zeigt die Anordnung im Dreieck. In zweiter

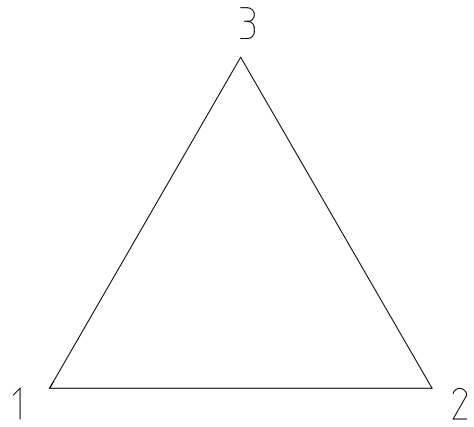


Abbildung 2: Anordnung der Atomrümpfe als Eckpunkte eines gleichseitigen Dreieckes

Quantisierung finden wir folgende Struktur des Hamiltons.

$$\hat{H} = -t \sum_{\langle i,j \rangle} [a_i^{\uparrow\dagger} a_j^{\uparrow} + a_i^{\downarrow\dagger} a_j^{\downarrow}] + U \sum_i n_i^{\uparrow} n_i^{\downarrow} + B \sum_i n_i^{\uparrow}; \quad t, U, B \in \mathbb{R}. \tag{4.32}$$

Dabei sind die Anzahloperatoren n gegeben durch:

$$n_i^{\uparrow} = a_i^{\uparrow\dagger} a_i^{\uparrow}, \quad n_i^{\downarrow} = a_i^{\downarrow\dagger} a_i^{\downarrow} \tag{4.33}$$

Ferner sind die $a_i^{\alpha\dagger}$ Erzeugeroperatoren für ein Elektron am Platz i mit Spin $\alpha \in \{\uparrow, \downarrow\}$ und die a_i^{α} Vernichteroperatoren für ein Elektron am Platz i mit Spin $\alpha \in \{\uparrow, \downarrow\}$. Diese Operatoren genügen Vertauschungsrelationen:

$$\begin{aligned} a_i^{\alpha\dagger} a_j^{\beta\dagger} + a_j^{\beta\dagger} a_i^{\alpha\dagger} &= 0 \ \forall i, j, \alpha, \beta \ \Rightarrow a_i^{\alpha\dagger} a_i^{\alpha\dagger} = 0 \\ a_i^{\alpha} a_j^{\beta} + a_j^{\beta} a_i^{\alpha} &= 0 \ \forall i, j, \alpha, \beta \ \Rightarrow a_i^{\alpha} a_i^{\alpha} = 0, \ Pauli - Prinzip \\ a_i^{\alpha\dagger} a_j^{\beta} + a_j^{\beta} a_i^{\alpha\dagger} &= \delta_{ij} \delta_{\alpha\beta} \end{aligned} \tag{4.34}$$

In diesem System gibt es einen ausgezeichneten Zustand, den Vakuumzustand. Wir notieren diesen mit $|0\rangle$. Ferner gilt für den Vakuumzustand:

$$\langle 0|0\rangle = 1, \ a_i^{\alpha}|0\rangle = 0 \tag{4.35}$$

Die Anwendung eines Vernichters auf das Vakkum liefert also 0, was so interpretiert werden kann, dass ein Teilchen, welches gar nicht vorhanden ist, nicht vernichtet werden kann. Wir wollen uns nun ansehen, wie der Hamiltonoperator unseres konkreten Systemes aussieht, dh. wir schreiben die Summenbeiträge aus. Wir finden

also einen Operator folgender Struktur vor:

$$\begin{aligned}\hat{H} = & - t[a_1^{\uparrow\dagger}a_2^{\uparrow} + a_2^{\uparrow\dagger}a_3^{\uparrow} + a_3^{\uparrow\dagger}a_1^{\uparrow} + a_1^{\downarrow\dagger}a_2^{\downarrow} + a_2^{\downarrow\dagger}a_3^{\downarrow} + a_3^{\downarrow\dagger}a_1^{\downarrow}] - \\ & - t[a_2^{\uparrow\dagger}a_1^{\uparrow} + a_3^{\uparrow\dagger}a_2^{\uparrow} + a_1^{\uparrow\dagger}a_3^{\uparrow} + a_2^{\downarrow\dagger}a_1^{\downarrow} + a_3^{\downarrow\dagger}a_2^{\downarrow} + a_1^{\downarrow\dagger}a_3^{\downarrow}] + \\ & + U[n_1^{\uparrow}n_1^{\downarrow} + n_2^{\uparrow}n_2^{\downarrow} + n_3^{\uparrow}n_3^{\downarrow}] + \\ & + B[n_1^{\uparrow} + n_2^{\uparrow} + n_3^{\uparrow}]\end{aligned} \tag{4.36}$$

15 mögliche Basiszustände:

Es gilt: $|\Phi\rangle = \langle\Phi|^{\dagger}$.

$$\begin{array}{llll} Kets: & & Bras: & \\ |\Phi_1\rangle &= a_1^{\uparrow\dagger}a_1^{\downarrow\dagger}|0\rangle & \langle\Phi_1| &= \langle 0|a_1^{\downarrow}a_1^{\uparrow} \\ |\Phi_2\rangle &= a_2^{\uparrow\dagger}a_2^{\downarrow\dagger}|0\rangle & \langle\Phi_2| &= \langle 0|a_2^{\downarrow}a_2^{\uparrow} \\ |\Phi_3\rangle &= a_3^{\uparrow\dagger}a_3^{\downarrow\dagger}|0\rangle & \langle\Phi_3| &= \langle 0|a_3^{\downarrow}a_3^{\uparrow} \\ |\Phi_4\rangle &= a_1^{\uparrow\dagger}a_2^{\downarrow\dagger}|0\rangle & \langle\Phi_4| &= \langle 0|a_2^{\downarrow}a_1^{\uparrow} \\ |\Phi_5\rangle &= a_1^{\uparrow\dagger}a_3^{\downarrow\dagger}|0\rangle & \langle\Phi_5| &= \langle 0|a_3^{\downarrow}a_1^{\uparrow} \\ |\Phi_6\rangle &= a_2^{\uparrow\dagger}a_3^{\downarrow\dagger}|0\rangle & \langle\Phi_6| &= \langle 0|a_3^{\downarrow}a_2^{\uparrow} \\ |\Phi_7\rangle &= a_1^{\downarrow\dagger}a_2^{\uparrow\dagger}|0\rangle & \langle\Phi_7| &= \langle 0|a_2^{\uparrow}a_1^{\downarrow} \\ |\Phi_8\rangle &= a_1^{\downarrow\dagger}a_3^{\uparrow\dagger}|0\rangle & \langle\Phi_8| &= \langle 0|a_3^{\uparrow}a_1^{\downarrow} \\ |\Phi_9\rangle &= a_2^{\downarrow\dagger}a_3^{\uparrow\dagger}|0\rangle & \langle\Phi_9| &= \langle 0|a_3^{\uparrow}a_2^{\downarrow} \\ |\Phi_{10}\rangle &= a_1^{\uparrow\dagger}a_2^{\uparrow\dagger}|0\rangle & \langle\Phi_{10}| &= \langle 0|a_2^{\uparrow}a_1^{\uparrow} \\ |\Phi_{11}\rangle &= a_1^{\uparrow\dagger}a_3^{\uparrow\dagger}|0\rangle & \langle\Phi_{11}| &= \langle 0|a_3^{\uparrow}a_1^{\uparrow} \\ |\Phi_{12}\rangle &= a_2^{\uparrow\dagger}a_3^{\uparrow\dagger}|0\rangle & \langle\Phi_{12}| &= \langle 0|a_3^{\uparrow}a_2^{\uparrow} \\ |\Phi_{13}\rangle &= a_1^{\downarrow\dagger}a_2^{\downarrow\dagger}|0\rangle & \langle\Phi_{13}| &= \langle 0|a_2^{\downarrow}a_1^{\downarrow} \\ |\Phi_{14}\rangle &= a_1^{\downarrow\dagger}a_3^{\downarrow\dagger}|0\rangle & \langle\Phi_{14}| &= \langle 0|a_3^{\downarrow}a_1^{\downarrow} \\ |\Phi_{15}\rangle &= a_2^{\downarrow\dagger}a_3^{\downarrow\dagger}|0\rangle & \langle\Phi_{15}| &= \langle 0|a_3^{\downarrow}a_2^{\downarrow} \end{array} \tag{4.37}$$

Die Basiszustände sind normiert und orthogonal: Für $(i,\alpha) \neq (j,\beta)$:
Norm:

$$\begin{aligned}\langle\Phi|\Phi\rangle &= \langle 0|a_j^{\beta}\ \underbrace{a_i^{\alpha}a_i^{\alpha\dagger}}_{=1-a_i^{\alpha\dagger}a_i^{\alpha}}\ a_j^{\beta\dagger}|0\rangle \\ &= \langle 0|\ \underbrace{a_j^{\beta}a_j^{\beta\dagger}}_{=1-a_j^{\beta\dagger}a_j^{\beta}}\ |0\rangle - \langle 0|a_j^{\beta}a_j^{\alpha\dagger}\ \underbrace{a_i^{\alpha}a_j^{\beta\dagger}}_{=-a_j^{\beta\dagger}a_i^{\alpha}}\ |0\rangle \\ &= \langle 0|0\rangle - \langle 0|a_j^{\beta\dagger}\underbrace{a_j^{\beta}|0\rangle}_{=0} + \langle 0|a_j^{\beta}a_i^{\alpha\dagger}a_j^{\beta\dagger}\underbrace{a_i^{\alpha}|0\rangle}_{=0} = 1\ \checkmark\end{aligned} \tag{4.38}$$

Orthogonalität:
Bei $\langle\Phi'|\Phi\rangle$ gilt $(i,j,\alpha,\beta) \neq (i',j',\alpha',\beta')$. Daher wird nicht wie bei der Norm zweimal die 1 erzeugt. Es bleiben ein oder zwei Vernichter über und daher

$$\langle\Phi'|\Phi\rangle = 0 \tag{4.39}$$

4.4.2 Wirkung des Anzahloperators $n_i^\alpha = a_i^{\alpha\dagger} a_i^\alpha$

Sehen wir uns nun die Wirkung des Anzahloperators auf einen Basiszustand an:

$$\begin{aligned} n_i^\alpha a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle &= a_i^{\alpha\dagger} a_i^\alpha a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle = a_j^{\beta\dagger} a_i^{\alpha\dagger} \underbrace{a_i^\alpha a_i^{\alpha\dagger}}_{=1-a_i^{\alpha\dagger}a_i^\alpha} |0\rangle \\ &= a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle - a_j^{\beta\dagger} a_i^{\alpha\dagger} a_i^{\alpha\dagger} \underbrace{a_i^\alpha|0\rangle}_{=0} \end{aligned} \tag{4.40}$$

Ganz analog behandeln wir die anderen Fälle:

$$\begin{aligned} n_i^\alpha a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle &= a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle \\ n_j^\beta a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle &= a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle \\ n_k^\gamma a_j^{\beta\dagger} a_i^{\alpha\dagger}|0\rangle &= 0,\ (k,\gamma) \neq (j,\beta)\ \textit{oder}\ (i,\alpha),\ \textit{dh. Quant}\ (k,\gamma)\ \textit{unbesetzt} \end{aligned} \tag{4.41}$$

4.4.3 Wirkung der kinetischen Terme

Die kinetischen Terme bewirken ein 'Hüpfen' der Elektronen im Gitter. Etwa springt nun ein Elektron mit Spin $\uparrow$ von 2 nach 1:

$$\begin{aligned} & a_1^{\uparrow\dagger} a_2^\uparrow|\Phi_2\rangle,\ |\Phi_2\rangle = a_2^{\uparrow\dagger} a_2^{\downarrow\dagger}|0\rangle \\ \Rightarrow a_1^{\uparrow\dagger} a_2^\uparrow|\Phi_2\rangle = a_1^{\uparrow\dagger} \underbrace{a_2^\uparrow a_2^{\uparrow\dagger}}_{=1-a_2^{\uparrow\dagger}a_2^\uparrow} a_2^{\downarrow\dagger}|0\rangle &= a_1^{\uparrow\dagger} a_2^{\downarrow\dagger}|0\rangle - a_1^{\uparrow\dagger} a_2^{\uparrow\dagger} \underbrace{a_2^\uparrow a_2^{\downarrow\dagger}}_{=-a_2^{\downarrow\dagger}a_2^\uparrow} |0\rangle \\ = a_1^{\uparrow\dagger} a_2^{\downarrow\dagger}|0\rangle + a_1^{\uparrow\dagger} a_2^{\uparrow\dagger} a_2^{\downarrow\dagger} \underbrace{a_2^\uparrow|0\rangle}_{=0} &= a_1^{\uparrow\dagger} a_2^{\downarrow\dagger}|0\rangle = |\Phi_4\rangle \end{aligned} \tag{4.42}$$

Ein weiteres Beispiel:

$$\begin{aligned} & a_1^{\uparrow\dagger} a_2^\uparrow|\Phi_7\rangle,\ |\Phi_7\rangle = a_1^{\downarrow\dagger} a_2^{\uparrow\dagger}|0\rangle \\ \Rightarrow a_1^{\uparrow\dagger} a_2^\uparrow|\Phi_7\rangle = a_1^{\uparrow\dagger} \underbrace{a_2^\uparrow a_1^{\downarrow\dagger}}_{=-a_1^{\downarrow\dagger}a_2^\uparrow} a_2^{\uparrow\dagger}|0\rangle &= -a_1^{\uparrow\dagger} a_1^{\downarrow\dagger} \underbrace{a_2^\uparrow a_2^{\uparrow\dagger}}_{=1-a_2^{\uparrow\dagger}a_2^\uparrow} |0\rangle \\ = -a_1^{\uparrow\dagger} a_1^{\downarrow\dagger}|0\rangle + a_1^{\uparrow\dagger} a_1^{\downarrow\dagger} a_2^{\uparrow\dagger} \underbrace{a_2^\uparrow|0\rangle}_{=0} &= -|\Phi_1\rangle \end{aligned} \tag{4.43}$$

Und noch eines zum Abschluss:

$$\begin{aligned} & a_1^{\uparrow\dagger} a_2^\uparrow|\Phi_{10}\rangle,\ |\Phi_{10}\rangle = a_1^{\uparrow\dagger} a_2^{\uparrow\dagger}|0\rangle \\ \Rightarrow a_1^{\uparrow\dagger} a_2^\uparrow|\Phi_{10}\rangle = a_1^{\uparrow\dagger} \underbrace{a_2^\uparrow a_1^{\uparrow\dagger}}_{=-a_1^{\uparrow\dagger}a_2^\uparrow} a_2^{\uparrow\dagger}|0\rangle &= -\underbrace{a_1^{\uparrow\dagger} a_1^{\uparrow\dagger}}_{=0} a_2^\uparrow a_2^{\uparrow\dagger}|0\rangle = 0 \end{aligned} \tag{4.44}$$

4.4.4 Matrixdarstellung des Hamiltonoperators

Mit den Rechenregeln die wir eben erarbeitet haben können wir nun die Matrixdarstellung des Hamiltonoperators bestimmen, was gleichzeitig auch Inhalt und Ziel des zweiten Projektes sein soll.

$$H_{ij} = \langle\Phi_i|\hat{H}|\Phi_j\rangle, \quad i,j = 1,2,\ldots,15 \tag{4.45}$$

wobei H dann eine 15×15-Matrix ist.
Darstellung von H:

$$\begin{aligned} \langle \Phi_i | \Phi_j \rangle &= \delta_{ij}, \quad \mathbb{1} = \sum_{i=1}^{15} |\Phi_i\rangle\langle\Phi_i| \\ \Rightarrow \hat{H} = \mathbb{1}\hat{H}\mathbb{1} &= \sum_{i,j=1}^{15} |\Phi_i\rangle \underbrace{\langle \Phi_i | \hat{H} | \Phi_j \rangle}_{=H_{ij}} \langle \Phi_j | \\ \Rightarrow \hat{H} &= \sum_{i,j=1}^{15} |\Phi_i\rangle H_{ij} \langle \Phi_j| \end{aligned} \tag{4.46}$$

Die Eigenwertgleichung für die Matrix H lautet:

$$H\vec{v}^{(n)} = E^{(n)}\vec{v}^{(n)} \tag{4.47}$$

wobei $E^{(n)}$ die Eigenwerte von H und $\vec{v}^{(n)}$ die Eigenvektoren von H sind und $n = 1, 2, \ldots, 15$.
Wir definieren:

$$|n\rangle = \sum_{l=1}^{15} |\Phi_l\rangle v_l^{(n)} \tag{4.48}$$

mit $v_l^{(n)}$ der l-ten Komponente von $\vec{v}^{(n)}$.
Dann:

$$\begin{aligned} \hat{H}|n\rangle &= \sum_{i,j} |\Phi_i\rangle H_{ij} \langle \Phi_j | \sum_l |\Phi_l\rangle v_l^{(n)} \\ &= \sum_{i,j,l} |\Phi_i\rangle H_{ij} \underbrace{\langle \Phi_j | \Phi_l \rangle}_{=\delta_{il}} v_l^{(n)} \\ &= \sum_{i,j} |\Phi_i\rangle H_{ij} v_j^{(n)} = \sum_i |\Phi_i\rangle \underbrace{\sum_j H_{ij} v_j^{(n)}}_{=v_i^{(n)} E^{(n)}} \\ &= E^{(n)} \sum_i |\Phi_i\rangle v_i^{(n)} = E^{(n)}|n\rangle \end{aligned} \tag{4.49}$$

Wir identifizieren also $|n\rangle$ als Eigenzustände von $\hat{H}$, sowie $E^{(n)}$ als Energieeigenwerte von $\hat{H}$.

5 Eigenwertprobleme

5.1 Allgemeine Bemerkungen

$$A\vec{x}^{(i)} = \lambda^{(i)}\vec{x}^{(i)}, \quad i = 1, \ldots, N, \; A \ldots N \times N \; Matrix \tag{5.1}$$

Im Prinzip:

$$P(\lambda) = det[\lambda \mathbb{1} - A] \tag{5.2}$$

Ausrechnen, numerische Nullstellensuche $\Rightarrow$ aufwendig, keine Eigenvektoren. Ohne Beweis notieren wir folgenden Satz.

Satz 4. *Es ist nicht möglich die Nullstellen eines allgemeinen Polynomes in endlich vielen Schritten zu finden.*
Zu jedem Polynom $P(\lambda)$ kann man eine Matrix A konstruieren, so dass $P(\lambda) = det[\lambda \mathbb{1} - A]$.

Diese Darstellung ist nicht eindeutig, da die Determinante invariant unter Ähnlichkeitstransformationen ist.
Das Eigenwertproblem kann im allgemeinen Falle nicht in endlich vielen Schritten gelöst werden. Daher sind iterative Methoden von Nöten.

Zentrales Werkzeug:

Ähnlichkeitstransformationen der Form $Q^{-1}AQ = D$, wobei $D = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$.
Wird ein Q dieser Form gefunden, so ist das Eigenwertproblem gelöst.
Weil: Q von links:
$AQ = QD = Q \cdot diag(\lambda_1, \lambda_2, \ldots, \lambda_N)$.

$$\begin{aligned} Q = (\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(N)}) \quad &\Rightarrow \\ A(\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(n)}) \quad &= \quad (\lambda_1\vec{v}^{(1)}, \lambda_2\vec{v}^{(2)}, \ldots, \lambda_N\vec{v}^{(N)}) \end{aligned} \tag{5.3}$$

sind alle Gleichungen des Eigenwertproblemes. Die Spalten von Q sind dabei die Eigenvektoren. Die Idee ist nun die Matrizen Q iterativ aufzubauen:

$$Q = Q_1 Q_2 \ldots Q_N$$

$$\lim_{k\to\infty} Q_k^{-1} Q_{k-1}^{-1} \cdots Q_2^{-1} Q_1^{-1} A Q_1 Q_2 \cdots Q_{k-1} Q_k = diag(\lambda_1, \lambda_2, \ldots, \lambda_N) \tag{5.4}$$

5.2 Die Jacobi Methode

Sei A reell und symmetrisch (so ist auch das Eigenwertproblem für hermite'sche, komplexe Matrizen gelöst).

Wir definieren Matrizen $Q^{(p,q)}$:

$$Q^{(p,q)} = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & c & & & & s & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & -s & & & & c & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \leftarrow p-te\ Zeile \\ \\ \\ \leftarrow q-te\ Zeile \\ \\ \\ \\ \end{matrix} \tag{5.5}$$

$$\begin{matrix} & & \uparrow & & \uparrow & & \\ & & p-te & & q-te & & \\ & & Spalte & & Spalte & & \end{matrix}$$

Für die Einträge der Matrix gilt dann:

$$(Q^{(p,q)})_{ij} = \begin{cases} 1 & für\ i = j \neq p\ oder\ q \\ c & für\ i = j = p\ oder\ q \\ s & für\ i = p,\ j = q \\ -s & für\ i = 1,\ j = p \\ 0 & sonst \end{cases} \tag{5.6}$$

Ferner sind $c, s \in \mathbb{R}$ und es gilt:

$$c^2 + s^2 = 1 \tag{5.7}$$

Die transponierte Matrix ${Q^{(p,q)}}^T$ ist dann von folgender Gestalt:

$${Q^{(p,q)}}^T = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & c & & & & -s & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & s & & & & c & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix} = {Q^{(p,q)}}^{-1} \tag{5.8}$$

<u>Definieren:</u>

$$A' = {Q^{(p,q)}}^T A Q^{(p,q)} \tag{5.9}$$

$$A'^T = {Q^{(p,q)}}^T A^T Q^{(p,q)} = {Q^{(p,q)}}^T A Q^{(p,q)} = A' \tag{5.10}$$

Eine Ähnlichkeitstransformation erhält also die Symmetrie von A. Auch A' ist symmetrisch und reell.

Ausmultiplikation:

$\tilde{A} = AQ^{(p,q)}$: Nur die p-te und q-te Spalte werden geändert:

$$\begin{aligned} \tilde{a}_{ij} &= a_{ij} \; für \; j \neq p \; oder \; q \\ \tilde{a}_{ip} &= a_{ip}c - a_{iq}s \\ \tilde{a}_{iq} &= a_{ip}s + a_{iq}c \end{aligned} \tag{5.11}$$

$A' = {Q^{(p,q)}}^T \tilde{A}$: Nur die p-te und q-te Zeile werden geändert:

$$\begin{aligned} a'_{ij} &= \tilde{a}_{ij} \; für \; j \neq p \; oder \; q \\ a'_{pj} &= \tilde{a}_{pj}c - \tilde{a}_{qj}s \\ a'_{qj} &= \tilde{a}_{pj}s + \tilde{a}_{qj}c \end{aligned} \tag{5.12}$$

Dann folgt:

$$\begin{aligned} a'_{ij} &= a_{ij} \; für \; i,j \neq p \; oder \; q \\ a'_{ip} &= a_{ip}c - a_{iq}s \; für \; i \neq p \; oder \; q \\ a'_{iq} &= a_{ip}s + a_{iq}c \; für \; i \neq p \; oder \; q \\ a'_{pp} &= (a_{pp}c - a_{pq}s)c - (a_{qp}c - a_{qq}s)s \\ &= a_{pp}c^2 + a_{qq}s^2 - 2a_{pq}sc \\ a'_{qq} &= (a_{pp}s + a_{pq}c)s + (a_{qp}s + a_{qq}c)c \\ &= a_{pp}s^2 + a_{qq}c^2 + 2a_{pq}sc \\ a'_{pq} &= (a_{pp}s + a_{pq}c)c - (a_{qp}s + a_{qq}c)s \\ &= a_{pq}(c^2 - s^2) + sc(a_{pp} - a_{qq}) \end{aligned} \tag{5.13}$$

A' symmetrisch. Wir wollen nun s und c so wählen, dass $a'_{pq} = 0$ ist:

$$a_{pq}(c^2 - s^2) + sc(a_{pp} - a_{qq}) = 0 \tag{5.14}$$

Dann:

$$\frac{c^2 - s^2}{sc} = a_{qq} - a_{pp}a_{pq} = 2\beta \tag{5.15}$$

Dabei stellt $a_{pq} = 0$ kein Problem dar, da ja für diesen Fall die Transformation ohnehin nicht notwendig wäre.

Wir rechnen also:

$$\begin{aligned}
c^2 - s^2 &= 2sc\beta \\
s^2 + c^2 &= 1 \\
\Rightarrow 1 - 2s^2 &= 2sc\beta \\
1 - 4s^2 + 4s^4 &= 4s^2c^2\beta^2 = 4s^2\beta^2 - 4s^4\beta^2 \qquad (5.16)\\
s^4(4 + 4\beta^2) - s^2(4 + 4\beta^2) + 1 &= 0 \\
s^4 - s^2 + \frac{1}{4 + 4\beta^2} &= 0 \\
s^2 = \frac{1}{2} \pm \sqrt{\frac{1}{4} - \frac{1}{4 + 4\beta^2}} &= \frac{1}{2} \pm \frac{1}{2}\frac{\beta}{\sqrt{1 + \beta^2}}
\end{aligned}$$

Dabei entscheiden wir uns für die '-' Lösung, da diese numerisch stabiler ist.

$$\begin{aligned}
c^2 = 1 - s^2 &= \frac{1}{2} + \frac{1}{2}\frac{\beta}{\sqrt{1 + \beta^2}} \\
sc = \sqrt{s^2c^2} &= \sqrt{(\frac{1}{2} - \frac{1}{2}\frac{\beta}{\sqrt{1 + \beta^2}})(\frac{1}{2} + \frac{1}{2}\frac{\beta}{\sqrt{1 + \beta^2}})} \qquad (5.17)\\
&= \frac{1}{2}\frac{1}{\sqrt{1 + \beta^2}}
\end{aligned}$$

<u>Zusammengefasst:</u>

$$A' = Q^{(p,q)^T} A Q^{(p,q)} \qquad (5.18)$$

Für einen Eintrag $a_{pq} \neq 0$ bilde:

$$\begin{aligned}
\beta &= \frac{1}{2}\frac{a_{qq} - a_{pp}}{a_{pq}} \\
s^2 &= \frac{1}{2} - \frac{1}{2}\beta\frac{1}{\sqrt{1 + \beta^2}} \qquad (5.19)\\
c^2 &= \frac{1}{2} + \frac{1}{2}\beta\frac{1}{\sqrt{1 + \beta^2}} \\
sc &= \frac{1}{2}\frac{1}{\sqrt{1 + \beta^2}}
\end{aligned}$$

(5.20)

$$\begin{aligned}
a'_{ij} &= a_{ij} \; f\ddot{u}r \; i, j \neq p \; oder \; q \\
a'_{ip} &= a_{ip}c - a_{iq}s \; f\ddot{u}r \; i \neq p \; oder \; q \\
a'_{iq} &= a_{ip}s + a_{iq}c \; f\ddot{u}r \; i \neq p \; oder \; q \\
a'_{pp} &= a_{pp}c^2 + a_{qq}s^2 - 2a_{pq}sc \qquad (5.21)\\
a'_{qq} &= a_{pp}s^2 + a_{qq}c^2 + 2a_{pq}sc \\
a'_{pq} &= 0
\end{aligned}$$

und symmetrisch ergänzt. Der Algorithmus ist von folgender Form:

1. $A^{(0)} = A,\ V^{(0)} = \mathbb{1}$, Initialisierung
2. $A^{(n)} = Q^T A^{(n-1)} Q$, ein sweep
   $V^{(n)} = V^{(n-1)} Q$
   $Q = Q^{(1,2)} Q^{(1,3)} \dots Q^{(1,N)} Q^{(2,3)} \dots Q^{(2,N)} \dots Q^{(N-1,N)}$
   (weglassen falls ${a_{pq}}^{(n-1)} = 0$)
3. $A^{(n)} \to diag(\lambda_1, \lambda_2, \dots, \lambda_N)\ für\ n \to \infty$
   $V^{(n)} \to (\vec{v}^{(1)}, \vec{v}^{(2)}, \dots, \vec{v}^{(N)})\ für\ n \to \infty$
4. Abbruchkriterium:
   $\sum_{i \neq j}^{N} a_{ij}^2 \leq \epsilon \sum_{i,j=1}^{N} a_{ij}^2$

Wie sieht es nun mit der Konvergenz aus? Dazu betrachten wir die Summe der Quadrate der Nichtdiagonalelemente.

$$S = \sum_{i \neq j} a_{ij}^2 = 2 \sum_{i<j} a_{ij}^2 \tag{5.22}$$

$S \geq 0$, wenn $S = 0$ Diagonalmatrix. Dann ist:

$$\begin{aligned} S' &= \sum_{i \neq j} {a'_{ij}}^2 = \sum_{i \neq j, i, j \neq p, q} (a'_{ij})^2 + 2 \sum_{i \neq p, q} (a'_{ip})^2 + (a'_{ip})^2 \\ &= \sum_{i \neq j, i, j \neq p, q} (a_{ij})^2 + 2 \sum_{i \neq p, q} [(a_{ip} c - a_{iq} s)^2 + (a_{ip} s + a_{iq} c)^2] \\ &= \sum_{i \neq j, i, j \neq p, q} (a_{ij})^2 + 2 \sum_{i \neq p, q} (a_{ip}^2 + a_{iq}^2) = S - 2 a_{pq}^2 \end{aligned} \tag{5.23}$$

Dabei ist S die alte Summe.
S konvergiert gegen Null, da die Summe stets verringert wird. Die Matrix geht dann gegen die Diagonalmatrix, auf deren Diagonale die Eigenwerte stehen.

$$S \to 0 \Rightarrow A^{(n)} \to diag(\lambda_1, \dots, \lambda_N) \tag{5.24}$$

Verallgemeinerung auf hermite'sche Matrizen

H hermite'sch, dh. die Eigenwerte sind reell.

$$\begin{aligned} H &= A + iB \\ H^\dagger &= A^T - iB^T = A + iB = H \\ \Rightarrow A^T &= A, B^T = -B \end{aligned} \tag{5.25}$$

Eigenwertproblem:

$$\begin{aligned} H\vec{v} &= r\vec{v} \\ \vec{v} &= \vec{u} + i\vec{w}, \quad \vec{u}, \vec{v} \in \mathbb{R} \\ \Rightarrow (A + iB)(\vec{u} + i\vec{w}) &= r(\vec{u} + i\vec{w}) \\ A\vec{u} - B\vec{w} &= \vec{r}\vec{u} \quad (\Re e) \\ B\vec{u} + A\vec{w} &= \vec{r}\vec{w} \quad (\Im m) \end{aligned} \tag{5.26}$$

$$\underbrace{\begin{pmatrix} A & -B \\ B & A \end{pmatrix}}_{M} \begin{pmatrix} \vec{u} \\ \vec{w} \end{pmatrix} = r \begin{pmatrix} \vec{u} \\ \vec{w} \end{pmatrix} \tag{5.27}$$

Wie schon in Gleichung (4.29) finden wir auch hier eine symmetrische Matrix: $M^T = M$. Wir haben es mit einem $2N \times 2N$-Problem zu tun. Wir bekommen also auch $2N$ Eigenwerte.

$$M\vec{x} = r\vec{x} \tag{5.28}$$

hat zwei Lösungen:

$$\vec{x} = \begin{pmatrix} \vec{u} \\ \vec{w} \end{pmatrix}, \quad \vec{x}' = \begin{pmatrix} -\vec{w} \\ \vec{u} \end{pmatrix} \tag{5.29}$$

$\vec{x}'$ ist auch eine Lösung, wie man durch Einsetzen und Vergleichen ersehen kann:

$$\begin{aligned} \begin{pmatrix} A & -B \\ B & A \end{pmatrix} \vec{x}' &= \begin{pmatrix} A & -B \\ B & A \end{pmatrix} \begin{pmatrix} -\vec{w} \\ \vec{u} \end{pmatrix} \\ &= \begin{pmatrix} -A\vec{w} - B\vec{u} \\ -B\vec{w} + A\vec{u} \end{pmatrix} \\ &= \begin{pmatrix} -r\vec{w} \\ r\vec{u} \end{pmatrix} = r\vec{x}' \end{aligned} \tag{5.30}$$

$$\begin{aligned} \vec{x} &\Rightarrow \vec{v} = \vec{u} + i\vec{w} \\ \vec{x}' &\Rightarrow \vec{v}' = -\vec{w} + i\vec{u} = i(\vec{u} + i\vec{w}) = i\vec{v} \end{aligned} \tag{5.31}$$

Eingebettetes Problem: Alle Eigenwerte sind mindestens doppelt entartet. Man kann also die Hälfte der Werte verwerfen.

5.3 Iterative Methoden für dominante Eigenwerte

Sei A reelle Matrix. Dann ist $P(\lambda) = \det(A - \lambda\mathbb{1})$ ein Polynom mit reellen Koeffizienten. Die Nullstellen (Eigenwerte) sind dann entweder reell oder kommen in komplex konjugierten Paaren. Wir haben also zwei Fälle zu betrachten:
reeller Fall: $\pm\lambda_1 = |\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq |\lambda_4| \ldots$
komplexer Fall: $\lambda_1 = \alpha + i\beta$, $\lambda_2 = \alpha - i\beta$
$\lambda_1 = |\lambda_2| > |\lambda_2| > |\lambda_3| \geq |\lambda_4| \ldots$

Betrachten also nun den reellen Fall:
Sei $\vec{x}^{(0)}$ beliebig, $\vec{x}^{(n)} = A\vec{x}^{(n-1)} = A^n\vec{x}^{(0)}$.
Die Entwicklung von $\vec{x}^{(0)}$ in den noch unbekannten Eigenvektoren von A ist dann:

$$\vec{x}^{(0)} = \sum_{i=1}^{N} c_i \cdot \vec{e}^{(i)}, \quad A\vec{e}^{(i)} = \lambda_i \vec{e}^{(i)}, \tag{5.32}$$

$$\begin{aligned} \vec{x}^{(n)} = A^n\vec{x}^{(0)} = A^n \sum_{i=1}^{N} c_i A^n \vec{e}^{(i)} &= \sum_{i=1}^{N} c_i \lambda_i^n \vec{e}^{(i)} \\ &= \lambda_1^n (c_1 \cdot e^{(1)} + R_n), \end{aligned} \tag{5.33}$$

$$R_n = \lambda_1^{-n}(\sum_{i=2}^{N} c_i \lambda_i^n \vec{e}^{(i)}) = \sum_{i=2}^{N} (\frac{\lambda_i}{\lambda_1})^n c_i \vec{e}^{(i)} \overset{n\to\infty}{\to} 0, \tag{5.34}$$

mit

$$|\frac{\lambda_i}{\lambda_1}| < 1,$$

und

$$\vec{x}^{(n)} \overset{n\to\infty}{\to} c_1 (\lambda_1)^n \vec{e}^{(1)}. \tag{5.35}$$

Als Vorraussetzung muss also gelten, dass $c_1 \neq 0$ ist. $\vec{x}^{(0)}$ muss entsprechend gewählt werden. Mit (5.35): $\vec{x}^{(n+1)} = \lambda_1 \vec{x}^{(n)} \Rightarrow \lambda_1$, und

$$\lambda_1 + (5.35) = c_1 \cdot \vec{e}^{(1)} \Rightarrow \vec{e}^{(1)}, \tag{5.36}$$

wobei im letzten Schritt normiert wurde. Betrachten wir nun den komplexen Fall: Dazu ein Beispiel:

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \quad (1-\lambda)^2 + 1 = 0 \Rightarrow (\lambda - 1)^2 = -1 \Rightarrow \lambda = 1 \pm i. \tag{5.37}$$

$$\vec{x}^{(0)} = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}, \vec{x}^{(1)} = \begin{pmatrix} \gamma - \delta \\ \gamma + \delta \end{pmatrix}, \vec{x}^{(2)} = 2\begin{pmatrix} -\delta \\ \gamma \end{pmatrix}, \tag{5.38}$$
$$\vec{x}^{(3)} = \begin{pmatrix} -2\gamma - 2\delta \\ 2\gamma - 2\delta \end{pmatrix}, \vec{x}^{(4)} = -4\begin{pmatrix} \gamma \\ \delta \end{pmatrix}, \ldots$$

$\Rightarrow$ Kombination aus Drehungen und Streckungen.

$$\lambda_1 = \rho e^{i\Theta}, \ \lambda_2 = \lambda_1^* = \rho e^{-i\Theta}, \tag{5.39}$$

und

$$\rho = |\lambda_1| = |\lambda_2| > |\lambda_3| \geq |\lambda_4| \geq \ldots. \tag{5.40}$$

$$\vec{x}^{(n)} = A\vec{x}^{(n-1)} = A^n \vec{x}^{(0)}, \tag{5.41}$$

$$\vec{x}^{(0)} = \sum_{i=1}^{N} c_i \vec{e}^{(i)},$$

$$\begin{aligned} \vec{x}^{(n)} &= A^n \vec{x}^{(0)} = \sum_{i=1}^{N} c_i A^n \vec{e}_1^{(i)} = \sum_{i=1}^{N} c_i (\lambda_i)^n \vec{e}^{(i)} \\ &= \rho^n (c, e^{i\Theta_n} \vec{e}^{(1)} + c_2 e^{-i\Theta_n} \vec{e}^{(2)} + R_n) \\ R_n &= \sum_{i=1}^{N} c_i (\frac{\lambda_i}{\rho})^n \vec{e}^{(i)} \overset{n\to\infty}{\to} 0, \end{aligned} \tag{5.42}$$

$$\begin{aligned} \vec{x}^{(n)} &\approx c_1 \rho^n e^{in\Theta} \vec{e}^{(1)} + c_2 \rho^n e^{-in\Theta} \vec{e}^{(2)} \\ &= c_1 \lambda_1^n \vec{e}^{(1)} + C_2 \lambda_2^n \vec{e}^{(2)}, \quad \lambda_2 = \bar{\lambda_1}. \end{aligned} \tag{5.43}$$

$$P(\lambda) = c \prod_{n=1}^{N} (\lambda - \lambda_n) \tag{5.44}$$

ist charakteristisches Polynom. Der Beitrag der beiden Eigenwerte $\lambda_1,\ \lambda_2$ zum charakteristischen Polynom ist

$$(\lambda-\lambda_1)(\lambda-\lambda_2)=(\lambda-\lambda_1)(\lambda-\bar{\lambda_1})=\lambda^2-\lambda\underbrace{(\lambda_1+\bar{\lambda_1})}_{-a}+\underbrace{\lambda_1\bar{\lambda_1}}_{b}=0. \tag{5.45}$$

Wir haben also:

$$\begin{aligned}\lambda^2+a\lambda+b &= 0 \quad /\cdot\lambda^n \\ \lambda^{2+n}+a\lambda^{1+n}+b\lambda^n &= 0.\end{aligned} \tag{5.46}$$

$a=-\lambda_1-\bar{\lambda_1}=-2\Re e(\lambda_1)\in\mathbb{R}$ und $b=\lambda_1\bar{\lambda_1}=|\lambda_1|^2\in\mathbb{R}$. Dann:

$$\begin{aligned}(\lambda_1)^{n+2}c_1\vec{e}^{(1)}+(\lambda_2)^{n+2}c_2\vec{e}^{(2)} &+ \\ +a((\lambda_1)^{n+1}c_1\vec{e}^{(1)}+(\lambda_2)^{n+1}c_2\vec{e}^{(2)}) &+ \\ +b((\lambda_1)^{n}c_1\vec{e}^{(1)}+(\lambda_2)^{n}c_2\vec{e}^{(2)}) &= 0, \quad a,b\in\mathbb{R}.\end{aligned} \tag{5.47}$$

Wir haben gezeigt:

$$\begin{aligned}\vec{x}^{(n+2)}+a\vec{x}^{(n+1)}+b\vec{x}^{(n)} &= 0 \\ \vec{x}_j^{(n+2)}+a\vec{x}_j^{(n+1)}+b\vec{x}_j^{(n)} &= 0, \quad a,b\in\mathbb{R}.\end{aligned} \tag{5.48}$$

Verwende nun die $\vec{x}^{(n+2)},\vec{x}^{(n+1)},\vec{x}^{(n)}$ um die reellen Proportionalitätskonstanten abzulesen.

$$\begin{aligned}\Re e(\lambda_1) &= \Re e(\lambda_2)=-\frac{1}{2}, \\ b &= \alpha^2+\beta^2=|\lambda_1|^2 \\ \alpha &= -\frac{a}{2} \\ \lambda_{1,2} &= \alpha\pm i\beta \\ \lambda_{1,2} &= \frac{1}{2}(-a\pm i\sqrt{4b-a^2}).\end{aligned} \tag{5.49}$$

6 Fouriertransformation in der linearen Algebra

6.1 Grundlegende Gleichungen

$$\sum_{n=0}^{N-1} q^n = \frac{1-q^n}{1-q}, \tag{6.1}$$

setzen $q = \exp(i\frac{2\pi}{N}k)$, $k \in \mathbb{Z}, k \notin N \cdot l, l \in \mathbb{Z}$.

$$\sum_{n=0}^{N-1} \exp(i\frac{2\pi}{N}k \cdot n) = \frac{1-e^{i\frac{2\pi}{N}k\cdot N}}{1-e^{i\frac{2\pi}{N}k}}. \tag{6.2}$$

Ferner ist

$$\frac{1}{N}\sum_{n=0}^{N-1} e^{i\frac{2\pi}{N}n\cdot k} = \delta_{k,0},\ k \in \mathbb{Z}. \tag{6.3}$$

Als geometrische Interpretation denke man an die 5-te Einheitswurzel. Wir definieren eine Matrix U mit Elementen U_{kn},

$$U_{kn} = \frac{1}{\sqrt{N}} e^{-i\frac{2\pi}{N}n\cdot k},\ n,k = 1,2,\ldots,N. \tag{6.4}$$

Die Matrix U ist unitär:

$$(UU^\dagger)_{kl} = \sum_{i=1}^{N} U_{kn}(U^\dagger)_{nl} = \sum_{n=1}^{N} U_{kn}U^*_{ln} \tag{6.5}$$

$$= \frac{1}{N}\sum_{n=1}^{N} e^{-i\frac{2\pi}{N}nk}\cdot e^{i\frac{2\pi}{N}ln} = \frac{1}{N}\sum_{n=1}^{N} e^{i\frac{2\pi}{N}n(l-k)}$$

$$= \delta_{l-k,0} = \delta_{lk}. \tag{6.6}$$

Die unitäre Transformation erhält das Spektrum. Wir wollen nun die Fouriertransfromation eines Vektors betrachten.

$$\tilde{\vec{v}} = U\vec{v}, \tag{6.7}$$

$$\tilde{v}_n = (Uv)_n = \frac{1}{\sqrt{N}}\sum_{k=1}^{N} e^{-i\frac{2\pi}{N}nk}v_k. \tag{6.8}$$

Unitarität: $U^\dagger\tilde{\vec{v}} = \vec{v}$. Wir betrachten weiters die Fouriertransformation einer Matrix:

$$\tilde{M} = UMU^\dagger, \tag{6.9}$$

und mit der Unitarität von U folgt

$$M = U^\dagger\tilde{M}U. \tag{6.10}$$

Ferner halten wir fest, dass das Spektrum von M gleich jenem von $\tilde{M}$ ist. Zuletzt bemerken wir, dass gilt:

$$M^{-1} = U^\dagger(\tilde{M})^{-1}U, \tag{6.11}$$

da

$$MM^{-1} = \underbrace{U^\dagger\tilde{M}U}_{M}\underbrace{U^\dagger(\tilde{M})^{-1}U}_{M^{-1}} = U^\dagger\underbrace{\tilde{M}\tilde{M}^{-1}}_{\mathbb{1}}U = U^\dagger U = \mathbb{1}. \tag{6.12}$$

6.2 Diagonalisierung von Matrizen mittels Fouriertransformation

Hierzu kommen Matrizen mit Strukturen aus konstanten Koeffizienten in Betracht. Wir wollen uns nun einem Beispiel zuwenden:

$$M = \begin{pmatrix} 3 & -1 & 0 & \cdots & \cdots & 0 & -1 \\ -1 & \ddots & \ddots & \ddots & & & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & & \ddots & \ddots & \ddots & -1 \\ -1 & 0 & \cdots & \cdots & 0 & -1 & 3 \end{pmatrix}, \tag{6.13}$$

wir haben also

$$M = 3\delta_{nm} - \delta_{n+1,m} - \delta_{n-1,m},\ n, m = 1, 2 \ldots N. \tag{6.14}$$

Wir nehmen periodische Randbedingungen an, dh. $N + 1 \to 1$ bzw. $0 \to N$. Wenn wir nun die Fouriertransformierte dieser Matrix betrachten, so finden wir:

$$\begin{aligned} \tilde{M}_{kl} &= \sum_{n,m} U_{kn} M_{nm} U^*_{lm} \qquad (6.15) \\ &= \frac{1}{N} \sum_{n,m} e^{-i\frac{2\pi}{N}k\cdot n} (3\delta_{n,m} - \delta_{n+1,m} - \delta_{n-1,m}) e^{i\frac{2\pi}{N}l\cdot m} \\ &= \frac{1}{N} \sum_{n} e^{-i\frac{2\pi}{N}k\cdot n} (3e^{i\frac{2\pi}{N}l\cdot n} - e^{i\frac{2\pi}{N}l(n+1)} - e^{i\frac{2\pi}{N}l(n-1)}) \\ &= \underbrace{\frac{1}{N} \sum_{n} e^{-i\frac{2\pi}{N}k\cdot n} \cdot e^{i\frac{2\pi}{N}l\cdot n}}_{\delta_{kl}} (2 - \underbrace{e^{i\frac{2\pi}{N}l} - e^{-i\frac{2\pi}{N}l}}_{2\cos(\frac{2\pi}{N}l)}) \\ &= \delta_{kl}(3 - 2\cos(\frac{2\pi}{N}l)), \end{aligned}$$

was offensichtlich eine diagonale Form hat. Dann gilt also:

$$\tilde{M} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N) = D, \tag{6.16}$$

mit

$$\lambda_l = 3 - 2\cos(\frac{2\pi}{N}). \tag{6.17}$$

Damit stellt die Inversion der Matrix kein Problem mehr dar, wir invertieren und transformieren zurück und finden so einen Ausdruck für die Inverse unserer Aus-

gangsmatrix M:

$$\begin{aligned} M^{-1}_{nm} &= (U^\dagger \tilde{M}^{-1} U)_{nm} = (U^\dagger D U)_{nm} \\ &= \sum_{k,l} U^\dagger_{nk} D^{-1}_{kl} U_{lm} \\ &= \frac{1}{N} \sum_{k,l} e^{i\frac{2\pi}{N}nk} \frac{\delta_{kl}}{\lambda_k} e^{-i\frac{2\pi}{N}lm} \\ &= \frac{1}{N} \sum_{k} \frac{e^{i\frac{2\pi}{N}k(n-m)}}{\lambda_k}. \end{aligned} \tag{6.18}$$

Weiters haben wir für die Eigenvektoren:

$$\tilde{M} = UMU^\dagger = D, \tag{6.19}$$

von links mir $U^\dagger$ multipliziert:

$$MU^\dagger = U^\dagger D \tag{6.20}$$

ist Eigenwertgleichungssatz. $U^\dagger_l$ ist also der zum l-ten Eigenwert λ_l der Matrix M gehörende Eigenvektor. Wir haben dann:

$$U^\dagger_l = \vec{v}^{(l)} = \frac{1}{\sqrt{N}} \begin{pmatrix} e^{i\frac{2\pi}{N}l} \\ e^{i\frac{2\pi}{N}2l} \\ \vdots \\ e^{i\frac{2\pi}{N}Nl} \end{pmatrix} = e^{ipx} = \psi_p(x), \tag{6.21}$$

also eine ebene Welle.

7 Partielle Differentialgleichungen

7.1 Relaxationsmethoden für Randwertprobleme

Wir betrachten:
Laplace-Gleichung $\Delta\phi(\vec{x}) = 0$ + Randbedingungen,
Poisson-Gleichung $\Delta\phi(\vec{x}) = \rho(\vec{x})$ + Randbedingungen,
mit $\vec{x}$ aus einem Gebiet Ω, ferner sei $\phi(\vec{x})\Big|_{\partial\Omega}$ vorgegeben (Dirichlet, alternativ könnte man auch Von Neumann Randbedingungen wählen). Ein Beispiel wäre etwa ein Faraday-Käfig der geerdet ist, an den Knotenpunkten des Käfigs muss also gelten, dass die Feldverteilungen $\phi(\vec{x}) = const = 0$ sind. Wir nehmen eine Diskretisierung vor:

$$\begin{aligned} \vec{x} &\rightarrow \vec{n}\cdot\delta \\ x_i &\rightarrow n_i\delta,\ n_i = 0,1,\dots N_i, \quad i = 1,2,\dots,d \\ \phi(\vec{x}) &\rightarrow \phi_{\vec{n}} \\ \rho(\vec{x}) &\rightarrow \rho_{\vec{n}}.\ n \in \Omega \end{aligned} \tag{7.1}$$

Die diskretisierte Poissongleichung ist dann von folgender Form:

$$\Delta\phi(\vec{x}) = \sum_{i=1}^{d} \frac{\phi_{\vec{n}+\hat{i}} - 2\phi_{\vec{n}} + \phi_{\vec{n}-\hat{i}}}{\delta^2}. \tag{7.2}$$

Dann haben wir:

$$\begin{aligned} \sum_{i=1}^{d}(\phi_{\vec{n}+\hat{i}} - 2\phi_{\vec{n}} + \phi_{\vec{n}-\hat{i}}) &= \delta^2\rho_{\vec{n}} \\ \sum_{i=1}^{d}(\phi_{\vec{n}+\hat{i}} + \phi_{\vec{n}-\hat{i}}) - \delta^2\rho_{\vec{n}} &= 2d\phi_{\vec{n}}, \end{aligned} \tag{7.3}$$

damit

$$\phi_{\vec{n}} = \frac{1}{2d}\sum_{i=1}^{d}(\phi_{\vec{n}+\hat{i}} + \phi_{\vec{n}-\hat{i}}) - \frac{\delta^2}{2d}\rho_{\vec{n}}. \tag{7.4}$$

der erste Term der rechten Seite der letzten Gleichung ist dabei eine Mittelung der Nachbarn, der letzte Term der rechten Seite ist eine Korrektur durch den Quellterm. Die Idee ist es nun, die Mittelwertbildung + Korrektur so lange zu iterieren, bis die $\phi_{\vec{n}}$ konvergiert sind. Wir haben es also mit Generationen von Feldern zu tun, also etwa ist $\phi_{\vec{n}}^{(t)}$ die t-te Generation der Felder. Wir haben dann folgende Form:

$$\phi_{\vec{n}}^{(t+1)} = \frac{1}{2d}\sum_{i=1}^{d}(\phi_{\vec{n}+\hat{i}}^{(t)} + \phi_{\vec{n}-\hat{i}}^{(t)}) - \frac{\delta^2}{2d}\rho_{\vec{n}}. \tag{7.5}$$

Vergleiche dazu die Jacobi-Methode, bzw. die schnellere Gauss-Seidel-Methode.

7.2 Sukzessive Überrelaxation

$$\bar{\phi}_{\vec{n}} = \frac{1}{2d}\sum_{i=1}^{d}(\phi^{(t)}_{\vec{n}+\hat{i}} + \phi^{(t+1)}_{\vec{n}-\hat{i}}) - \frac{\delta^2}{2d}\rho_{\vec{n}}. \tag{7.6}$$

$$\phi^{(t+1)}_{\vec{n}} = \phi^{(t)}_{\vec{n}} + \omega(\bar{\phi}_{\vec{n}} - \phi^{(t)}_{\vec{n}}), \tag{7.7}$$

wobei wir mit

$\omega = 1$	Gauss-Seidel
$\omega > 1$	Überrelaxation
$\omega < 1$	Unterrelaxation

erhalten. Bei der Überrelaxation wird versucht die Änderung beim nächsten Schritt bereits zu berücksichtigen.

Satz 5. *Für $0 < \omega < 2$ haben wir Konvergenz.*

Bemerkung: Überrelaxation ist in dieser Hinsicht besser als Gauss-Seidel.
Wir brechen ab, wenn $|\phi^{(t+1)}_{\vec{n}} - \phi^{(t)}_{\vec{n}}| < \epsilon, \quad \forall \vec{n}$ erfüllt ist. Für die Lösung haben wir:

$$\phi^{(t+1)}_{\vec{n}} = \phi^{(t)}_{\vec{n}} \Rightarrow \bar{\phi}_{\vec{n}} - \phi^{(t)}_{\vec{n}} = 0, \tag{7.8}$$

dh.

$$\phi^{(t)}_{\vec{n}} = \frac{1}{2d}\sum_{i=1}^{d}(\phi_{\vec{n}+\hat{i}} + \phi_{\vec{n}-\hat{i}}) - \frac{\delta^2}{2d}\rho_{\vec{n}} \tag{7.9}$$

ist eine Lösung. Die Geschwindigkeit von SOR hängt stark von ω ab. ω muss experimentell bestimmt werden.

7.3 Anfangswertprobleme

Nehmen wir an, wir hätten eine Anfangsverteilung $U(\vec{x},t)\Big|_{t=0} = \rho(\vec{x})$ vorgegeben.Wir wollen die Evolution von $U(\vec{x},t)$ in der Zeit betrachten. Dabei haben wir das (numerische) Problem der Stabilität der Zeitentwicklung zu lösen. Betrachten wir wieder ein Beispiel:
Wärmeleitung/Diffusionsgleichung:

$$\frac{\partial}{\partial t}U(\vec{x},t) = \nabla D(\vec{x})U(\vec{x},t)\nabla U(\vec{x},t), \tag{7.10}$$

somit haben wir für ein konstantes D (Diffusionskonstante):

$$\frac{\partial}{\partial t}U(\vec{x},t) = D\Delta U(\vec{x},t). \tag{7.11}$$

Für eine Wellengleichung hätten wir etwa

$$\frac{\partial^2}{\partial t^2} = U(\vec{x},t) = v^2\Delta U(\vec{x},t), \tag{7.12}$$

und für die Schrödingergleichung

$$\frac{i}{\hbar}\frac{\partial}{\partial t}\psi(\vec{x},t) = \hat{H}\psi(\vec{x},t), \tag{7.13}$$

mit

$$\hat{H} = -\frac{\hbar^2}{2m}\Delta + V(\vec{x}). \tag{7.14}$$

Einfache partielle Differentialgleichungen: *Flusskonservative Systeme*, dh.

$$\frac{\partial \vec{U}(x,t)}{\partial t} = -\frac{\partial}{\partial x}\vec{F}(\vec{U}(x,t)). \tag{7.15}$$

Manche partielle Differentialgleichungen lassen sich in flusskonservative Systeme überführen. Betrachten wir das anhand der Wellengleichung.

$$\frac{\partial^2}{\partial t^2}\phi(x,t) = v^2\frac{\partial^2\phi(x,t)}{\partial x^2}. \tag{7.16}$$

Wir definieren:

$$r(x,t) = v\frac{\partial\phi(x,t)}{\partial x}, \tag{7.17}$$

$$s(x,t) = \frac{\partial\phi(x,t)}{\partial t}, \tag{7.18}$$

und wir betrachten zwei Differentialgleichungen:

$$\frac{\partial r}{\partial t} = v\frac{\partial s}{\partial x}, \tag{7.19}$$

$$\frac{\partial s}{\partial t} = v\frac{\partial r}{\partial x}. \tag{7.20}$$

Unter Verwendung dieser beiden Gleichungen haben wir dann:

$$\frac{\partial r}{\partial t} = v\frac{\partial}{\partial t}\frac{\partial}{\partial x}\phi = v\frac{\partial s}{\partial x} = v\frac{\partial}{\partial x}\frac{\partial}{\partial t}\phi \tag{7.21}$$

als Konsistenzgleichung. Weiters haben wir:

$$\frac{\partial s}{\partial t} = \frac{\partial}{\partial t}\frac{\partial}{\partial t}\phi = v\frac{\partial r}{\partial x} = v^2\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi \tag{7.22}$$

Wellengleichung. Wir definieren:

$$\vec{U}(\vec{x},t) = \begin{pmatrix} r(\vec{x},t) \\ s(\vec{x},t) \end{pmatrix}, \qquad \frac{\partial\vec{U}(x,t)}{\partial t} = -\frac{\partial}{\partial x}\vec{F}(\vec{U}(x,t)). \tag{7.23}$$

Damit ist

$$\vec{F}(\vec{U}(x,t)) = -\begin{pmatrix} 0 & v \\ v & 0 \end{pmatrix}\vec{U}(x,t), \tag{7.24}$$

und als Lösung des flusskonservativen Systemes haben wir

$$\begin{matrix} r(x,t) \\ s(x,t) \end{matrix} \Rightarrow \phi(x,t) = \int_0^t s(x,t')dt'. \tag{7.25}$$

7.4 Spezielle Verfahren bei Diffusions- und Schrödingergleichung

7.4.1 Diffusionsgleichung mit konstanten Koeffizienten

$$\frac{\partial U}{\partial t} = D\frac{\partial^2 U}{\partial x^2}. \tag{7.26}$$

Wir nehemen eine *forward time centred space* Diskretisierung vor:

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = D\frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{\Delta x^2} + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2). \tag{7.27}$$

$$U_j^{n+1} = U_j^n + \frac{\Delta t D}{\Delta x^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n). \tag{7.28}$$

Nachdem Stabilität bei AWP stets ein Problem darstellt, nehmen wir eine Stabilitätsanalyse vor.

$$2\frac{D\Delta t}{\Delta x^2} \overset{!}{<} 1, \tag{7.29}$$

bzw.

$$\Delta t \overset{!}{<} \frac{\Delta x^2}{2D}. \tag{7.30}$$

Für die meisten Anwendungen ist ein zu kleines Δt erforderlich. Eine alternative Diskretisierung wäre etwa folgende:

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = D\frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}}{\Delta x^2} + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2). \tag{7.31}$$

Nun Mitteln wir die beiden Diskretisierungen und finden:

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = \frac{D}{2}\frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1} + U_{j+1}^n - 2U_j^n + U_{j-1}^n}{\Delta x^2}. \tag{7.32}$$

Damit folgt:

$$U_j^{n+1} - \frac{D\Delta t}{2\Delta x^2}(U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}) = U_j^n + \frac{D\Delta t}{2\Delta x^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n). \tag{7.33}$$

Zusammen mit

$$H_{jj'} = \frac{D\Delta t}{2\Delta x^2}(\delta_{j+1,j'} - 2\delta_{jj'} + \delta_{j-1,j'}), \tag{7.34}$$

sowie periodischen Randbedingungen, haben wir dann:

$$\begin{aligned} \sum_{j'}(\mathbb{1} - H)_{jj'}U_j^{n+1} &= \sum_{j'}(\mathbb{1} + H)_{jj'}U_{j'}^n \\ (\mathbb{1} - h)U^{n+1} &= (\mathbb{1} + H)U^n \\ U^{n+1} &= (\mathbb{1} - H)^{-1}(\mathbb{1} + H)U^n \end{aligned} \tag{7.35}$$

ist das *Crank-Nicholson*-Schema. Dieses implizite Schema ist stabil. Für ein konstantes D kann $(\mathbb{1} - H)^{-1}$ mittels Fouriertransformation berechnet werden, vergleiche dazu Übung 11.

7.4.2 Zeitunabhängige Schrödingergleichung

$$i\frac{\partial\psi(x,t)}{\partial t} = -\frac{\partial^2\psi(x,t)}{\partial x^2} + V(x)\psi(x,t). \tag{7.36}$$

Die forward-time-centred-space Diskretisierung liefert:

$$i\frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} = -\frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{\Delta x^2} + V_j\psi_j^n. \tag{7.37}$$

Hier ist die Wahrscheinlichkeitsdichte nicht erhalten,

$$\int_{-\infty}^{\infty} |\psi(x,t)|^2 dx \tag{7.38}$$

ist nicht konstant 1 für alle t.

$$\begin{aligned} i\frac{\partial\psi}{\partial t} &= \hat{H}\psi, \\ \text{mit } \hat{H} &= -\frac{\partial^2}{\partial x^2} + V(x). \end{aligned} \tag{7.39}$$

Die unitäre Zeitentwicklung ist

$$\psi(x,t) = e^{-i\hat{H}t}\psi(x,0). \tag{7.40}$$

Die Gleichung

$$\psi(x,\Delta t) = e^{-i\hat{H}\Delta t}\psi(x,0) \tag{7.41}$$

von links mit $e^{i\frac{\hat{H}\Delta t}{2}}$ multipliziert liefert:

$$e^{i\hat{H}\frac{\Delta t}{2}}\psi(x,\Delta t) = e^{-i\hat{H}\frac{\Delta t}{2}}\psi(x,0). \tag{7.42}$$

Wenn wir nun in Δt entwickeln (vorrausgesetzt $\hat{H}$ ist ein beschränkter Operator), so erhalten wir:

$$(\mathbb{1} + i\frac{\Delta t}{2}\hat{H})\psi(x,\Delta t) = (\mathbb{1} - i\frac{\Delta t}{2}\hat{H})\psi(x,0) + \mathcal{O}(\Delta t^2). \tag{7.43}$$

Nun können wir eine Diskretisierung vornehmen:

$$\hat{H}\psi_j^n = -\frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{\Delta x^2} + V_j\psi_j^n = \sum_{j'} H_{jj'}\psi_{j'}^n, \tag{7.44}$$

mit

$$H_{jj'} = -\frac{1}{\Delta x^2}(\delta_{j+1,j'} - 2\delta_{jj'} + \delta_{j-1,j'} + V_j\delta_{jj'}). \tag{7.45}$$

Wieder wollen wir zum Crank-Nicholson-Schema gelangen, es folgt also:

$$\sum_{j'}(\mathbb{1} + i\frac{\Delta t}{2}H)_{jj'}\psi_{j'}^{(n+1)} = \sum_{j'}(\mathbb{1} - i\frac{\Delta t}{2}H)_{jj'}\psi_{j'}^n \tag{7.46}$$

und damit

$$(\mathbb{1} + i\frac{\Delta t}{2}H)\psi^{n+1} = (\mathbb{1} - i\frac{\Delta t}{2}H)\psi^n, \tag{7.47}$$

und schließlich

$$\psi^{n+1} = (\mathbb{1} + i\frac{\Delta t}{2}H)^{-1}(\mathbb{1} - i\frac{\Delta t}{2}H)\psi^n. \tag{7.48}$$

Die Inversion ist hier schwierig, da keine Inversion durch Fouriertransformation vorgenommen werden kann, da keine konstanten Koeffizienten in V_j sind.

8 Projektausarbeitungen

8.1 Übung 1: Ebenes Fachwerk

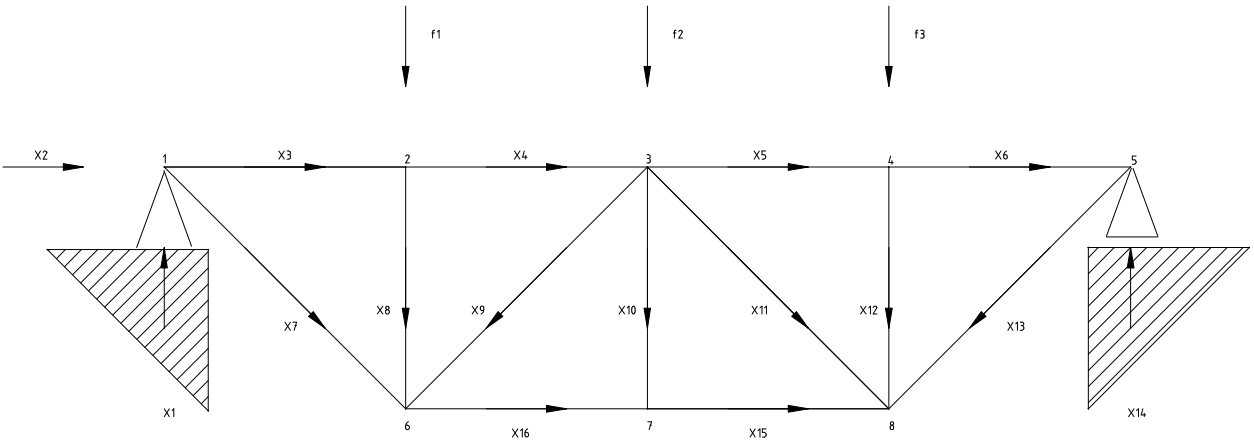


Abbildung 3: Ebenes Fachwerk mit Kräfteverteilung

Wir sehen uns nocheinmal das ebene Fachwerk an. Im Weiteren soll das Gleichungssystem aufgestellt werden, indem Knoten für Knoten durchgegangen wird.

Knoten 1:

$$\begin{aligned} \vec{x_1} + \vec{x_2} + \vec{x_3} + \vec{x_7} &= 0 \\ x_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} + x_2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_3 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_7 \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} &= 0 \\ x_2 + x_3 + \frac{1}{\sqrt{2}} x_7 &= 0 \\ x_1 - \frac{1}{\sqrt{2}} x_7 &= 0 \end{aligned} \tag{8.1}$$

Knoten 2:

$$\begin{aligned} \vec{x_3} + \vec{x_4} + \vec{x_8} + \vec{f_1} &= 0 \\ x_3 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_4 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_8 \begin{pmatrix} 0 \\ -1 \end{pmatrix} + f_1 \begin{pmatrix} 0 \\ -1 \end{pmatrix} &= 0 \\ x_3 + x_4 &= 0 \\ -x_8 &= f_1 \end{aligned} \tag{8.2}$$

Knoten 3:

$$\vec{x_4} + \vec{x_5} + \vec{x_9} + \vec{x_{10}} + \vec{x_{11}} + \vec{f_2} = 0$$

$$x_4 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_5 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_9 \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} + x_{10} \begin{pmatrix} 0 \\ -1 \end{pmatrix} + x_{11} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + f_2 \begin{pmatrix} 0 \\ -1 \end{pmatrix} = 0 \tag{8.3}$$

$$x_4 + x_5 - \frac{1}{\sqrt{2}} x_9 + \frac{1}{\sqrt{2}} x_{11} = 0$$

$$-\frac{1}{\sqrt{2}} x_9 - x_{10} - \frac{1}{\sqrt{2}} x_{11} = f_2$$

Knoten 4:

$$\vec{x_5} + \vec{x_6} + \vec{x_{12}} + \vec{f_3} = 0$$

$$x_5 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_6 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_{12} \begin{pmatrix} 0 \\ -1 \end{pmatrix} + f_3 \begin{pmatrix} 0 \\ -1 \end{pmatrix} = 0 \tag{8.4}$$

$$x_5 + x_6 = 0$$

$$-x_{12} = f_3$$

Knoten 5:

$$\vec{x_6} + \vec{x_{13}} + \vec{x_{14}} = 0$$

$$x_6 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_{13} \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + x_{14} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0 \tag{8.5}$$

$$x_6 - \frac{1}{\sqrt{2}} x_{13} = 0$$

$$-\frac{1}{\sqrt{2}} x_{13} + x_{14} = 0$$

Knoten 6:

$$\vec{x_7} + \vec{x_8} + \vec{x_9} + \vec{x_{16}} = 0$$

$$x_7 \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + x_8 \begin{pmatrix} 0 \\ -1 \end{pmatrix} + x_9 \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + x_{16} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0 \tag{8.6}$$

$$\frac{1}{\sqrt{2}} x_7 - \frac{1}{\sqrt{2}} x_9 + x_{16} = 0$$

$$-\frac{1}{\sqrt{2}} x_7 - x_8 - \frac{1}{\sqrt{2}} x_9 = 0$$

Knoten 7:

$$\begin{aligned} \vec{x_{16}} + \vec{x_{10}} + \vec{x_{15}} &= 0 \\ x_{16}\begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_{10}\begin{pmatrix} 0 \\ -1 \end{pmatrix} + x_{15}\begin{pmatrix} 1 \\ 0 \end{pmatrix} &= 0 \\ x_{16} + x_{15} &= 0 \\ x_{10} &= 0 \end{aligned} \tag{8.7}$$

Knoten 8:

$$\begin{aligned} \vec{x_{15}} + \vec{x_{11}} + \vec{x_{12}} + \vec{x_{13}} &= 0 \\ x_{15}\begin{pmatrix} 1 \\ 0 \end{pmatrix} + x_{11}\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + x_{12}\begin{pmatrix} 0 \\ -1 \end{pmatrix} + x_{13}\begin{pmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} &= 0 \\ x_{15} + \frac{1}{\sqrt{2}}x_{11} - \frac{1}{\sqrt{2}}x_{13} &= 0 \\ -\frac{1}{\sqrt{2}}x_{11} - x_{12} - \frac{1}{\sqrt{2}}x_{13} &= 0 \end{aligned} \tag{8.8}$$

Das Gleichungssystem lautet dann:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & -1 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & -1 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & -1 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ f_1 \\ 0 \\ f_2 \\ 0 \\ f_3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{8.9}$$

8.2 Übung 2: Gauß'sches Eliminationsverfahren

Bei dieser Übung ist das Gauß'sche Eliminationsverfahren zu Implementieren, sowie den Algorithmus für die Rücksubstitution. Der gesamte Code ist schon in Form eines Pseudocodes vorgegeben, und ist lediglich in die entsprechende Form zu bringen. Ich habe mich entschieden alle Programmieraufgaben für diesen Kurs in `FORTRAN` zu schreiben. Nachstehend also der Code, sowie der Output.

8.2.1 Sourcecode - Uebung2.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS

SUBROUTINE ge (M, p)                        !Gauss-Elimination mit Pivoting
        IMPLICIT NONE
        REAL, INTENT(INOUT) :: M(:, :)
        INTEGER, INTENT(INOUT) :: p(:)
        REAL :: piv, f
        INTEGER :: ipiv, aux

        INTEGER :: n, i, j, L
        L = SIZE (M, DIM=1)                 !# der Zeilen und Spalten

        DO n=1, L                           !Befuellen des Indexfeldes
         p(n) = n
        END DO
        DO n=1, L - 1                       !Masterloop

         piv = 0.0                          !Pivotelement ermitteln
         DO i=n, L
          IF (ABS(M(p(i),n)) > piv) THEN
           piv = ABS(M(p(i),n))
           ipiv = i
          END IF
         END DO

         aux     = p(n)                     !Hilfsvariable beladen
         p(n)    = p(ipiv)                  !Vertauschen im Indexfeld
         p(ipiv) = aux                      !Vertauschung abeschliessen

         DO i = n+1, L
          f = M(p(i),n)/M(p(n),n)
          M(p(i),n) = f                     !Faktoren statt 0 schreiben
          DO j = n+1, L
           M(p(i),j) = M(p(i),j)-f*M(p(n),j)
          END DO
         END DO

        END DO                              !Ende Masterloop

END SUBROUTINE ge

SUBROUTINE resubst (B, M, p, X)             !Ruecksubstitution

        IMPLICIT NONE

        REAL, INTENT(INOUT) :: B(:, :)
        REAL, INTENT(INOUT) :: M(:, :)
        REAL, INTENT(INOUT) :: X(:, :)
        INTEGER, INTENT(INOUT) :: p(:)

        INTEGER n, i, j, L, LB, ib
```

```
        L  =  SIZE(M, DIM=1)            !# der Zeilen
        LB =  SIZE(B, DIM=2)            !# der Gleichungssysteme


        DO n=1, L-1                     !Eliminationsschritte fuer rechte Seite B
         DO i=n+1,L
          DO j=1, LB
           B(p(i),j)=B(p(i),j)-M(p(i),n)*B(p(n),j)
          END DO
         END DO
        END DO


        DO ib=1,LB                      !Masterloop ueber Spalten von B
         DO n=L,1,-1
          X(n,ib)=B(p(n),ib)
          DO j=n+1, L
           X(n,ib)=X(n,ib)-M(p(n),j)*X(j,ib)
          END DO
          X(n,ib)=X(n,ib)/M(p(n),n)
         END DO
        END DO

END SUBROUTINE  resubst

SUBROUTINE ausgabe (a)
        IMPLICIT NONE
        REAL, INTENT(IN) :: a(:, :)

        INTEGER :: i
        DO i=1, SIZE(a, DIM=1)
                PRINT *, a(i, :)
        END DO
        PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
        USE LinAg
        IMPLICIT none
        REAL :: M(3,3), B(3,1)  !Matrix M und B
        REAL :: X(3,1)          !Matrix X
        INTEGER :: p(3)         !Permutationsvektor
        M(1,1) = 1.0            !Befuellen der Matrix
        M(1,2) = 2.0
        M(1,3) = 3.0
        M(2,1) = 7.0
        M(2,2) = -1.0
        M(2,3) = 4.0
        M(3,1) = 1.0
        M(3,2) = 1.0
        M(3,3) = 1.0
        B(1,1) = 2.0            !Befuellen des Vektors
        B(2,1) = 9.0
        B(3,1) = 4.0
        PRINT *, "Dieses Programm bringt Matrix M auf Zeilenstufenform und Resubstituiert."
        PRINT *
        PRINT *, "Matrix M:"
        CALL ausgabe (M)        !Kontrollausgabe
        CALL ge (M, p)          !Gauss-Elimination
        PRINT *, "Matrix M, nun auf Zeilenstufenform:"
        PRINT *, "(unter der Hauptdiagonale stehen die Faktoren f)"
        CALL ausgabe (M)        !Zeilenstufenform
        PRINT *, "mit Permutationsvektor (Reihenfolge der Zeilen):"
```

```
        PRINT *,p(:)              !Permutationsfolge
        PRINT *
        CALL resubst(B, M, p, X)!Resubstitution
        PRINT *, "Loesungsvektor, beginnend mit x1, x2, usf...:"
        CALL ausgabe (X)          !Loesungsaufgabe
END PROGRAM main
```

8.2.2 Output des Programmes

```
 Dieses Programm bringt Matrix M auf Zeilenstufenform und Resubstituiert.

 Matrix M:
   1.000000       2.000000       3.000000
   7.000000      -1.000000       4.000000
   1.000000       1.000000       1.000000

 Matrix M, nun auf Zeilenstufenform:
 (unter der Hauptdiagonale stehen die Faktoren f)
  0.1428571       2.142857       2.428571
   7.000000      -1.000000       4.000000
  0.1428571      0.5333334     -0.8666668

 mit Permutationsvektor (Reihenfolge der Zeilen):
           2           1           3

 Loesungsvektor, beginnend mit x1, x2, usf...:
   3.307692
   3.384615
  -2.692307
```

8.3 Übung 3: Zusammenfassung

In dieser Übung fassen wir nun unsere Unterprogramme von vorher zusammen. Außerdem fügen wir ein Unterprogramm für Matrixinversion hinzu, und ein weiteres zur Berechnung der Determinante. Eine Testfunktion die verifiziert ob es sich wirklich um die Inverse Matrix handelt wird ebenfalls implementiert.

Die Unterprogramme sind jetzt:

ge (M, p), führt die Gauß'sche Elimination aus.
resubst(B, M, p, X), führt die Resubstitution aus.
matrixinversion(M, p, Y), führt die Inversion aus und speichert die Inverse nach Y.
testinv(oldM, Y) multipliziert die alte Matrix M mit ihrer Inversen, liefert also $\mathbb{1}$.
det(M, p) berechnet die Determinante (zerstört dabei p).

8.3.1 Sourcecode - Uebung3.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS

SUBROUTINE ge (M, p)                        !Gauss-Elimination mit Pivoting
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: M(:, :)
        INTEGER, INTENT(INOUT) :: p(:)
        REAL(KIND=8) :: piv, f
        INTEGER :: ipiv, aux

        INTEGER :: n, i, j, L
        L = SIZE (M, DIM=1)                 !# der Zeilen und Spalten

        DO n=1, L                           !Befuellen des Indexfeldes
         p(n) = n
        END DO
        DO n=1, L - 1                       !Masterloop

         piv = 0.d0                         !Pivotelement ermitteln
         DO i=n, L
          IF (ABS(M(p(i),n)) > piv) THEN
            piv = ABS(M(p(i),n))
            ipiv = i
          END IF
         END DO

         aux     = p(n)                     !Hilfsvariable beladen
         p(n)    = p(ipiv)                  !Vertauschen im Indexfeld
         p(ipiv) = aux                      !Vertauschung abeschliessen

         DO i = n+1, L
          f = M(p(i),n)/M(p(n),n)
          M(p(i),n) = f                     !Faktoren statt 0 schreiben
          DO j = n+1, L
           M(p(i),j) = M(p(i),j)-f*M(p(n),j)
          END DO
         END DO

        END DO                              !Ende Masterloop

END SUBROUTINE ge

SUBROUTINE resubst (B, M, p, X)             !Ruecksubstitution
```

```
        IMPLICIT NONE

        REAL(KIND=8), INTENT(INOUT) :: B(:, :)
        REAL(KIND=8), INTENT(INOUT) :: M(:, :)
        REAL(KIND=8), INTENT(INOUT) :: X(:, :)
        INTEGER, INTENT(INOUT) :: p(:)

        INTEGER n, i, j, L, LB, ib

        L  =  SIZE(M, DIM=1)              !# der Zeilen
        LB =  SIZE(B, DIM=2)              !# der Gleichungssysteme


        DO n=1, L-1                       !Eliminationsschritte fuer rechte Seite B
         DO i=n+1,L
          DO j=1, LB
           B(p(i),j)=B(p(i),j)-M(p(i),n)*B(p(n),j)
          END DO
         END DO
        END DO

        DO ib=1,LB, 1                     !Masterloop ueber Spalten von B
         DO n=L,1,-1
          X(n,ib)=B(p(n),ib)
          DO j=n+1, L
           X(n,ib)=X(n,ib)-M(p(n),j)*X(j,ib)
          END DO
          X(n,ib)=X(n,ib)/M(p(n),n)
         END DO
        END DO
END SUBROUTINE  resubst

SUBROUTINE matrixinversion (M, p, Y)    !Matrixinversion

        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: M(:, :)
        REAL(KIND=8), INTENT(INOUT) :: Y(:, :)
        INTEGER, INTENT(INOUT) :: p(:)
        REAL(KIND=8) :: B(SIZE(M, DIM=1),SIZE(M, DIM=2))
        REAL(KIND=8) :: X(SIZE(Y, DIM=1),SIZE(Y, DIM=2))
        INTEGER L, i, n, j
        L = SIZE(M, DIM=1)
        B = 0.d0
        DO i=1, L                         !Fuellen der Einheitsmatrix
         B(i,i)=1.d0
        END DO
        CALL resubst (B, M, p, X)         !Loesen
        Y = X
END SUBROUTINE matrixinversion

SUBROUTINE testinv(oldM, Y)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: oldM(:, :)
        REAL(KIND=8), INTENT(INOUT) :: Y(:, :)
        REAL(KIND=8) :: E(SIZE(Y, DIM=1), SIZE(Y, DIM=2))
        E = 0.d0
        E = MATMUL(oldM, Y)
        CALL ausgabe(E)                   !Muss Einheit ergeben
END SUBROUTINE testinv

SUBROUTINE det(M, p)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: M(:, :)
        INTEGER, INTENT(INOUT) :: p(:)
        REAL(KIND=8) d
        INTEGER L, s, i, aux
        L = SIZE(M, DIM=1)
```

```
        s = 1                               !Vorzeichen
        d = 1.d0
        DO i=1, L
         d = d*M(p(i),i)
        END DO
        DO i=1, L                           !Vorzeichen ermitteln
         DO WHILE (i /= p(i))
          aux = p(i)
          p(i) = p(aux)
          p(aux) = aux
          s = -s
         END DO
        END DO
        d = d*s
        PRINT *, "Die Determinante von M ist:"
        PRINT *, d
END SUBROUTINE det

SUBROUTINE ausgabe (a)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(IN) :: a(:, :)

        INTEGER :: i, j
        DO i=1, SIZE(a, DIM=1)
         DO j=1, SIZE(a, DIM=2)
          WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
         END DO
         WRITE(*,*)
        END DO
        PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
        USE LinAg
        IMPLICIT none
        REAL(KIND=8) :: oldM(3,3), M(3,3), B(3,2)       !Matrix M und B
        REAL(KIND=8) :: X(3,2)                  !Matrix X
        REAL(KIND=8) :: Y(3,3)                  !Matrix Y, beinhaltet spaeter Inverse
        INTEGER :: p(3)                 !Permutationsvektor
        M(1,1) = 1.d0                   !Befuellen der Matrix
        M(1,2) = 2.d0
        M(1,3) = 3.d0
        M(2,1) = 7.d0
        M(2,2) = -1.d0
        M(2,3) = 4.d0
        M(3,1) = 1.d0
        M(3,2) = 1.d0
        M(3,3) = 1.d0
        B(1,1) = 2.d0                   !Befuellen des Vektors
        B(2,1) = 9.d0
        B(3,1) = 4.d0
        B(1,2) = 4.d0
        B(2,2) = -1.d0
        B(3,2) = 0.d0
        oldM = M                        !Speichern der urspr. Matrix
        PRINT *, "Dieses Programm bringt Matrix M auf Zeilenstufenform und Resubstituiert."
        PRINT *
        PRINT *, "Matrix M:"
        CALL ausgabe (oldM)             !Kontrollausgabe
        CALL ge (M, p)                  !Gauss-Elimination
        PRINT *, "Matrix M, nun auf Zeilenstufenform:"
        PRINT *, "(unter der Hauptdiagonale stehen die Faktoren f)"
        CALL ausgabe (M)                !Zeilenstufenform
        PRINT *, "mit Permutationsvektor (Reihenfolge der Zeilen):"
```

```
        PRINT *,p(:)            !Permutationsfolge
        PRINT *
        PRINT *, "Matrix B:"
        CALL ausgabe (B)
        PRINT *
        PRINT *, "Loesungsvektor, beginnend mit x1, x2, usf. untereinander fuer
        System 1, 2, usf..:"
        CALL resubst(B, M, p, X)        !Resubstitution
        CALL ausgabe (X)                !Loesungsausgabe
        CALL matrixinversion (M, p, Y)  !Invertiere M, schreibe Inverse in Y
        PRINT *, "Die Inverse der urspruenglichen Matrix M:"
        CALL ausgabe (Y)                !Ausgabe der Inversen
        PRINT *
        PRINT *, "Testen die Inverse:"
        CALL testinv(oldM, Y)           !Multipliziere Inverse mit urspr. Matrix
        PRINT *
        PRINT *, "Berechnung der Determinante: (zerstoert p-Vektor):"
        CALL det(M, p)
END PROGRAM main
```

8.3.2 Output des Programmes

```
 Dieses Programm bringt Matrix M auf Zeilenstufenform und Resubstituiert.

 Matrix M:
 1.00000 2.00000 3.00000
 7.00000-1.00000 4.00000
 1.00000 1.00000 1.00000

 Matrix M, nun auf Zeilenstufenform:
 (unter der Hauptdiagonale stehen die Faktoren f)
 0.14286 2.14286 2.42857
 7.00000-1.00000 4.00000
 0.14286 0.53333-0.86667

 mit Permutationsvektor (Reihenfolge der Zeilen):
           2           1           3

 Matrix B:
 2.00000 4.00000
 9.00000-1.00000
 4.00000 0.00000


 Loesungsvektor, beginnend mit x1, x2, usf. untereinander fuer System 1, 2, usf..:
 3.30769-1.61538
 3.38462-0.76923
-2.69231 2.38462

 Die Inverse der urspruenglichen Matrix M:
-0.38462 0.07692 0.84615
-0.23077-0.15385 1.30769
 0.61538 0.07692-1.15385
```

```
Testen die Inverse:
1.00000 0.00000 0.00000
0.00000 1.00000 0.00000
0.00000 0.00000 1.00000


Berechnung der Determinante: (zerstoert p-Vektor):
Die Determinante von M ist:
  13.0000000000000
```

8.4 Übung 4: Das erste Projekt

Bei dieser Übung sollen die Programmteile aus vorherigen Übungen verwendet werden um folgende Aufgabe zu lösen. Das Gleichungssystem des Fachwerkes von vorher soll mit Gauss'scher Elimination gelöst werden. Dabei soll das Fachwerk zuerst symmetrisch, dann asymmetrisch und auch noch dynamisch belastet werden. Für die symmetrische Belastung ist die Kraft wie folgt aufzuteilen: $f_1 = f/3,\ f_2 = f/3,\ f_3 = f/3$, wobei $f \in [0, 1]$ in kleinen Schritten erhöht werden soll. Das asymmetrische Belastungsprofil ist $f_1 = 0.2f,\ f_2 = 0.3f,\ f_3 = 0.5f$, wieder ist f aus dem selben Intervall. Für das dynamische Belastungsprofil habe ich mich für eine Gauss'sche Verteilung entschieden. Für alle drei Fälle sind Plots zu erstellen, die in den statischen Fällen die Kräfte f_i

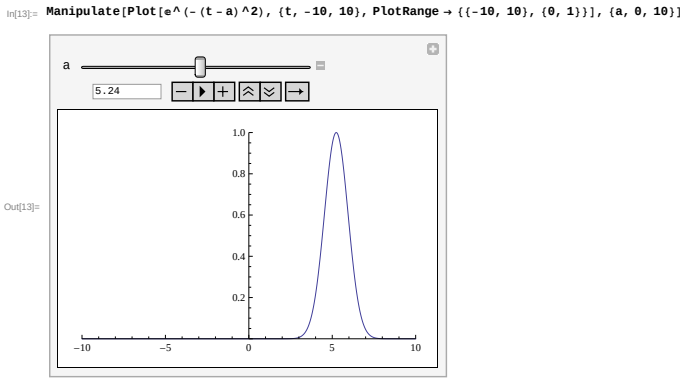


Abbildung 4: Gaussverteilung als Belastungsprofil. Der freie Parameter lässt die Kurve über die Brücke wandern.

über f, und im dynamischen Falle über der Zeit t darstellen. Wir finden starke Entartung vor,

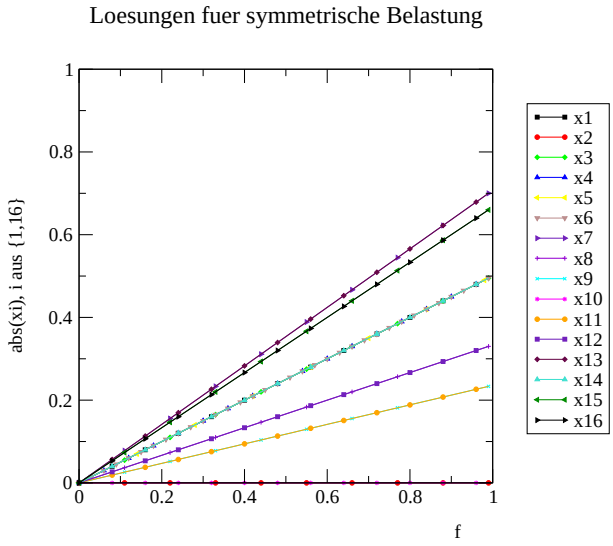


Abbildung 5: Symmetrisches Belastungsprofil.

was aber aufgrund der Symmetrie nicht weiter verwunderlich ist. Außerdem sind die Kräfte 2 und 10 gleich Null. Somit kann die mittlere Strebe des Fachwerkes entfernt werden.

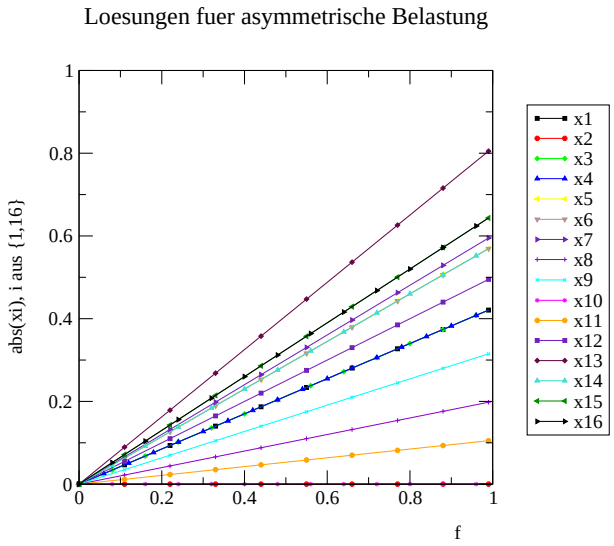


Abbildung 6: Asymmetrisches Belastungsprofil.

Der Entartungsgrad ist nun reduziert, dennoch gibt es verbleibende Entartungen im System. Zuletzt folgt das dynamische Belastungsprofil. Die Gaussverteilung fährt in ca. 10 Zeiteinheiten über die Brücke. Die Breite des Gauss ist dabei so gewählt, dass er gerade zwei Belastungspunkte zu einer Zeit mit etwa einem Zehntel seiner Höhe belasten kann, nie aber alle drei Punkte gleichzeitig.

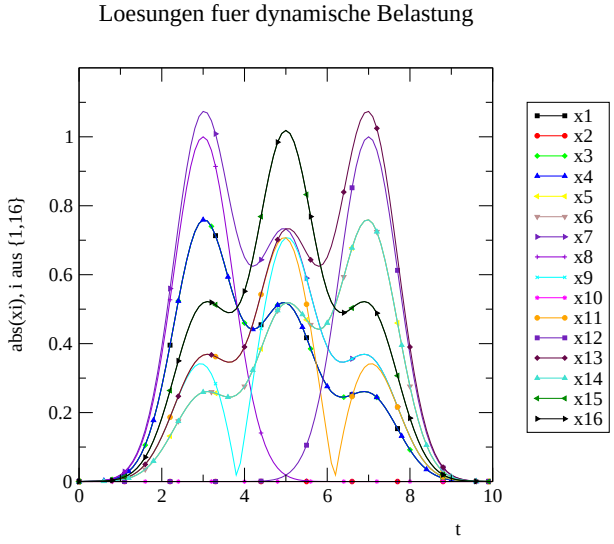


Abbildung 7: Dynamisches Belastungsprofil.

8.4.1 Sourcecode - Uebung4.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS

SUBROUTINE ge (M, p)                        !Gauss-Elimination mit Pivoting
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: M(:, :)
        INTEGER, INTENT(INOUT) :: p(:)
        REAL(KIND=8) :: piv, f
        INTEGER :: ipiv, aux

        INTEGER :: n, i, j, L
        L = SIZE (M, DIM=1)                 !# der Zeilen und Spalten

        DO n=1, L                           !Befuellen des Indexfeldes
         p(n) = n
        END DO
        DO n=1, L - 1                       !Masterloop

         piv = 0.d0                         !Pivotelement ermitteln
         DO i=n, L
          IF (ABS(M(p(i),n)) > piv) THEN
           piv = ABS(M(p(i),n))
           ipiv = i
          END IF
         END DO

         aux     = p(n)                     !Hilfsvariable beladen
         p(n)    = p(ipiv)                  !Vertauschen im Indexfeld
         p(ipiv) = aux                      !Vertauschung abeschliessen

         DO i = n+1, L
          f = M(p(i),n)/M(p(n),n)
          M(p(i),n) = f                     !Faktoren statt 0 schreiben
          DO j = n+1, L
           M(p(i),j) = M(p(i),j)-f*M(p(n),j)
          END DO
         END DO

        END DO                              !Ende Masterloop

END SUBROUTINE ge

SUBROUTINE resubst (B, M, p, X)             !Ruecksubstitution

        IMPLICIT NONE

        REAL(KIND=8), INTENT(INOUT) :: B(:, :)
        REAL(KIND=8), INTENT(INOUT) :: M(:, :)
        REAL(KIND=8), INTENT(INOUT) :: X(:, :)
        INTEGER, INTENT(INOUT) :: p(:)

        INTEGER n, i, j, L, LB, ib

        L  =  SIZE(M, DIM=1)                !# der Zeilen
        LB =  SIZE(B, DIM=2)                !# der Gleichungssysteme


        DO n=1, L-1                         !Eliminationsschritte fuer rechte Seite B
         DO i=n+1,L
          DO j=1, LB
           B(p(i),j)=B(p(i),j)-M(p(i),n)*B(p(n),j)
          END DO
         END DO
```

```
        END DO

        DO ib=1,LB, 1                    !Masterloop ueber Spalten von B
         DO n=L,1,-1
          X(n,ib)=B(p(n),ib)
          DO j=n+1, L
           X(n,ib)=X(n,ib)-M(p(n),j)*X(j,ib)
          END DO
          X(n,ib)=X(n,ib)/M(p(n),n)
         END DO
        END DO
END SUBROUTINE  resubst

SUBROUTINE ausgabe (a)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(IN) :: a(:, :)

        INTEGER :: i, j
        DO i=1, SIZE(a, DIM=1)
         DO j=1, SIZE(a, DIM=2)
          WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
         END DO
         WRITE(*,*)
        END DO
        PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
        USE LinAg
        IMPLICIT none
        INTEGER :: N=100
        REAL(KIND=8) :: M(16,16), B(16,100)      !Matrix M und B
        REAL(KIND=8) :: X(16,100)                !Matrix X
        REAL(KIND=8) :: f, t                     !Kraft f, Zeit t
        REAL(KIND=8) :: a1, a2, a3               !Argumente a1, a2, a3
        INTEGER :: p(16)                         !Permutationsvektor
        INTEGER :: i, j, k                       !Schleifenzaehler
        M       =  0.d0                          !Befuellen der Matrix mit 0
        M(1,2)  =  1.d0                          !Codieren der Gleichungen des Fachwerkes
        M(1,3)  =  1.d0
        M(1,7)  =  1.d0/(sqrt(2.d0))
        M(2,1)  =  1.d0
        M(2,7)  = -1.d0/(sqrt(2.d0))
        M(3,3)  =  1.d0
        M(3,4)  =  1.d0
        M(4,8)  = -1.d0
        M(5,4)  =  1.d0
        M(5,5)  =  1.d0
        M(5,9)  = -1.d0/(sqrt(2.d0))
        M(5,11) =  1.d0/(sqrt(2.d0))
        M(6,9)  = -1.d0/(sqrt(2.d0))
        M(6,10) = -1.d0
        M(6,11) = -1.d0/(sqrt(2.d0))
        M(7,5)  =  1.d0
        M(7,6)  =  1.d0
        M(8,12) = -1.d0
        M(9,6)  =  1.d0
        M(9,13) = -1.d0/(sqrt(2.d0))
        M(10,13)= -1.d0/(sqrt(2.d0))
        M(10,14)=  1.d0
        M(11,7) =  1.d0/(sqrt(2.d0))
        M(11,9) = -1.d0/(sqrt(2.d0))
        M(11,16)=  1.d0
        M(12,7) = -1.d0/(sqrt(2.d0))
```

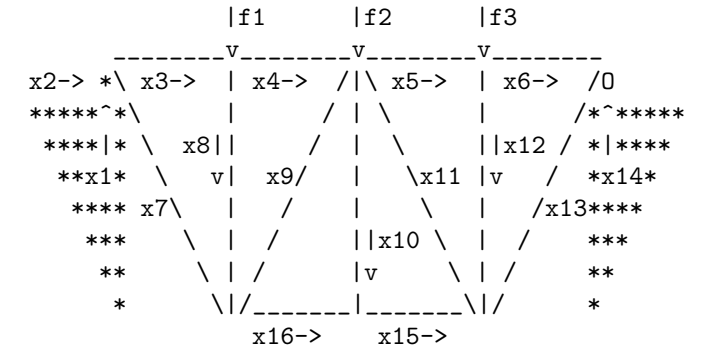
```
M(12,8)  = -1.d0
M(12,9)  = -1.d0/(sqrt(2.d0))
M(13,15) =  1.d0
M(13,16) =  1.d0
M(14,10) =  1.d0
M(15,11) =  1.d0/(sqrt(2.d0))
M(15,13) = -1.d0/(sqrt(2.d0))
M(15,15) =  1.d0
M(16,11) = -1.d0/(sqrt(2.d0))
M(16,12) = -1.d0
M(16,13) = -1.d0/(sqrt(2.d0))


B = 0.d0                                          !Befuellen des Vektors mit 0
f = 0.d0                                          !Null setzen der Belastung

PRINT *, "Dieses Programm loest das Gleichungssystem fuer ein einfaches Fachwerk."
PRINT *
PRINT *
PRINT *, "                |f1       |f2       |f3"
PRINT *, "         --------v--------v--------v--------"
PRINT *, " x2-> *\ x3->   | x4->   /|\ x5->   | x6->   /O"
PRINT *, " *****^*\       |       /  |  \      |       /*^*****"
PRINT *, "  ****|* \  x8||      /   |   \     ||x12  / *|****"
PRINT *, "   **x1*  \    v|  x9/    |    \x11  |v    /  **x14*"
PRINT *, "    **** x7\    |   /     |     \    |   /x13****"
PRINT *, "     ***     \  |  /      ||x10 \   |  /     ***"
PRINT *, "      **      \ | /       |v      \  | /      **"
PRINT *, "       *       \|/--------|--------\|/       *"
PRINT *, "                  x16->     x15->"
PRINT *
PRINT *, "wobei * ein festes, und O ein nicht festes Auflager ist."
PRINT *, "Alle Kraefte in den Diagonalen weisen entlang diesen nach unten."
PRINT *, "Das Problem wird einmal symmetrisch belastet geloest, einmal"
PRINT *, "asymmetrisch belastet geloest, und zuletzt dynamisch belastet."
PRINT *, "Die Ausgabe der Loesungen erfolgt jeweils in die Dateien"
PRINT *, "sym.dat, asym.dat, sowie dyn.dat, sodass eine weitere Ver-"
PRINT *, "arbeitung mit z.B. xmgrace moeglich ist."
PRINT *
PRINT *
PRINT *,"Vorbereiten des Vektors B fuer symmetrische Belastung. Dabei ist"
PRINT *,"f1=f/3, f2=f/3, f3=f/3, f aus [0,1] mit Increment 0.01."

DO i=1, N                                         !Symmetrische Belastung vorbereiten
 B(4,i) = f/3.d0;
 B(6,i) = f/3.d0;
 B(8,i) = f/3.d0;
 f = f + 0.01
END DO

CALL ge (M, p)                                    !Gauss-Elimination
CALL resubst(B, M, p, X)                          !Resubstitution

OPEN (UNIT=10, FILE="sym.dat", ACTION="write")    !Stream fuer File
f = 0.d0                                          !Kraft Null setzen
DO j=1, N
 DO k=1, 16
  WRITE (10,'(F8.5)',ADVANCE="NO")(abs(X(k,j)))   !Schreibe Daten auf File
 END DO
 WRITE (10,'(F8.5)', ADVANCE="NO") f              !Schreibe Kraft
 WRITE (10,*)                                     !Naechste Zeile
 f = f + 0.01                                     !erhoehe Kraft
END DO
CLOSE (UNIT=10)                                   !Beende Stream

PRINT *
PRINT *, "Symmetrischen Fall berechnet. Daten in sym.dat."
```

```
PRINT *
PRINT *
PRINT *,"Vorbereiten des Vektors B fuer asymmetrische Belastung. Dabei ist"
PRINT *,"f1=f*0.2, f2=f*0.3, f3=f*0.5, f aus [0,1] mit Increment 0.01."
PRINT *

B = 0.d0                                          !Ruecksetzen der B-Matrix
f = 0.d0                                          !Rucksetzen der Kraft

DO i=1, N                                         !Asymmetrische Belastung vorbereiten
 B(4,i) = f*0.2;
 B(6,i) = f*0.3;
 B(8,i) = f*0.5;
 f = f + 0.01
END DO

CALL resubst(B, M, p, X)                          !Resubstitution

OPEN (UNIT=20, FILE="asym.dat", ACTION="write")   !Stream fuer File
f = 0.d0                                          !Kraft Null setzen
DO j=1, N
 DO k=1, 16
  WRITE (20,'(F8.5)',ADVANCE="NO")(abs(X(k,j)))   !Schreibe Daten auf File
 END DO
 WRITE (20,'(F8.5)', ADVANCE="NO") f              !Schreibe Kraft
 WRITE (20,*)                                     !Naechste Zeile
 f = f + 0.01                                     !erhoehe Kraft
END DO
CLOSE (UNIT=20)                                   !Beende Stream

PRINT *, "Asymmetrischen Fall berechnet. Daten in asym.dat."
PRINT *
PRINT *
PRINT *,"Vorbereiten des Vektors B fuer dynamische Belastung. Dabei ist"
PRINT *,"f1=EXP(-(t-3)^2, f2=EXP(-(t-5)^2, f3=EXP(-(t-7)^2), t aus [0,10]"
PRINT *,"mit Increment 0.1."
PRINT *

B  =  0.d0                                        !Ruecksetzen der B-Matrix
t  =  0.d0                                        !Null setzen der Zeit
a1 =  0.d0                                        !Null setzen von Argument a1
a2 =  0.d0                                        !Null setzen von Argument a2
a3 =  0.d0                                        !Null setzen von Argument a3

DO i=1, N                                         !Dynamische Belastung vorbereiten
 a1 = -(t-3.d0)*(t-3.d0)                          !Berechne Argument 1
 a2 = -(t-5.d0)*(t-5.d0)                          !Berechne Argument 2
 a3 = -(t-7.d0)*(t-7.d0)                          !Berechne Argument 3
 B(4,i) = EXP(a1)
 B(6,i) = EXP(a2)
 B(8,i) = EXP(a3)
 t = t + 0.1
END DO

CALL resubst(B, M, p, X)                          !Resubstitution

OPEN (UNIT=30, FILE="dyn.dat", ACTION="write")    !Stream fuer File
t = 0.d0                                          !Zeit Null setzen
DO j=1, N
 DO k=1, 16
  WRITE (30,'(F8.5)',ADVANCE="NO")(abs(X(k,j)))   !Schreibe Daten auf File
 END DO
 WRITE (30,'(F8.5)', ADVANCE="NO") t              !Schreibe Zeit
 WRITE (30,*)                                     !Naechste Zeile
 t = t + 0.1                                      !erhoehe Zeit
END DO
CLOSE (UNIT=30)                                   !Beende Stream
```

```
        PRINT *, "Dynamischer Fall berechnet. Daten in dyn.dat."
        PRINT *
END PROGRAM main
```

8.4.2 Output des Programmes

```
Dieses Programm loest das Gleichungssystem fuer ein einfaches Fachwerk.
```




```
wobei * ein festes, und O ein nicht festes Auflager ist.
Alle Kraefte in den Diagonalen weisen entlang diesen nach unten.
Das Problem wird einmal symmetrisch belastet geloest, einmal
asymmetrisch belastet geloest, und zuletzt dynamisch belastet.
Die Ausgabe der Loesungen erfolgt jeweils in die Dateien
sym.dat, asym.dat, sowie dyn.dat, sodass eine weitere Ver-
arbeitung mit z.B. xmgrace moeglich ist.


Vorbereiten des Vektors B fuer symmetrische Belastung. Dabei ist
f1=f/3, f2=f/3, f3=f/3, f aus [0,1] mit Increment 0.01.

Symmetrischen Fall berechnet. Daten in sym.dat.


Vorbereiten des Vektors B fuer asymmetrische Belastung. Dabei ist
f1=f*0.2, f2=f*0.3, f3=f*0.5, f aus [0,1] mit Increment 0.01.

Asymmetrischen Fall berechnet. Daten in asym.dat.


Vorbereiten des Vektors B fuer dynamische Belastung. Dabei ist
f1=EXP(-(t-3)^2, f2=EXP(-(t-5)^2, f3=EXP(-(t-7)^2), t aus [0,10]
mit Increment 0.1.

Dynamischer Fall berechnet. Daten in dyn.dat.
```

8.5 Übung 5: LU Zerlegung

Diese Übung ist eine Implementierung des LU Verfahrens und Lösung eines Gleichungssystemes mit Rücksubstitution. Die entsprechenden Gleichungen sind im Theorieteil nachzulesen.

8.5.1 Sourcecode - Uebung5.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS


SUBROUTINE lu (A,L,U)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: A(:,:)
        REAL(KIND=8), INTENT(INOUT) :: L(:,:)
        REAL(KIND=8), INTENT(INOUT) :: U(:,:)
        REAL(KIND=8) :: sum
        INTEGER :: i,j,k                                    !Zaehlparameter
        sum = 0.d0

        DO i=1, SIZE(A,DIM=1)
         DO j=i, SIZE(A,DIM=1)                              !Bilde Summe
          DO k=1, i-1
           sum = sum + L(i,k)*U(k,j)
          END DO
          U(i,j) = A(i,j) - sum                             !Berechne Element v. U
          sum = 0.d0                                        !Setze Summe zurueck
          DO k=1, i-1
           sum = sum + U(k,i)*L(j,k)                        !Bilde Summe
          END DO
          L(j,i) = (A(j,i)-sum)/U(i,i)                      !Berechne Element v. L
          sum = 0.d0                                        !Setze Summe zurueck
         END DO
        END DO
END SUBROUTINE lu


SUBROUTINE solve (A,L,U,x,y,b)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: A(:,:)
        REAL(KIND=8), INTENT(INOUT) :: L(:,:)
        REAL(KIND=8), INTENT(INOUT) :: U(:,:)
        REAL(KIND=8), INTENT(INOUT) :: x(:)
        REAL(KIND=8), INTENT(INOUT) :: y(:)
        REAL(KIND=8), INTENT(INOUT) :: b(:)
        REAL(KIND=8) :: sum
        INTEGER :: i,j                                      !Zaehlparameter
        sum = 0.d0                                          !Setze Summe 0

        DO i=1, SIZE(A,DIM=1)
         DO j=1, i-1
          sum = sum + L(i,j)*y(j)                           !Bilde Summe
         END DO
         y(i) = b(i) - sum                                  !Bereche y-Elemente
         sum = 0.d0                                         !Setze Summe zurueck
        END DO

        DO i=SIZE(A,DIM=1), 1, -1                           !Schleife laeuft verkehrt
         DO j=i+1, SIZE(A, DIM=1)
          sum = sum + U(i,j)*x(j)                           !Berechne Summe
         END DO
         x(i) = (y(i) - sum)/u(i,i)                         !Berechne x-Elemente
         sum = 0.d0                                         !Setze Summe zurueck
```

```
        END DO
END SUBROUTINE solve

SUBROUTINE ausgabe (a)                                  !Ausgabe fuer Matrizen
        IMPLICIT NONE
        REAL(KIND=8), INTENT(IN) :: a(:, :)

        INTEGER :: i, j
        DO i=1, SIZE(a, DIM=1)
         DO j=1, SIZE(a, DIM=2)
          WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
         END DO
         WRITE(*,*)
        END DO
        PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
        USE LinAg
        IMPLICIT none
        REAL(KIND=8) ::  A(2,2), b(2)                   !Matrix A und Vektor b
        REAL(KIND=8) :: L(2,2), U(2,2)
        REAL(KIND=8) :: x(2)                            !Vektor x
        REAL(KIND=8) :: y(2)                            !Vektor y
        INTEGER :: i
        A(1,1) = 3.d0                                   !Befuellen der Matrix
        A(1,2) = -1.d0
        A(2,1) = -1.d0
        A(2,2) = 2.d0
        B(1) = 1.d0                                     !Befuellen des Vektors
        B(2) = 3.d0
        U = 0.d0                                        !Null setzen von L und U
        L = 0.d0
        x = 0.d0                                        !Null setzen von x und y
        y = 0.d0
        PRINT *, "Dieses Programm zerlegt die Matrix A in A=LU"
        PRINT *, "und loest anschliessend durch Resubstitution."
        PRINT *
        CALL lu(A,L,U)                                  !Rufe Zerlegung
        PRINT *,"A = "
        CALL ausgabe (A)
        PRINT *,"L = "
        CALL ausgabe (L)
        PRINT *,"U = "
        CALL ausgabe (U)
        CALL solve (A,L,U,x,y,b)                        !Rufe Resubstitution
        PRINT *,"Die Loesung ist:"
        PRINT *,"x = "
        DO i=1, SIZE(x,DIM=1)                           !Gebe Loesung aus
         PRINT *,x(i)
        END DO

END PROGRAM main
```

8.5.2 Output des Programmes

```
 Dieses Programm zerlegt die Matrix A in A=LU
 und loest anschliessend durch Resubstitution.

 A =
 3.00000-1.00000
-1.00000 2.00000

 L =
 1.00000 0.00000
-0.33333 1.00000

 U =
 3.00000-1.00000
 0.00000 1.66667

 Die Loesung ist:
 x =
   1.00000000000000
   2.00000000000000
```

8.6 Übung 6: SOR Verfahren

In dieser Übung wird das SOR Verfahren implementiert. Der Relaxationsparameter ω liegt dabei im Intervall $(0, 2)$, das Abbruchkriterium ist $||x^{(n+1)} - x^{(n)}|| < \epsilon$. Das Programm arbeitet mit einer Matrix A, welche $N \times N$, und wie folgt gebaut ist:

- Die Hauptdiagonalelemente haben den Wert N
- Alle anderen Elemente werden zufällig aus $[0, 1]$ gewählt

Für dieses Programm habe ich vorerst $N = 10$ gesetzt. Nach der Anwendung des SOR Verfahrens wird die Lösung ausgegeben, zur Kontrolle wird $A.x$ berechnet, welches natürlich b ergeben soll, wobei b aus Gründen der Praktikabilität zuerst ebenfalls mit Zufallswerten zwischen 0 und 1 gefüllt wurde. Danach wird für dieses Problem ω von 0 bis 2 variiert, die Anzahl der nötigen Iterationsschritte wird festgehalten und zu jedem ω in eine Datei `runs.dat` geschrieben.

8.6.1 Sourcecode - Uebung6.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS

SUBROUTINE SOR (A,x,b,omega,runs)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: A(:,:)
        REAL(KIND=8), INTENT(INOUT) :: x(:)
        REAL(KIND=8), INTENT(INOUT) :: b(:)
        REAL(KIND=8), INTENT(IN) :: omega
        REAL(KIND=8) :: sumless, sumgreat, epsilon, abbruch
        REAL(KIND=8) :: x0(SIZE(A,DIM=1))
        INTEGER, INTENT(INOUT) :: runs

        INTEGER :: i,j,k, N                                  !Zaehlparameter

        sumless = 0.d0                                       !Summe Null setzen
        sumgreat = 0.d0                                      !Summe Null setzen
        epsilon = 0.0001                                     !Epsilon setzen
        x0 = 0.2                                             !Abbruchbedingung
        x = 0.d0

        N = SIZE(A, DIM=1)                                   !N bestimmen

        abbruch = 0.d0                                       !Abbruch 0 setzen
        DO i=1, N                                            !Abbruch berechnen
         abbruch = abbruch + (x(i)-x0(i))*(x(i)-x0(i))
        END DO
        abbruch = SQRT(abbruch)

        DO WHILE ( abbruch >= epsilon )                      !Beginne Iteration

         DO i=1, N

          DO j=1, i-1                                        !erste Summe
           sumless = sumless + A(i,j)*x(j)
          END DO

          DO j=i+1, N                                        !zweite Summe
           sumgreat = sumgreat + A(i,j)*x0(j)
          END DO

          x(i)= omega/A(i,i)*(b(i)-sumless - sumgreat)+(1-omega)*x0(i)
```

```
            sumless = 0.d0                                  !Summen ruecksetzen
            sumgreat = 0.d0
           END DO

           abbruch = 0.d0
           DO k=1, N
            abbruch = abbruch + (x(k)-x0(k))*(x(k)-x0(k))
           END DO
           abbruch = SQRT(abbruch)
           runs = runs + 1
           x0 = x
          END DO
END SUBROUTINE SOR

SUBROUTINE ausgabe (a)                                      !Ausgabe fuer Matrizen
          IMPLICIT NONE
          REAL(KIND=8), INTENT(IN) :: a(:, :)

          INTEGER :: i, j
          DO i=1, SIZE(a, DIM=1)
           DO j=1, SIZE(a, DIM=2)
            WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
           END DO
           WRITE(*,*)
          END DO
          PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
          USE LinAg
          IMPLICIT none
          INTEGER, PARAMETER :: N = 10                      !Dimension des Problemes
          REAL(KIND=8) ::  A(N,N), b(N), c(N)               !Matrix A und Vektor b
          REAL(KIND=8) :: x(N)                              !Vektor x
          REAL(KIND=8) :: omega                             !Streckparameter
          INTEGER :: runs                                   !Anzahl der Iterationen
          INTEGER :: i,j                                    !Zaehlparameter

          DO i=1, N                                         !Matrix zufaellig fuellen
           DO j=1, N
            CALL RANDOM_NUMBER(A(i,j))
           END DO
          END DO

          DO i=1, N                                         !Hauptdiagonale fuellen
           A(i,i) = N
           CALL RANDOM_NUMBER(b(i))                         !Zielvektor fuellen
          END DO

          x = 0.d0
          c = 0.d0
          omega = 1.1
          runs = 0

          PRINT *, "Dieses Programm loest das System Ax = b durch SOR."
          PRINT *

          CALL SOR(A,x,b,omega,runs)

          PRINT *,"A = "
          CALL ausgabe (A)
          PRINT *
          PRINT *,"b = "
```

```
        DO i=1, N
         PRINT *, b(i)
        END DO
        PRINT *
        PRINT *,"x = "
        DO i=1, N
         PRINT *, x(i)
        END DO
        PRINT *
        DO i=1, N
         DO j=1, N
          c(i) = c(i) + A(i,j)*x(j)
         END DO
        END DO
        PRINT *,"A.x="
        DO i=1, N
         PRINT *, c(i)
        END DO

        x = 0.d0                                          !Null setzen von x
        omega = 0.0                                       !Omega setzen
        runs = 0                                          !Zahl d. Iterationen 0 setzen

        PRINT *
        PRINT *, "Variiere omega nun von 0 bis 2 und schreibe Anzahl der"
        PRINT *, "Iterationen auf File runs.dat."
        PRINT *

        OPEN(UNIT=20, FILE="runs.dat", ACTION="write")
        DO i=1, 99
         omega = omega + 0.02
         CALL SOR (A,x,b,omega,runs)                          !SOR rufen
         WRITE(20,'(F8.5)', ADVANCE="NO") omega
         WRITE(20, '(I8)') runs
         runs = 0
        END DO
        CLOSE (UNIT=20)

        PRINT *,"Fertig!"
        PRINT *
END PROGRAM main
```

8.6.2 Output des Programmes

```
 Dieses Programm loest das System Ax = b durch SOR.

 A =
10.00000 0.56682 0.96592 0.74793 0.36739 0.48064 0.07375 0.00536 0.34708 0.34224
 0.2179510.00000 0.90052 0.38677 0.44548 0.66193 0.01611 0.65085 0.64641 0.32299
 0.85569 0.4012910.00000 0.96854 0.59840 0.67298 0.45688 0.33002 0.10038 0.75545
 0.60569 0.71905 0.8973310.00000 0.15072 0.61231 0.97866 0.99914 0.25680 0.55087
 0.65905 0.55401 0.97776 0.9019210.00000 0.72886 0.40246 0.92863 0.14784 0.67453
 0.76961 0.33932 0.11582 0.61437 0.8206210.00000 0.73113 0.49760 0.37480 0.42151
 0.55290 0.99792 0.99039 0.74631 0.95376 0.0932710.00000 0.75176 0.94685 0.70618
 0.81381 0.55859 0.06171 0.48038 0.59769 0.13753 0.5874010.00000 0.88588 0.30381
 0.66966 0.66494 0.50368 0.26158 0.07656 0.10125 0.54927 0.3755810.00000 0.79292
 0.62088 0.77360 0.95358 0.11424 0.31846 0.59682 0.04815 0.11421 0.2159610.00000


 b =
  7.334180249993905E-002
  0.246861739727688
  0.443384266796283
  0.208367573254460
  0.566998342734826
  2.431239969651688E-002
  0.420290577071217
  0.397853024116983
  0.976585425492088
  0.692605032992832

 x =
 -3.580363434601721E-003
  1.072050083774652E-002
  3.430890457834038E-002
  7.135803609459092E-003
  4.450261546325729E-002
 -1.069150408907612E-002
  1.841710880159257E-002
  2.563150778095614E-002
  8.812972216090180E-002
  6.223664602317071E-002

 A.x=
  7.334458605292137E-002
  0.246877169322753
  0.443410664036121
  0.208394446477588
  0.567018274308642
  2.432831113577354E-002
  0.420306420528633
  0.397861504976695
  0.976590927469712
  0.692606998078928

 Variiere omega nun von 0 bis 2 und schreibe Anzahl der
```

```
Iterationen auf File runs.dat.

Fertig!
```

8.7 Übung 7: Hubbardmodell Matrixelemente

In dieser Übung sollen die Matrixelemente des Hubbardmodelles, wie es in Kapitel 4 beschrieben wurde, berechnet werden.
Der Hamiltonoperator ist schon in Gleichung (4.36) beschrieben und diente als Ausgangspunkt. Die Terme die aus Anzahloperatoren bestehen sind sehr leicht umzusetzen, da sie den Zustand reproduzieren oder 0 liefern. Die Programmierung ist daher einfach. Weniger einfach allerdings sind die Beiträge der Hoppingterme. Zunächst wurde eine Codierung der Basiszustände vorgenommen. Es gilt:

$$b = (i \cdot 1 + j \cdot 10) + (k \cdot 100 + l \cdot 1000) \tag{8.10}$$

Dabei steht i für die Position des ersten Teilchens, j für den Spin des ersten Teilchens, k für die Position des zweiten Teilchens und l für den Spin des zweiten Teilchens. Spin up wurde dabei mit 1 und Spin down mit 2 definiert. Die Positionen können die Werte 1, 2 und 3 annehmen. Beispiel:

$$(2 \cdot 1 + 2 \cdot 10) + (3 \cdot 100 + 2 \cdot 1000) = 2322 \tag{8.11}$$

repräsentiert das erste Teilchen an der Position 2 mit Spin down und das zweite Teilchen an Position 3 und Spin down.
Das Konzept wurde ebenfalls für die Codierung der Hoppingterme übernommen. Dabei ist es völlig ausreichend, denn die Information ob es sich um einen Vernichter oder Erzeuger handelt ist durch dessen Position schon vorgegeben. Man kann nun den allgemeinsten Fall der Hoppingterme wie folgt anschreiben:

$$\begin{aligned} & a_i^{\alpha\dagger} a_j^{\alpha} a_k^{\beta\dagger} a_l^{\gamma\dagger} |0\rangle \\ = & a_i^{\alpha\dagger} a_l^{\gamma\dagger} \delta_{kj} \delta_{\alpha\beta} |0\rangle - a_i^{\alpha\dagger} a_k^{\beta\dagger} \delta_{jl} \delta_{\gamma\alpha} |0\rangle \end{aligned} \tag{8.12}$$

Dies folgt zwanglos durch einfaches ausrechnen. Da die Hoppingterme stets mit selbem Spin kommen, brauchen wir nur einen Spinindex α für den Hamiltonbeitrag. Ebenfalls kann man nun leicht zeigen, dass die beiden Terme exklusiv sind, dh. sie schließen einander aus: Ist der erste Term ungleich Null so ist der zweite gleich Null und umgekehrt. Dann ist die Wirkung des Hoppingtermes durch eine einfache Fallunterscheidung zu implementieren. Wir wollen zeigen, dass die Terme exklusiv sind: Wir greifen uns die Indices k und l heraus.

Fall 1: $k = l$

Dann muss aber $\beta \neq \gamma$ sein, da wir es mit Fermionen zu tun haben. Ein Blick auf die Kroneckerdeltas in den beiden Termen zeigt uns dann, dass $\delta_{kj}\delta_{\alpha\beta}$ und $\delta_{jl}\delta_{\gamma\alpha}$ nicht gleichzeitig erfüllt werden kann, da entweder $\beta = \alpha$ oder $\gamma = \alpha$ ist. Dieser Fall führt also auf Exklusivität.

Fall 2: $k \neq l$

Wieder kann $\delta_{kj}\delta_{\alpha\beta}$ und $\delta_{jl}\delta_{\gamma\alpha}$ nicht gleichzeitig erfüllt werden, da entweder $j = k$ oder $j = l$ ist. Damit ist die Exklusivität gezeigt.

Damit ist das Problem relativ einfach zu programmieren.

8.7.1 Sourcecode - Uebung7.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS

FUNCTION akreuza (brastate, contrib, state)             !Hoppingterm
        IMPLICIT NONE
        INTEGER, INTENT(INOUT) :: contrib, brastate, state
        INTEGER :: akreuza
        INTEGER cpos1, cspi1, cpos2, cspi2              !Vernichterindices 2
        INTEGER spos1, sspi1, spos2, sspi2              !Basisindices
        INTEGER newstate1, newstate2                    !neuer Zustand

        newstate1 = 0                                   !neuen Zust. Null setzen
        newstate2 = 0

        CALL decode (contrib,cpos1,cspi1,cpos2,cspi2)   !Decodierung contrib
        CALL decode (state,spos1,sspi1,spos2,sspi2)     !Decodierung state

        !geleistete Vorarbeit: Ich habe allgemein gezeigt, dass nur in zwei exclusiven
        !Faellen ein Beitrag entsteht:
        !Fall 1: cpos2==spos1 AND cspi2==sspi1:
        !        spos1=cpos1, sspi1=cspi1, spos2=spos2, sspi1=sspi1 ist neuer State
        !Fall 2: cpos2==spos2 AND cspi2==sspi2:
        !        spos2=spos1, sspi2=sspi1, spos1=cpos1, sspi1=cspi1 ist neuer State

        IF((cpos2 .EQ. spos1).AND.(cspi2 .EQ. sspi1)) THEN  !Zustand ueberpruefen
         spos1 = cpos1                                  !Fall 1
         sspi1 = cspi1
         CALL code (newstate1, spos1, sspi1, spos2, sspi2)  !Neuer Zustand
         CALL code (newstate2, spos2, sspi2, spos1, sspi1)  !Neuer Zustand
         newstate2 = -newstate2                                 !Vorzeichen wechselt
        ELSE IF((cpos2 .EQ. spos2).AND.(cspi2 .EQ. sspi2)) THEN
         spos2 = spos1
         sspi2 = sspi1
         spos1 = cpos1
         sspi1 = cspi1
         CALL code (newstate1, spos1, sspi1, spos2, sspi2)   !Neuer Zustand
         CALL code (newstate2, spos2, sspi2, spos1, sspi1)  !Neuer Zustand
         newstate2 = -newstate2                             !Vorzeichen wechselt
         newstate1 = -newstate1                             !Beitrag kommt negativ
         newstate2 = -newstate2
        ELSE
         spos1 = 0
         sspi1 = 0
         spos2 = 0
         sspi2 = 0
         CALL code (newstate1, spos1, sspi1, spos2, sspi2)   !Neuer Zustand
         CALL code (newstate2, spos2, sspi2, spos1, sspi1)   !Neuer Zustand
         newstate2 = -newstate2                             !Vorzeichen wechselt
        END IF
        IF (brastate .EQ. ABS(newstate1)) THEN
         akreuza = newstate1/ABS(newstate1)
        ELSE IF(brastate .EQ. ABS(newstate2)) THEN
         akreuza = newstate2/ABS(newstate2)
        ELSE
         akreuza = 0
        END IF
END FUNCTION akreuza

FUNCTION anzahl(brastate,state)                         !B-Term
        IMPLICIT NONE
        INTEGER, INTENT(INOUT) ::  brastate, state
        INTEGER :: n,anzahl, i
        INTEGER spos1, sspi1, spos2, sspi2              !Basisindices
```

```
        n=0                                                    !Anzahl Null setzen

        CALL decode (state,spos1,sspi1,spos2,sspi2)            !Decodierung state

        !Ganz allgemein gilt:
        !Es wird nur ein Beitrag erhalten, wenn das Tupel
        !des Anzahloperators in dem State vorkommt

        DO i=1, 3
         IF((i .EQ. spos1) .AND. (1 .EQ. sspi1)) THEN
          n = n+1
         ELSE IF((i .EQ. spos2) .AND. (1 .EQ. sspi2)) THEN
          n = n+1
         END IF
        END DO

        IF (brastate .EQ. state) THEN
         anzahl = n
        ELSE
         anzahl = 0
        END IF
END FUNCTION anzahl

FUNCTION anzahl1(brastate,state)                               !U-Beitrag

        IMPLICIT NONE
        INTEGER, INTENT(INOUT) ::  brastate, state
        INTEGER :: n,anzahl1, i
        INTEGER spos1, sspi1, spos2, sspi2                     !Basisindices

        n=0                                                    !Anzahl Null setzen

        CALL decode (state,spos1,sspi1,spos2,sspi2)            !Decodierung state

        !Ganz allgemein gilt:
        !Es wird nur ein Beitrag erhalten, wenn antiparallele
        !Spins an einem Ort auftreten

        IF((spos1 .EQ. spos2) .AND. (sspi1 .NE. sspi2)) THEN
         n = 1
        END IF

        IF (brastate .EQ. state) THEN
         anzahl1 = n
        ELSE
         anzahl1 = 0
        END IF
END FUNCTION anzahl1

SUBROUTINE decode (x, a, b, c, d)                              !Zustand decodieren:
        IMPLICIT NONE
        INTEGER, INTENT(IN) :: x                               !erhalte Indices
        INTEGER, INTENT(INOUT) :: a, b, c, d
        INTEGER  x1, x2, x3
        a = MOD(x,10)
        x1 = (x-a)/10
        b = MOD(x1,10)
        x2 = (x1-b)/10
        c = MOD(x2,10)
        x3 = (x2-c)/10
        d = MOD(x3,10)
END SUBROUTINE decode

SUBROUTINE code (x, a, b, c, d)                                !Zustand codieren:
        IMPLICIT NONE
        INTEGER, INTENT(INOUT) :: x                            !erhalte Zahl
```

```
        INTEGER, INTENT(IN) :: a, b, c, d
        x = a + 10*b + 100*c + 1000*d
END SUBROUTINE code

SUBROUTINE ausgabe (a)                                  !Ausgabe fuer Matrizen
        IMPLICIT NONE
        REAL(KIND=8), INTENT(IN) :: a(:, :)

        INTEGER :: i, j
        DO i=1, SIZE(a, DIM=1)
         DO j=1, SIZE(a, DIM=2)
          WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
         END DO
         WRITE(*,*)
        END DO
        PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
        USE LinAg
        IMPLICIT none
        INTEGER, PARAMETER :: N = 15            !Dimension des Problemes
        REAL(KIND=8) :: t                       !Parameter t
        REAL(KIND=8) :: U                       !Parameter U
        REAL(KIND=8) :: B                       !Parameter B
        INTEGER :: b1(N)                        !Basissatz
        INTEGER :: h1(12)                       !Hoppingteil des Hamilton
        INTEGER :: state, brastate              !aktueller Zustand u. Brazust.
        INTEGER :: contrib, cont                !Operatorbeitrag des Hamilton
        REAL(KIND=8) ::  H(N,N)                 !Matrix H
        INTEGER :: i,j,k                        !Zaehlparameter

        t = 1.d0                                !Setzen von t
        U = 1.d0                                !Setzen von U
        B = 1.d0                                !Aeusseres Feld B

        H  = 0.d0                               !Matrix Null setzen
        b1 = 0
        h1 = 0

        b1(1)  =  2111                          !Eincodieren der Basiszustaende:
        b1(2)  =  2212                          !Es gilt:
        b1(3)  =  2313                          !(i*1+j*10)+(k*100+l*1000)
        b1(4)  =  2211                          !Die erste Klammer steht fuer
        b1(5)  =  2311                          !das erste Teilchen, die zweite fuer
        b1(6)  =  2312                          !das zweite. Spin up = 1, Spin dwn = 2
        b1(7)  =  1221                          !und es gibt Positionen 1,2 und 3.
        b1(8)  =  1321                          !i und k sind Positionsparameter,
        b1(9)  =  1322                          !j und l Spinparameter. Zum Beispiel
        b1(10) =  1211                          !ist der Basiszustand 15 gegeben durch
        b1(11) =  1311                          !erstes Teilchen: Pos 2, Spin Down und
        b1(12) =  1312                          !zweites Teilchen: Pos 3, Spin Down. Dann:
        b1(13) =  2221                          !(2*1+2*10)+(3*100+2*1000) = 2322
        b1(14) =  2321
        b1(15) =  2322

        h1(1)  =  1211                          !Die Codierung erfolgt gleich wie oben,
        h1(2)  =  1312                          !wobei der erste Teil immer fuer den
        h1(3)  =  1113                          !(links stehenden) Erzeuger, der zweite
        h1(4)  =  2221                          !Teil immer fuer den Vernichter steht.
        h1(5)  =  2322                          !Der Vernichter ist also stets zuerst
        h1(6)  =  2123                          !anzuwenden, hernach der Erzeuger auf den
        h1(7)  =  1112                          !Zustand.
        h1(8)  =  1213
```

```
        h1(9)  =  1311
        h1(10) =  2122
        h1(11) =  2223
        h1(12) =  2321



        !Berechne die Matrixelemente von H
        cont = 0                                  !Beitrag Null setzen

        DO i=1, N                                 !Durchlaufe Bras
         DO j=1, N                                !Durchlaufe Kets
           brastate = b1(i)                       !Bra auswaehlen
           state = b1(j)                          !Ket auswaehlen
           DO k=1, 12                             !Durchlaeuft Beitrage von H-hopping
            contrib = h1(k)                       !Beitraege von H-hopping auswaehlen
            cont = cont + akreuza(brastate, contrib, state)
           END DO
           cont = -t*cont                         !Beitrag mit -t multiplizieren
           H(i,j) = cont                          !Matrixelement ist fertig
           cont = 0                               !Beitrag ruecksetzen
           cont = B*anzahl(brastate,state)        !B-Beitraege ermitteln
           H(i,j) = H(i,j) + cont                 !Matrixelement aktualisieren
           cont = 0                               !Beitrag ruecksetzen
           cont = U*anzahl1(brastate,state)       !U-Beitraege ermitteln
           H(i,j) = H(i,j) + cont                 !Matrixelement aktualisieren
           cont = 0                               !Beitrag ruecksetzen
         END DO
        END DO


        CALL ausgabe(H)                           !Matrix H ausgeben

        !state = b1(7)                            !Test
        !contrib = h1(1)
        !brastate = b1(1)
        !cont =  akreuza (brastate, contrib, state)
        !PRINT *, cont
END PROGRAM main
```

8.7.2 Output des Programmes

```
 2.00  0.00  0.00 -1.00 -1.00  0.00  1.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  2.00  0.00 -1.00  0.00 -1.00  1.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  2.00  0.00 -1.00 -1.00  0.00  1.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00
-1.00 -1.00  0.00  1.00 -1.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00
-1.00  0.00 -1.00 -1.00  1.00 -1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00 -1.00 -1.00  0.00 -1.00  1.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 1.00  1.00  0.00  0.00  0.00  1.00  1.00 -1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 1.00  0.00  1.00  0.00  0.00  0.00 -1.00  1.00 -1.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  1.00  1.00  1.00  0.00  0.00  0.00 -1.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  2.00 -1.00  1.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 -1.00  2.00 -1.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00 -1.00  2.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 -1.00  1.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 -1.00  0.00 -1.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00 -1.00  0.00
```

8.8 Übung 8: Eigenwertberechnung nach Jakobi

In dieser Übung soll ein Programm geschrieben werden, welches das Eigenwertproblem für eine symmetrische Matrix A mittels der Jakobi-Methode berechnet. Die Vorgehensweise dazu ist aus dem Theorieteil zu ersehen, außerdem treten kaum programmiertechnische Finessen auf. So ist der Code weitgehend selbsterklärend, erhellend wirken zudem noch die Kommentare. Die Diagonalisierung erfolgt in einem Unterprogramm mit Namen `EWP`, welches die Eigenvektoren zurückgibt und A direkt diagonalisiert. Nach erfolgter Diagonalisierung wird überprüft ob die gefundenen Lösungen die Eigenwertgleichung erfüllen.

8.8.1 Sourcecode-Uebung8.f90

```
MODULE LinAg
        IMPLICIT NONE

        CONTAINS

FUNCTION EWP (A, N)
        IMPLICIT NONE
        REAL(KIND=8), INTENT(INOUT) :: A(:,:)          !Matrix A
        INTEGER, INTENT(IN) :: N                       !Dimension
        REAL(KIND=8) :: s2,s                           !Parameter s^2
        REAL(KIND=8) :: c2,c                           !Parameter c^2
        REAL(KIND=8) :: sc, t                          !Parameter sc
        REAL(KIND=8) :: beta                           !Parameter beta
        REAL(KIND=8) :: V(N,N)                         !Temporaere Matrix
        REAL(KIND=8) :: x(N),y(N)                      !Hilfsvektoren
        REAL(KIND=8) :: EWP(N,N)                       !Loesungsmatrix
        REAL(KIND=8) :: epsilon                        !Abbruchbedingung
        INTEGER :: i,j                                 !Zaehlparameter
        INTEGER :: p,q                                 !Matrixindices
        REAL(KIND=8) :: element                        !Hilfselement

        epsilon = 1e-6                                 !Abbruchbedingung

        V = 0.d0                                       !Matrix V loeschen
        DO i=1, N
         V(i,i) = 1.d0                                 !V initialisieren
        END DO

        DO WHILE (abbruch(A,N) .GE.  epsilon)          !Beginn der Diagonalisierung
          DO p=1, N-1
           DO q=p+1, N
            IF( ABS(A(p,q)) .NE. 0.d0) THEN            !Falls apq=0 ist keine Aktion noetig
             beta = (A(q,q)-A(p,p))/(2*A(p,q))         !Berechne Parameter beta,s2,s1,sc
             s2 = 0.5 - 0.5*beta*(1/(SQRT(1+beta*beta)))
             c2 = 0.5 + 0.5*beta*(1/(SQRT(1+beta*beta)))
             sc = 0.5*(1/(SQRT(1+beta*beta)))
             s = SQRT(s2)
             c = SQRT(c2)

             DO i=1, N                                 !A berechnen
              IF((i .NE. p) .AND. (i .NE. q))  THEN
               element = A(i,p)
               A(i,p) = A(i,p)*c-A(i,q)*s
               A(i,q) = element*s+A(i,q)*c
               A(p,i) = A(i,p)
               A(q,i) = A(i,q)
              END IF
             END DO
             element = A(p,p)
             A(p,p) = A(p,p)*c2+A(q,q)*s2-2*A(p,q)*sc
             A(q,q) = element*s2+A(q,q)*c2+2*A(p,q)*sc
             A(p,q) = 0
             A(q,p) = 0
```

```
            DO i=1, N                                   !V berechnen
              element = V(i,p)
              V(i,p) = V(i,p)*c-V(i,q)*s
              V(i,q) = element*s+V(i,q)*c
            END DO


          END IF
         END DO
        END DO
       END DO
       EWP = V                                          !Eigenvektoren zurueckgeben

END FUNCTION EWP

FUNCTION abbruch(A, N)
       IMPLICIT NONE
       REAL(KIND=8), INTENT(INOUT) :: A(:,:)            !Matrix A
       INTEGER, INTENT(IN) :: N                         !Dimensionalitaet
       INTEGER :: i,j                                   !Laufparameter
       REAL(KIND=8) :: sum1, sum2                       !Summen
       REAL(KIND=8) :: abbruch                          !Abbruchkriterium

       sum1 = 0                                         !Null setzen der Summen
       sum2 = 0

       DO i=1, N
        DO j=1, N
          sum1 = sum1 + A(i,j)**2
          IF ( i .NE. j) THEN
           sum2 = sum2 + A(i,j)**2
          END IF
        END DO
       END DO

       abbruch = sum2/sum1                              !Abbruch, wenn dieser Bruch
                                                        !kleiner Epsilon

END FUNCTION abbruch


SUBROUTINE ausgabe (a)                                  !Ausgabe fuer Matrizen
       IMPLICIT NONE
       REAL(KIND=8), INTENT(IN) :: a(:, :)

       INTEGER :: i, j
       DO i=1, SIZE(a, DIM=1)
        DO j=1, SIZE(a, DIM=2)
          WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
        END DO
        WRITE(*,*)
       END DO
       PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
       USE LinAg
       IMPLICIT none
       INTEGER, PARAMETER :: N = 4                      !Dimension des Problemes
       REAL(KIND=8) :: oldA(N,N), A(N,N), M(N,N)        !Matrix A, Loesungsmatrix M
       INTEGER :: i,j                                   !Zaehlparameter

       A(1,1)=8
```

```
        A(2,2)=6
        A(3,3)=9
        A(4,4)=7
        A(1,2)=-1
        A(1,3)=3
        A(1,4)=-1
        A(2,3)=2
        A(2,4)=0
        A(3,4)=1

        DO i=2, N                                    !Matrix symmetrisieren
         DO j=1, i-1
           A(i,j)=A(j,i)
         END DO
        END DO

        oldA = A                                     !speichern der Ausgangsmatrix

        PRINT *
        PRINT *,"A="
        PRINT *
        CALL ausgabe(A)

        PRINT *, "Dieses Programm loest das Eigenwertproblem fuer A durch "
        PRINT *, "die zyklische Jakobi Methode. A ist dann diagonal, die Eigenwerte"
        PRINT *, "stehen auf der Hauptdiagonale. V beinhaltet ferner die zugehoerigen"
        PRINT *, "Eigenvektoren, welche in eine Matrix zusammengefasst wurden."
        PRINT *

        M = EWP(A,N)                                 !Loesen des EWPs
        PRINT *,"A ="
        CALL ausgabe(A)                              !Loesung ausgeben
        PRINT *
        PRINT *,"V="
        CALL ausgabe(M)
        PRINT *
        PRINT *,"Probe: Berechne AV. Dies muss dasselbe liefern wie V.r, wobei r die"
        PRINT *,"Diagonalisierte Matrix A ist. Aufgrund der Rechenvorschrift fuer Matrix-"
        PRINT *,"multiplikation muss die Diagonalmatrix RECHTS stehen!"
        PRINT *,"A.V="
        CALL ausgabe(MATMUL(oldA,M))
        PRINT *
        PRINT *,"V.r="
        CALL ausgabe(MATMUL(M,A))                    !ACHTUNG: Wegen der Matrix-
        PRINT *                                      !multiplikation muss die die
        PRINT *,"Fertig!"                            !Eigenwerte beinhaltende Dia-
        PRINT *                                      !gonalmatrix RECHTS stehen!!!!
END PROGRAM main
```

8.8.2 Output des Programmes

```
 A=

 8.00000-1.00000 3.00000-1.00000
-1.00000 6.00000 2.00000 0.00000
 3.00000 2.00000 9.00000 1.00000
-1.00000 0.00000 1.00000 7.00000

 Dieses Programm loest das Eigenwertproblem fuer A durch
 die zyklische Jakobi Methode. A ist dann diagonal, die Eigenwerte
 stehen auf der Hauptdiagonale. V beinhaltet ferner die zugehoerigen
 Eigenvektoren, welche in eine Matrix zusammengefasst wurden.

 A =
11.70430 0.00000 0.00000 0.00000
 0.00000 6.59234 0.00000 0.00000
 0.00000 0.00000 8.40766 0.00000
 0.00000 0.00000 0.00000 3.29570


 V=
 0.58230 0.23010-0.57304-0.52878
 0.17578-0.62898 0.47230-0.59197
 0.79249-0.07123 0.28205 0.53604
 0.04468 0.73917 0.60746-0.28745


 Probe: Berechne AV. Dies muss dasselbe liefern wie V.r, wobei r die
 Diagonalisierte Matrix A ist. Aufgrund der Rechenvorschrift fuer Matrix-
 multiplikation muss die Diagonalmatrix RECHTS stehen!
 A.V=
 6.81539 1.51687-4.81795-1.74270
 2.05733-4.14642 3.97095-1.95094
 9.27551-0.46960 2.37138 1.76662
 0.52295 4.87285 5.10728-0.94736


 V.r=
 6.81539 1.51687-4.81795-1.74270
 2.05733-4.14642 3.97095-1.95094
 9.27551-0.46960 2.37138 1.76662
 0.52295 4.87285 5.10728-0.94736


 Fertig!
```

8.9 Übung 9: Hubbard Modell

In dieser Übung soll das Problem des Hubbard-Modelles mit dem Programm, welches in der vorherigen Übung erzeugt wurde, gelöst werden. Dazu sollen die Eigenwerte $E^{(n)}$ und die dazugehörigen Eigenvektoren $v^{(n)}$, $n = 1, \ldots, 15$ berechnet werden. Die Einträge $v_i^{(n)}$ geben die Gewichte an aus denen der physikalische Zustand $|n\rangle$ in der gewählten Basis $|\phi_i\rangle$ aufgebaut ist, dh.

$$|n\rangle = \sum_{i=1}^{1} 5 v_i^{(n)} |\phi_i\rangle. \tag{8.13}$$

Das Eigenwertproblem soll für unterschiedliche Parameter U im Intervall $U \in [-10, 10]$ in Schritten von $\Delta U = 0.1$ untersucht werden. Ausserdem sollen die Fälle $B = 3.0, 6.0, 12.0$ untersucht werden. Die Eigenwerte $E^{(n)}, n = 1, 2, \ldots, 15$ sind als Funktion von U aufzutragen, ausserdem sollen die Gewichte für den Grundzustand (Zustand mit niedrigster Energie) als Funktion von U studiert werden. Im Folgenden sehen wir die Eigenwerte, geplottet als Funktion des Parameters U.

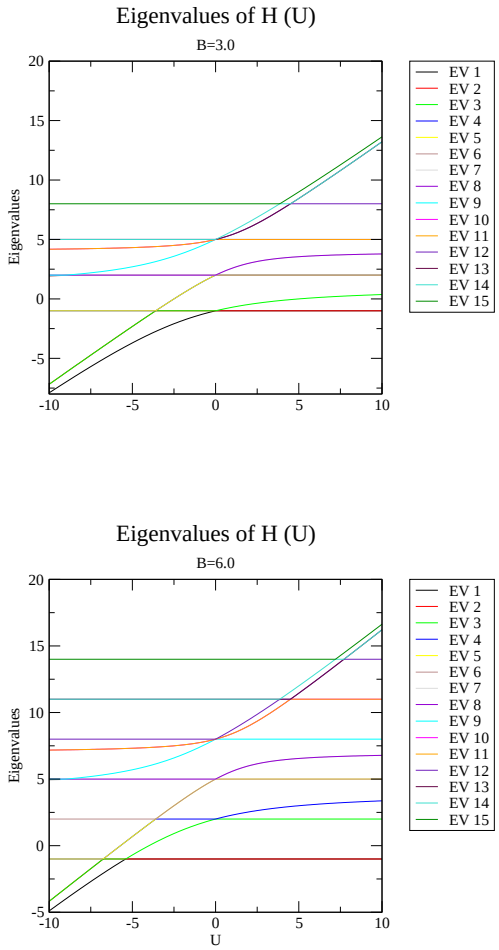


Abbildung 8: Eigenwerte als Funktion des Parameters U für $B = 3$ und $B = 6$

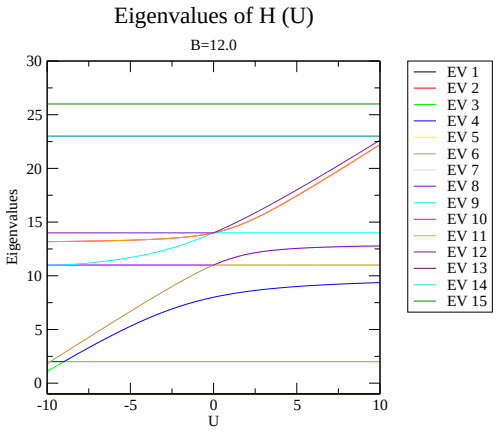


Abbildung 9: Eigenwerte als Funktion des Parameters U für $B = 12$

8.9.1 Sourcecode-Uebung9.f90

```
MODULE LinAg
IMPLICIT NONE

CONTAINS

FUNCTION EWP (A, N)
IMPLICIT NONE
REAL(KIND=8), INTENT(INOUT) :: A(:,:)          !Matrix A
INTEGER, INTENT(IN) :: N                       !Dimension
REAL(KIND=8) :: s2,s                           !Parameter s^2
REAL(KIND=8) :: c2,c                           !Parameter c^2
REAL(KIND=8) :: sc, t                          !Parameter sc
REAL(KIND=8) :: beta                           !Parameter beta
REAL(KIND=8) :: V(N,N)                         !Temporaere Matrix
REAL(KIND=8) :: x(N),y(N)                      !Hilfsvektoren
REAL(KIND=8) :: EWP(N,N)                       !Loesungsmatrix
REAL(KIND=8) :: epsilon                        !Abbruchbedingung
INTEGER :: i,j                                 !Zaehlparameter
INTEGER :: p,q                                 !Matrixindices
REAL(KIND=8) :: element                        !Hilfselement

epsilon = 1e-6                                 !Abbruchbedingung

V = 0.d0                                       !Matrix V loeschen
DO i=1, N
 V(i,i) = 1.d0                                 !V initialisieren
END DO

DO WHILE (abbruch(A,N) .GE.  epsilon)          !Beginn der Diagonalisierung
  DO p=1, N-1
   DO q=p+1, N
    IF( ABS(A(p,q)) .NE. 0.d0) THEN            !Falls apq=0 ist keine Aktion noetig
     beta = (A(q,q)-A(p,p))/(2*A(p,q))         !Berechne Parameter beta,s2,s1,sc
     s2 = 0.5 - 0.5*beta*(1/(SQRT(1+beta*beta)))
     c2 = 0.5 + 0.5*beta*(1/(SQRT(1+beta*beta)))
     sc = 0.5*(1/(SQRT(1+beta*beta)))
     s = SQRT(s2)
     c = SQRT(c2)

     DO i=1, N                                 !A berechnen
```

```
        IF((i .NE. p) .AND. (i .NE. q))  THEN
         element = A(i,p)
         A(i,p) = A(i,p)*c-A(i,q)*s
         A(i,q) = element*s+A(i,q)*c
         A(p,i) = A(i,p)
         A(q,i) = A(i,q)
        END IF
       END DO
       element = A(p,p)
       A(p,p) = A(p,p)*c2+A(q,q)*s2-2*A(p,q)*sc
       A(q,q) = element*s2+A(q,q)*c2+2*A(p,q)*sc
       A(p,q) = 0
       A(q,p) = 0

       DO i=1, N                                    !V berechnen
        element = V(i,p)
        V(i,p) = V(i,p)*c-V(i,q)*s
        V(i,q) = element*s+V(i,q)*c
       END DO


     END IF
    END DO
   END DO
END DO
EWP = V                                             !Eigenvektoren zurueckgeben

END FUNCTION EWP

FUNCTION abbruch(A, N)
IMPLICIT NONE
REAL(KIND=8), INTENT(INOUT) :: A(:,:)               !Matrix A
INTEGER, INTENT(IN) :: N                            !Dimensionalitaet
INTEGER :: i,j                                      !Laufparameter
REAL(KIND=8) :: sum1, sum2                          !Summen
REAL(KIND=8) :: abbruch                             !Abbruchkriterium

sum1 = 0                                            !Null setzen der Summen
sum2 = 0

DO i=1, N
 DO j=1, N
  sum1 = sum1 + A(i,j)**2
  IF ( i .NE. j) THEN
   sum2 = sum2 + A(i,j)**2
  END IF
 END DO
END DO

abbruch = sum2/sum1                                 !Abbruch, wenn dieser Bruch
                                                    !kleiner Epsilon

END FUNCTION abbruch


SUBROUTINE ausgabe (a)                                      !Ausgabe fuer Matrizen
IMPLICIT NONE
REAL(KIND=8), INTENT(IN) :: a(:, :)

INTEGER :: i, j
DO i=1, SIZE(a, DIM=1)
 DO j=1, SIZE(a, DIM=2)
  WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
 END DO
 WRITE(*,*)
END DO
PRINT * !Leerzeile
```

```
END SUBROUTINE ausgabe


END MODULE LinAg

PROGRAM main
USE LinAg
IMPLICIT none
INTEGER, PARAMETER :: N = 15                    !Dimension des Problemes
REAL(KIND=8) :: U, B, t                         !Parameter des Systems
REAL(KIND=8) ::  H(N,N), oldH(N,N), L(N,N)      !Matrix A, Loesungsmatrix L
INTEGER :: i,j, index, set                      !Zaehlparameter
REAL(KIND=8) :: aux, min                        !Hilfsvariable


U = -10.0                                       !U vorsetzen
B = 3.0                                         !B vorseetzen
set = 0                                         !Normierung noch nicht vorgenommen

PRINT *
PRINT *
PRINT *, "Dieses Programm loest das Eigenwertproblem fuer H  durch "
PRINT *, "die zyklische Jakobi Methode. H ist die Matrixdarstellung des"
PRINT *, "Hamiltonoperators des Hubbardmodelles aus Uebung 7."
PRINT *, "Das Problem wird fuer verschiedene Parameter U und B geloest,"
PRINT *, "t wurde stets auf 1 gesetzt. Ferner werden die Loesungen in 3 "
PRINT *, "Dateien geschrieben, die jeweils die Staerke des aeusseren Feldes"
PRINT *, "als Namen tragen, also etwa B3.dat."
PRINT *


OPEN (UNIT=20, FILE="B3.dat", ACTION="write")   !Ausgabe vorbereiten
OPEN (UNIT=30, FILE="B6.dat", ACTION="write")
OPEN (UNIT=40, FILE="B12.dat", ACTION="write")
OPEN (UNIT=200, FILE="B3v.dat", ACTION="write") !Ausgabe vorbereiten
OPEN (UNIT=300, FILE="B6v.dat", ACTION="write")
OPEN (UNIT=400, FILE="B12v.dat", ACTION="write")

200 FORMAT(f8.4, 15(2x,f8.4))                   !Formatter anweisen



DO WHILE (U .LE. 10.01)                         !Rundungsfehler beruecksichtigt
 DO  WHILE (B .LE. 12.0)                        !B...aeusseres Feld

 t = 1.0                                        !t bleibt stets auf 1 gesetzt

 H = 0.d0                                       !H zuruecksetzen
 H(1,1) = U + B                                 !H befuellen
 H(2,2) = U + B
 H(3,3) = U + B
 DO i=4, 9
  H(i,i) = B
 END DO
 H(10,10) = 2*B
 H(11,11) = 2*B
 H(12,12) = 2*B
 H(1,4) = -t
 H(1,5) = -t
 H(1,7) = t
 H(1,8) = t
 H(2,4) = -t
 H(2,6) = -t
 H(2,7) = t
 H(2,9) = t
 H(3,5) = -t
```

```
 H(3,6) = -t
 H(3,8) = t
 H(3,9) = t
 H(4,5) = -t
 H(4,9) = t
 H(5,6) = -t
 H(6,7) = t
 H(7,8) = -t
 H(8,9) = -t
 H(10,11) = -t
 H(10,12) = t
 H(11,12) = -t
 H(13,14) = -t
 H(13,15) = t
 H(14,15) = -t


 DO i=2, N                                    !Matrix symmetrisieren
  DO j=1, i-1
   H(i,j) = H(j,i)
  END DO
 END DO

oldH = H

PRINT *, "H="
CALL ausgabe(H)
L = EWP(H,N)
PRINT *, " H="
CALL ausgabe(H)
PRINT *
PRINT *, " Eigenvektoren L="
CALL ausgabe(L)
PRINT *
PRINT *,"H.L="
CALL ausgabe(MATMUL(oldH,L))
PRINT *
PRINT *,"L.r="
CALL ausgabe(MATMUL(L,H))
PRINT *
 DO i=1, N                                    !Sortieren der EW. Sonst Oszillationen
  min = H(i,i)                                !da Phasen frei waehlbar, also auch -1
  DO j=i+1, N
   IF(min .GT. H(j,j)) THEN
    min = H(j,j)
    aux = H(i,i)
    H(i,i) = min
    H(j,j) = aux
   END IF
  END DO
 END DO


 IF( B .EQ. 3.0) THEN                         !Daten entsprechend schreiben
  WRITE(20,200, ADVANCE="NO") U               !Eigenwerte schreiben
  WRITE(200,200, ADVANCE="NO") U              !Eigenvektoren schreiben
  DO i=1, N
   WRITE(20,200, ADVANCE="NO") H(i,i)
  END DO
!  IF( set .EQ. 0) THEN
!  DO i=1, N                                   !Sortieren der Eigenvektoren
!   IF( L(1,i) .NE. 0.) THEN
!    IF( L(1,i) .LT. 0.) THEN
!     index = i
!     set = 1
!    END IF
!   END IF
```

```
!   END DO
! END IF
! IF( L(1,index) .LT. 0.) THEN
!   L(1,:) = -L(1,:)
! END IF
   DO i=1, N
    WRITE(200,200, ADVANCE="NO") L(1,i)
   END DO
   WRITE(20,200)
   WRITE(200,200)
  ELSE IF( B .EQ. 6.0) THEN
   WRITE(30,200, ADVANCE="NO") U
   DO i=1, N
    WRITE(30,200, ADVANCE="NO") H(i,i)
   END DO
   WRITE(30,200)
  ELSE
   WRITE(40,200, ADVANCE="NO") U
   DO i=1, N
    WRITE(40,200, ADVANCE="NO") H(i,i)
   END DO
   WRITE(40,200)
  END IF

  B = 2*B                                          !B aktualisieren
  END DO
B = 3.0                                            !B ruecksetzen
U = U + 0.1                                        !U aktualisieren
 set = 0
END DO

!PRINT *, "Eigenwerte: H="
!CALL ausgabe(H)
!PRINT *, "Eigenvektoren: V="
!CALL ausgabe(L)
!CALL ausgabe(MATMUL(H,L))

CLOSE(UNIT=20)                                     !Dateien schliessen
CLOSE(UNIT=30)
CLOSE(UNIT=40)

PRINT *, "Fertig!"
PRINT *

END PROGRAM main
```

8.9.2 Output des Programmes

```
 Dieses Programm loest das Eigenwertproblem fuer H  durch
 die zyklische Jakobi Methode. H ist die Matrixdarstellung des
 Hamiltonoperators des Hubbardmodelles aus Uebung 7.
 Das Problem wird fuer verschiedene Parameter U und B geloest,
 t wurde stets auf 1 gesetzt. Ferner werden die Loesungen in 3
 Dateien geschrieben, die jeweils die Staerke des aeusseren Feldes
 als Namen tragen, also etwa B3.dat.

 Eigenwerte: H=
-1.00000 0.00000 0.00061-0.00000 0.00000 0.00081-0.00000 0.00164-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00000-1.00000 0.00000-0.00000 0.00000 0.00000-0.00000 0.00000-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00061 0.00000 2.00000-0.00000-0.00000 0.00006-0.00000 0.00012-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.00000-0.00000-0.00000 9.36675 0.00000 0.00000-0.00000 0.00000-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00000 0.00000-0.00000 0.0000011.00000 0.00000-0.00000 0.00000-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00081 0.00000 0.00006 0.00000 0.0000011.00000-0.00000 0.00000-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.00000-0.00000-0.00000-0.00000-0.00000-0.0000012.78301-0.00000-0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00164 0.00000 0.00012 0.00000 0.00000 0.00000-0.0000012.78301 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.00000-0.00000-0.00000-0.00000-0.00000-0.00000-0.00000 0.0000014.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000022.21699 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000022.21699 0.00000 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000022.63325 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000023.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000023.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000026.00000

 Eigenvektoren: V=
-0.12621-0.24195 0.56352-0.00000-0.00000-0.76984-0.00000-0.10885-0.05892 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.12605 0.78775 0.56338 0.00000-0.00000 0.17552-0.00000 0.10555-0.06486 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.12607-0.54580 0.56332 0.00000-0.00000 0.59459-0.00000 0.00350 0.12378 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.39839-0.05922-0.08918-0.55335 0.40825 0.06451 0.16473-0.01561-0.57045 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.39862 0.08547-0.08917-0.13402-0.40825 0.01905 0.56158-0.48593 0.29891 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
-0.39815-0.02625-0.08917 0.41933 0.40825-0.08352 0.39685 0.50212 0.27154 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.39839 0.05922 0.08918-0.55335 0.40825-0.06451 0.16473 0.01562 0.57045 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.39862-0.08547 0.08917-0.13402-0.40825-0.01905 0.56158 0.48593-0.29891 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.39815 0.02625 0.08917 0.41933 0.40825 0.08352 0.39685-0.50212-0.27154 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.40825 0.70711 0.57735 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000-0.40825 0.70711-0.57735 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000-0.81650 0.00000 0.57735 0.00000 0.00000 0.00000
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.40825 0.70711 0.57735
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000-0.40825 0.70711-0.57735
 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000-0.81650 0.00000 0.57735

 Fertig!
```

8.9.3 Sourcecode-Uebung9nr.f90

Nachdem sich für das Plotten der Einträge des Eigenvektors zum niedrigsten Eigenwert Probleme ergeben haben, dh. der Graph starke Oszillationen aufwies, wurde hier, um die Umnormierung (gegebenenfalles Multiplikation mit -1) der nur bis auf eine Phase von π bestimmten Einträge nicht vornehmen zu müssen, die Implementierung der zyklischen Jacobi-Methode nach Anleitung aus Numerical Recipes in Fortran vorgenommen. Im Folgenden ist der Sourcecode abgedruckt.

```
MODULE nr
IMPLICIT NONE

CONTAINS

SUBROUTINE jacobi(a, d, v, nrot)
IMPLICIT NONE
INTEGER, INTENT(OUT) :: nrot
REAL(KIND=8), DIMENSION(:), INTENT(OUT):: d
REAL(KIND=8), DIMENSION(:,:), INTENT(INOUT) :: a
REAL(KIND=8), DIMENSION(:,:), INTENT(out) :: v

!Berechnet alle Eigenwerte und Eigenvektoren einer reellen, symm.
!NxN Matrix a. Bei der Ausgabe werden die Elemente von a ueber der
!Diagonale vernichtet. d ist ein Vektor der Laenge N der die
!Eigenwerte von a zurueckgibt. v ist eine NxN Matrix, deren Spalten
!bei der Ausgabe die normierten Eigenvektoren beinhalten. Der
!Parameter nrot gibt die Anzahl der benoetigten Jacobi-Rotationen
!zurueck. Dieser Algorithmus wurde direkt aus NUMERICAL RECIPES IN
!FORTRAN 90, ISBN 0-521-57439-0 entnommen.

INTEGER :: i, ip, iq, n
INTEGER :: j, k, l
REAL(KIND=8) :: c,g,h,s,sm,t,tau,theta,tresh
REAL(KIND=8), DIMENSION(size(d)) :: b,z

!Initialisiere v auf einheitsmatrix
v(:,:) = 0.0
DO i=1, SIZE(d)
 v(i,i) = 1.
END DO

!Speichere die Diagonale von a in vektor b
DO i=1, SIZE(d)
 b(i) = a(i,i)
END DO

n = SIZE(d)

d(:)=b(:)
z(:)=0.0
nrot = 0

DO i=1, 50
!Summiere Offdiagonalelemente
 sm = 0.
 DO j=1, SIZE(d)-1
  DO k=j+1, SIZE(d)
   sm = sm + ABS(a(j,k))
  END DO
 END DO
 IF(sm == 0.0) RETURN
 IF( i < 4) THEN !Vgl Funktion merge in nr
  tresh = 0.2*sm/n**2
 ELSE
  tresh = 0.0
 END IF
 DO ip=1, n-1
  DO iq=ip+1, n
   g=100.0 * ABS(a(ip,iq))
```

```
    IF ((i>4) .AND. (ABS(d(ip))+g==ABS(d(ip))) .AND. (ABS(d(iq))+g==ABS(d(iq)))) THEN
     a(ip,iq)=0.0
    ELSE IF (ABS(a(ip,iq))>tresh) THEN
     h=d(iq)-d(ip)
     IF(ABS(h)+g==ABS(h)) THEN
      t=a(ip,iq)/h
     ELSE
      theta=0.5*h/a(ip,iq)
      t=1.0/(ABS(theta)+SQRT(1.0+theta**2))
      IF (theta < 0.0) t=-t
     END IF
     c=1.0/SQRT(1+t**2)
     s=t*c
     tau=s/(1.0+c)
     h=t*a(ip,iq)
     z(ip)=z(ip)-h
     z(iq)=z(iq)+h
     d(ip)=d(ip)-h
     d(iq)=d(iq)+h
     a(ip,iq)=0.0
     CALL jrotate(a(1:ip-1,ip),a(1:ip-1,iq))
     CALL jrotate(a(ip,ip+1:iq-1),a(ip+1:iq-1,iq))
     CALL jrotate(a(ip,iq+1:n),a(iq,iq+1:n))
     CALL jrotate(v(:,ip),v(:,iq))
     nrot = nrot +1
    END IF
   END DO
  END DO
  b(:)=b(:)+z(:)
  d(:)=b(:)
  z(:)=0.0
END DO

CONTAINS

SUBROUTINE jrotate(a1,a2)
REAL(KIND=8), DIMENSION(:), INTENT(INOUT) :: a1,a2
REAL(KIND=8), DIMENSION(SIZE(a1)) :: wk1
wk1(:)=a1(:)
a1(:)=a1(:)-s*(a2(:)+a1(:)*tau)
a2(:)=a2(:)+s*(wk1(:)-a2(:)*tau)
END SUBROUTINE jrotate
END SUBROUTINE jacobi



SUBROUTINE ausgabe (a)                                    !Ausgabe fuer Matrizen
IMPLICIT NONE
REAL(KIND=8), INTENT(IN) :: a(:, :)

INTEGER :: i, j
DO i=1, SIZE(a, DIM=1)
 DO j=1, SIZE(a, DIM=2)
  WRITE ( *,'(F8.5)', ADVANCE="NO") (a(i,j))
 END DO
 WRITE(*,*)
END DO
PRINT * !Leerzeile
END SUBROUTINE ausgabe


END MODULE nr

PROGRAM main
USE nr
IMPLICIT none
INTEGER, PARAMETER :: N = 15                          !Dimension des Problemes
```

```
REAL(KIND=8) :: U, B, t                          !Parameter des Systems
REAL(KIND=8) ::  H(N,N), V(N,N)                  !Matrix A, Loesungsmatrix L
INTEGER :: i,j,k,l,pos,sig                       !Zaehlparameter, Position
REAL(KIND=8) :: d(N)                             !Vektor fuer Eigenwerte
REAL(KIND=8) :: min                              !Minimalwert
INTEGER :: nrot = 100000                         !Anzahl Rotationen
INTEGER :: flag = 1                              !flag fuer Normierung


!CALL ausgabe(A)
!CALL jacobi(A,d,v,nrot)
!PRINT *, "v="
!CALL ausgabe(v)
!PRINT *,"d="
!PRINT *, d(:)

U = -10.0                                        !U vorsetzen
B = 3.0                                          !B vorseetzen


PRINT *
PRINT *, "Variante NUMERICAL RECIPES"
PRINT *, "Dieser Algorithmus wurde direkt aus"
PRINT *, "NUMERICAL RECIPES IN FORTRAN 90, ISBN 0-521-57439-0 entnommen."
PRINT *, "Dieses Programm loest das Eigenwertproblem fuer H  durch "
PRINT *, "die zyklische Jakobi Methode. H ist die Matrixdarstellung des"
PRINT *, "Hamiltonoperators des Hubbardmodelles aus Uebung 7."
PRINT *, "Das Problem wird fuer verschiedene Parameter U und B geloest,"
PRINT *, "t wurde stets auf 1 gesetzt. Ferner werden die Loesungen in 3 "
PRINT *, "Dateien geschrieben, die jeweils die Staerke des aeusseren Feldes"
PRINT *, "als Namen tragen, also etwa B3.dat."
PRINT *


OPEN (UNIT=20, FILE="B3.dat", ACTION="write")    !Ausgabe vorbereiten
OPEN (UNIT=30, FILE="B6.dat", ACTION="write")
OPEN (UNIT=40, FILE="B12.dat", ACTION="write")
OPEN (UNIT=200, FILE="B3v.dat", ACTION="write") !Ausgabe vorbereiten
OPEN (UNIT=300, FILE="B6v.dat", ACTION="write")
OPEN (UNIT=400, FILE="B12v.dat", ACTION="write")

200 FORMAT(f8.4, 15(2x,f8.4))                    !Formater anweisen


DO WHILE(U .LE. 10.0)
 DO WHILE(B .LE. 12.0)

  t = 1.0                                        !t bleibt stets auf 1 gesetzt

  H = 0.d0                                       !H zuruecksetzen
  H(1,1) = U + B                                        !H befuellen
  H(2,2) = U + B
  H(3,3) = U + B
  DO i=4, 9
   H(i,i) = B
  END DO
  H(10,10) = 2*B
  H(11,11) = 2*B
  H(12,12) = 2*B
  H(1,4) = -t
  H(1,5) = -t
  H(1,7) = t
  H(1,8) = t
  H(2,4) = -t
  H(2,6) = -t
  H(2,7) = t
  H(2,9) = t
```

```
 H(3,5) = -t
 H(3,6) = -t
 H(3,8) = t
 H(3,9) = t
 H(4,5) = -t
 H(4,9) = t
 H(5,6) = -t
 H(6,7) = t
 H(7,8) = -t
 H(8,9) = -t
 H(10,11) = -t
 H(10,12) = t
 H(11,12) = -t
 H(13,14) = -t
 H(13,15) = t
 H(14,15) = -t


 DO i=2, N                      !Matrix symmetrisieren
  DO j=1, i-1
   H(i,j) = H(j,i)
  END DO
 END DO                         !Matrix befuellen abgeschlossen


 CALL jacobi(H,d,V,nrot)        !Diagonalisiere

 min = d(1)                     !Setze vorl. Minwert
 pos = 1
 DO l=2, N                      !Finde Position des kleinsten Eigenwertes
  IF(d(l) < min) THEN
   min = d(l)                   !Aktualisiere min
   pos = l                      !Lerne neue Position
  END IF
 END DO

 IF( B .EQ. 3.0) THEN
  WRITE(20,200, ADVANCE="NO") U
  WRITE(200,200, ADVANCE="NO") U
  DO i=1, N
   WRITE(20,200, ADVANCE="NO") d(i)
   WRITE(200,200, ADVANCE="NO") V(i,pos)
  END DO
  WRITE(20,200)                 !CR, LF in 20
  WRITE(200,200)                !CR, LF in 200
 ELSE IF( B .EQ. 6.0) THEN
  WRITE(30,200, ADVANCE="NO") U
  WRITE(300,200, ADVANCE="NO") U
  DO i=1, N
   WRITE(30,200, ADVANCE="NO") d(i)
   WRITE(300,200, ADVANCE="NO") V(i,pos)
  END DO
  WRITE(30,200)                 !CR, LF in 30
  WRITE(300,200)                !CR, LF in 300
 ELSE
  WRITE(40,200, ADVANCE="NO") U
  WRITE(400,200, ADVANCE="NO") U
  DO i=1, N
   WRITE(40,200, ADVANCE="NO") d(i)
   WRITE(400,200, ADVANCE="NO") V(i,pos)
  END DO
  WRITE(40,200)                 !CR, LF in 40
  WRITE(400,200)                !CR, LF in 400
 END IF

 B = 2*B                        !B aktualisieren
END DO
```

```
 B = 3.0                          !B ruecksetzen
 U = U + 0.1                      !U aktualisieren
END DO
PRINT *, "U="
PRINT *, U
PRINT *, "B="
PRINT *, B
PRINT *, "d="
PRINT *, d(:)
PRINT *, "V="
CALL ausgabe(V)

CLOSE(UNIT=20)                                    !Dateien schliessen
CLOSE(UNIT=30)
CLOSE(UNIT=40)
CLOSE(UNIT=200)
CLOSE(UNIT=300)
CLOSE(UNIT=400)

PRINT *, "Fertig!"
PRINT *

END PROGRAM main
```

Es bleibt noch, die entsprechenden Eigenvektoreinträge als Funktion von U zu Plotten.

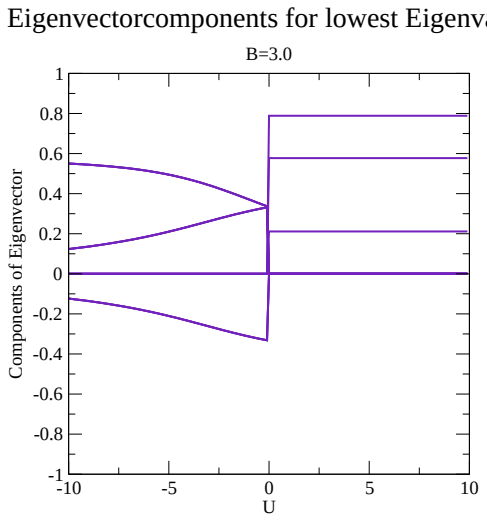


Abbildung 10: Einträge des Eigenvektors zum kleinsten Eigenwert als Funktion des Parameters U für $B = 3$

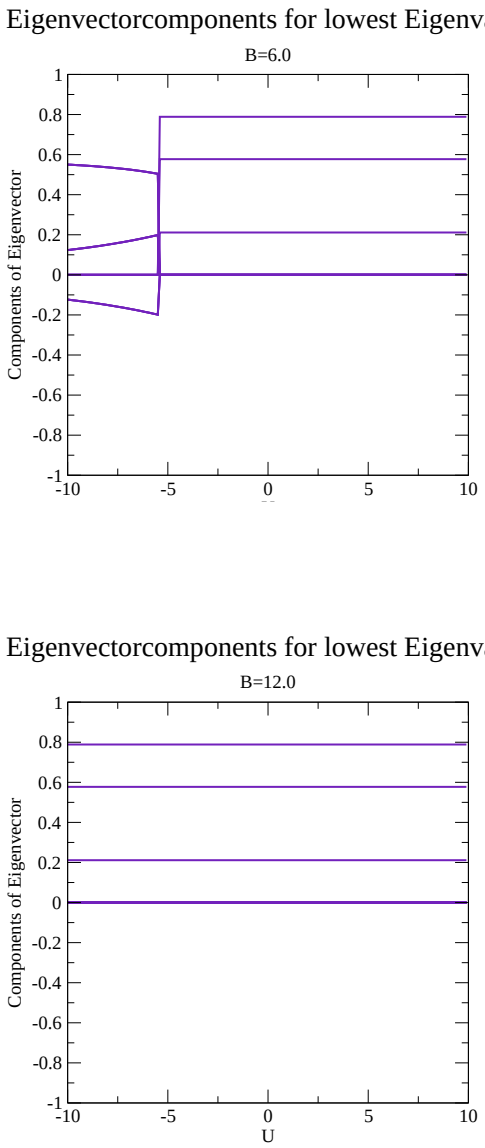


Abbildung 11: Einträge der Eigenvektoren zum kleinsten Eigenwert als Funktion des Parameters U für $B = 6$ und $B = 12$

8.10 Übung 10: Fourier-Methode

Zum Üben der Fouriermethode soll folgendes Problem gelöst werden:

$$D\phi(\vec{x}) = \rho(\vec{x}). \tag{8.14}$$

Dabei ist D der Differentialoperator

$$D = -\Delta + \mu^2 = -\sum_{i=1}^{3} \frac{\partial^2}{\partial x_i^2} + \mu^2. \tag{8.15}$$

Wenn man nun den Differentialoperator diskretisiert, dann lässt sich das Problem in folgender Form schreiben:

$$\sum_{\vec{m}} D_{\vec{n},\vec{m}} \phi_{\vec{m}} = \rho_{\vec{n}}, \tag{8.16}$$

was offensichtlich durch $\phi = D^{-1}\rho$ gelöst wird. Die Aufgabe besteht nun darin, die inverse Matrix D^{-1} mit Hilfe der diskreten Fourier-Transformation

$$U_{\vec{k},\vec{n}} = \frac{1}{N^{3/2}} e^{-i\frac{2\pi}{N}\vec{n}\cdot\vec{k}} = \prod_{i=1}^{3} \frac{1}{\sqrt{N}} e^{-i\frac{2\pi}{N}n_i k_i} \tag{8.17}$$

zu bestimmen.

8.10.1 Lösung

Zunächst wollen wir den Differentialoperator D diskretisieren:

$$D_{\vec{n},\vec{m}} = -\sum_{i=1}^{3} \frac{\delta_{\vec{n}+\hat{i},\vec{m}} - 2\delta_{\vec{n},\vec{m}} + \delta_{\vec{n}-\hat{i},\vec{m}}}{a^2} + \mu^2 \delta_{\vec{n},\vec{m}}, \tag{8.18}$$

wobei

$$\delta_{\vec{n},\vec{m}} = \delta_{n_1 m_1}\delta_{n_2 m_2}\delta_{n_3 m_3} \tag{8.19}$$

ist. Als nächtes wollen wir die Fouriertransformierte Matrix $\tilde{D}_{\vec{n},\vec{m}}$ bestimmen.

$$\begin{aligned}
\tilde{D}_{\vec{n},\vec{m}} &= \sum_{\vec{n},\vec{m}} = U_{\vec{k},\vec{n}} D_{\vec{n},\vec{m}} U^*_{\vec{l},\vec{m}} \qquad (8.20)\\
&= \frac{1}{N^3}\sum_{\vec{n},\vec{m}} e^{-i\frac{2\pi}{N}\vec{n}\cdot\vec{k}}\left(\frac{-1}{a^2}\sum_{i=1}^{3}(\delta_{\vec{n}+\hat{i},\vec{m}} - 2\delta_{\vec{n},\vec{m}} + \delta_{\vec{n}-\hat{i},\vec{m}}) + \mu^2\delta_{\vec{n},\vec{m}}\right)e^{i\frac{2\pi}{N}\vec{l}\cdot\vec{m}}\\
&= \frac{1}{N^3}\sum_{\vec{n}} e^{-i\frac{2\pi}{N}\vec{n}\cdot\vec{k}}\frac{-1}{a^2}\sum_{i=1}^{3}(e^{i\frac{2\pi}{N}\vec{l}\cdot(\vec{n}+\hat{i})} - 2e^{i\frac{2\pi}{N}\vec{l}\cdot\vec{n}} + e^{i\frac{2\pi}{N}\vec{l}(\vec{n}-\hat{i})}) + \delta_{\vec{k},\vec{l}}\mu^2\\
&= \underbrace{\frac{1}{N^3}\sum_{\vec{n}} e^{-i\frac{2\pi}{N}\vec{n}\cdot\vec{k}}e^{i\frac{2\pi}{N}\vec{n}\cdot\vec{l}}}_{\delta_{\vec{k},\vec{l}}}\left(\frac{-1}{a^2}\sum_{i=1}^{3}(e^{i\frac{2\pi}{N}\vec{l}\cdot\hat{i}} + e^{-i\frac{2\pi}{N}\vec{l}\cdot\hat{i}} - 2)\right) + \delta_{\vec{k},\vec{l}}\mu^2\\
&= \delta_{\vec{k},\vec{l}}\left(\frac{2}{a^2}\sum_{i=1}^{3}(1-\cos(\frac{2\pi}{N}l_i)) + \mu^2\right).
\end{aligned}$$

Wir haben also eine Diagonalmatrix durch die Fouriertransformation erhalten, dh. $\tilde{D} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N) = M$.

$$\lambda_{\vec{l}} = \frac{2}{a^2}\sum_{i=1}^{3}(1-\cos(\frac{2\pi}{N}l_i)) + \mu^2. \tag{8.21}$$

Damit können wir nun die gesuchte inverse Matrix durch Rücktransformation der invertierten Diagonalmatrix gewinnen. Wir haben:

$$\begin{aligned}
D^{-1}_{\vec{n},\vec{m}} &= (U^\dagger \tilde{D}^{-1} U)_{\vec{n},\vec{m}} = (U^\dagger M^{-1} U)_{\vec{n},\vec{m}} \qquad (8.22)\\
&= \sum_{\vec{k},\vec{l}} U^\dagger_{\vec{n},\vec{k}} M^{-1}_{\vec{k},\vec{l}} U_{\vec{l},\vec{m}} = \frac{1}{N^3}\sum_{\vec{k},\vec{l}} e^{i\frac{2\pi}{N}\vec{n}\cdot\vec{k}}\frac{\delta_{\vec{k},\vec{l}}}{\lambda_{\vec{k}}}e^{-i\frac{2\pi}{N}\vec{l}\cdot\vec{m}}\\
&= \frac{1}{N^3}\sum_{\vec{k}}\frac{1}{\lambda_{\vec{k}}}e^{i\frac{2\pi}{N}\vec{k}(\vec{n}-\vec{m})}.
\end{aligned}$$

8.11 Übung 11: Diffusionsgleichung

In dieser Übung wollen wir die Diffusionsgleichung mit konstanter Diffusionskonstante D numerisch für 1+1 Dimension lösen. Die Diskretisierung der Gleichung mit dem Crank-Nicholson Schema ist:

$$u_n^{(\tau)} = \sum_{m=1}^{N} ((\mathbb{1} - H)^{-1}(\mathbb{1} + H))_{nm} u_m^{(\tau-1)}. \tag{8.23}$$

Die Ortskoordinate $x = n\Delta x$ mit $n = 1, 2, \ldots, N$ und die Zeit ist $t = \tau\Delta t$ mit $\tau = 0, 1, 2, \ldots$. Die Matrix H ist gegeben durch

$$H_{nm} = \alpha(\delta_{n+1,m} - 2\delta_{n,m} + \delta_{n-1,m}), \tag{8.24}$$

mit $\alpha = D\Delta t/(2\Delta x^2)$. Die Randbedingungen werden als periodisch angenommen. Es soll nun die Matrix $(\mathbb{1} - H)^{-1}(\mathbb{1} + H)$ unter Verwendung der Methoden aus dem vorigen Beispiel berechnet werden. Ausserdem soll ein Ausdruck gefunden werden, der die Verteilung $u_n^{(\tau)}$ im Zeitschritt τ direkt aus der gegebenen Anfangskonfiguration $u_n^{(0)}$ berechnet.

8.11.1 Lösung

Wir betrachten zunächts:

$$M_{nm} = (\mathbb{1} - H)_{nm} = (\delta_{nm} - \alpha(\delta_{n+1,m} - 2\delta_{nm} + \delta_{n-1,m})). \tag{8.25}$$

Wenn wir nun Fouriertransformieren, dann erhalten wir für $\tilde{M}$:

$$\begin{aligned}
\tilde{M}_{kl} &= \sum_{n,m} U_{kn} M_{nm} U^*_{lm} \qquad (8.26)\\
&= \frac{1}{N}\sum_{n,m} e^{-i\frac{2\pi}{N}kn}(\delta_{nm} - \alpha(\delta_{n+1,m} - 2\delta_{n,m} + \delta_{n-1,m}))e^{i\frac{2\pi}{N}lm}\\
&= \frac{1}{N}\sum_{n} e^{-i\frac{2\pi}{N}kn}(e^{i\frac{2\pi}{N}ln} - \alpha e^{i\frac{2\pi}{N}l(n+1)} + 2\alpha e^{i\frac{2\pi}{N}ln} - \alpha e^{i\frac{2\pi}{N}l(n-1)})\\
&= \underbrace{\frac{1}{N}\sum_{n} e^{-i\frac{2\pi}{N}kn} e^{i\frac{2\pi}{N}ln}}_{\delta_{kl}}(1 + 2\alpha - \alpha e^{i\frac{2\pi}{N}l} - \alpha e^{-i\frac{2\pi}{N}l})\\
&= \delta_{kl}(1 + 2\alpha - 2\alpha\cos(\frac{2\pi}{N}l))\\
&= \delta_{kl}(1 + 2\alpha(1 - \cos(\frac{2\pi}{N}l)).
\end{aligned}$$

Durch eine analoge Rechnung finden wir für die Fouriertransformierte $\tilde{N}$ der Matrix $N_{nm} = (\mathbb{1} + H)_{nm} = (\delta_{nm} + \alpha(\delta_{n+1,m} - 2\delta_{n,m} + \delta_{n-1,m}))$:

$$\begin{aligned}
\tilde{N}_{kl} &= \sum_{n,m} U_{kn} N_{nm} U^*_{lm} \qquad (8.27)\\
&= \frac{1}{N} \sum_{n,m} e^{-i\frac{2\pi}{N}kn} (\delta_{nm} + \alpha(\delta_{n+1,m} - 2\delta_{n,m} + \delta_{n-1,m})) e^{i\frac{2\pi}{N}lm}\\
&= \frac{1}{N} \sum_{n} e^{-i\frac{2\pi}{N}kn} (e^{i\frac{2\pi}{N}ln} + \alpha e^{i\frac{2\pi}{N}l(n+1)} - 2\alpha e^{i\frac{2\pi}{N}ln} + \alpha e^{i\frac{2\pi}{N}l(n-1)})\\
&= \underbrace{\frac{1}{N} \sum_{n} e^{-i\frac{2\pi}{N}kn} e^{i\frac{2\pi}{N}ln}}_{\delta_{kl}} (1 - 2\alpha + \alpha e^{i\frac{2\pi}{N}l} + \alpha e^{-i\frac{2\pi}{N}l})\\
&= \delta_{kl}(1 - 2\alpha + 2\alpha\cos(\frac{2\pi}{N}l))\\
&= \delta_{kl}(1 - 2\alpha(1 - \cos(\frac{2\pi}{N}l)).
\end{aligned}$$

Als nächtes wollen wir die Matrix $\tilde{M}$ invertieren. Da diese nun diagonal ist, können wir dies schnell erreichen.

$$M^{-1}_{nm} = (U^\dagger \tilde{M}^{-1} U)_{(nm)}, \qquad (8.28)$$

bzw.

$$\lambda_l = (1 + 2\alpha(1 - \cos(\frac{2\pi}{N}l)))^{-1}. \qquad (8.29)$$

Die gesamte Matrix L, die wir für die Berechnung der Diffusionsgleichung benötigen, ist nun durch Rücktransformation der zusammengesetzten Matrix $\tilde{L}$ gegeben, mit

$$\tilde{L} = \frac{1 - 2\alpha(1 - \cos(\frac{2\pi}{N}l))}{1 + 2\alpha(1 - \cos(\frac{2\pi}{N}l))}\delta_{kl}, \qquad (8.30)$$

und

$$\begin{aligned}
L_{nm} &= \frac{1}{N} \sum_{kl} U^\dagger_{nk} \tilde{L} U_{lm} \qquad (8.31)\\
&= \frac{1}{N} \sum_{kl} e^{i\frac{2\pi}{N}nk} \frac{1 - 2\alpha(1 - \cos(\frac{2\pi}{N}l))}{1 + 2\alpha(1 - \cos(\frac{2\pi}{N}l))} \delta_{kl} e^{i\frac{2\pi}{N}lm}\\
&= \underbrace{\frac{1}{N} \sum_{k} e^{i\frac{2\pi}{N}k(n-m)}}_{\delta_{n-m,0} = \delta_{nm}} \frac{1 - 2\alpha(1 - \cos(\frac{2\pi}{N}k))}{1 + 2\alpha(1 - \cos(\frac{2\pi}{N}k))}.
\end{aligned}$$

Damit können wir die Rekursionsformel angeben:

$$u^{(\tau)}_n = \sum_{m=1}^{N} L_{nm} u^{(\tau-1)}_m, \qquad (8.32)$$

bzw.

$$u^{(\tau)}_n = \frac{1}{N} \sum_{k,m} e^{i\frac{2\pi}{N}k(n-m)} \frac{1 - 2\alpha(1 - \cos(\frac{2\pi}{N}k))}{1 + 2\alpha(1 - \cos(\frac{2\pi}{N}k))} u^{(\tau-1)}_m. \qquad (8.33)$$

Es bleibt noch, die Rekursionsformel so auszudrücken, dass die Verteilung zur Zeit τ in Termen der Anfangskonfiguration ($\tau = 0$) gegeben ist. Dazu setzen wir u in den eigenen Rekursionsausdruck

ein und finden:

$$\begin{aligned} u_n^{(1)} &= Lu_n^{(0)} \\ u_n^{(2)} &= Lu_n^{(1)} = LLu_n^{(0)} = L^2 u_n^{(0)} \\ \vdots &= \vdots \\ u_n^{(\tau)} &= \underbrace{L \dots L}_{\tau \text{ mal}} u_n^{(0)} = L^\tau u_n^{(0)}. \end{aligned} \tag{8.34}$$

8.12 Übung 12: Lösung der Diffusionsgleichung

In dieser Übung soll die Lösung für die Diffusionsgleichung aus der vorherigen Übung implementiert werden. Dazu soll ein lokalisiertes Anfangsprofil zur Zeit $(t = 0)$ vorgegeben werden, die Verteilung zu verschiedenen Zeiten soll dann studiert werden. Die Ausbreitung der Anregung soll graphisch dargestellt werden.
In der Simulation wurde eine Ausdehnung in x mit 100 angenommen, auf der ein Gauß'sches Anfangsprofil, welches in der Mitte lokalisiert ist, aufgespannt wurde.

$$u^{(t=0)} = 50 * \exp(-\frac{(x-50)^2}{100}). \tag{8.35}$$

Das Programm liefert eine Datei `diffusion.dat`, welche 3 Spalten mit Werten beinhaltet. Diese

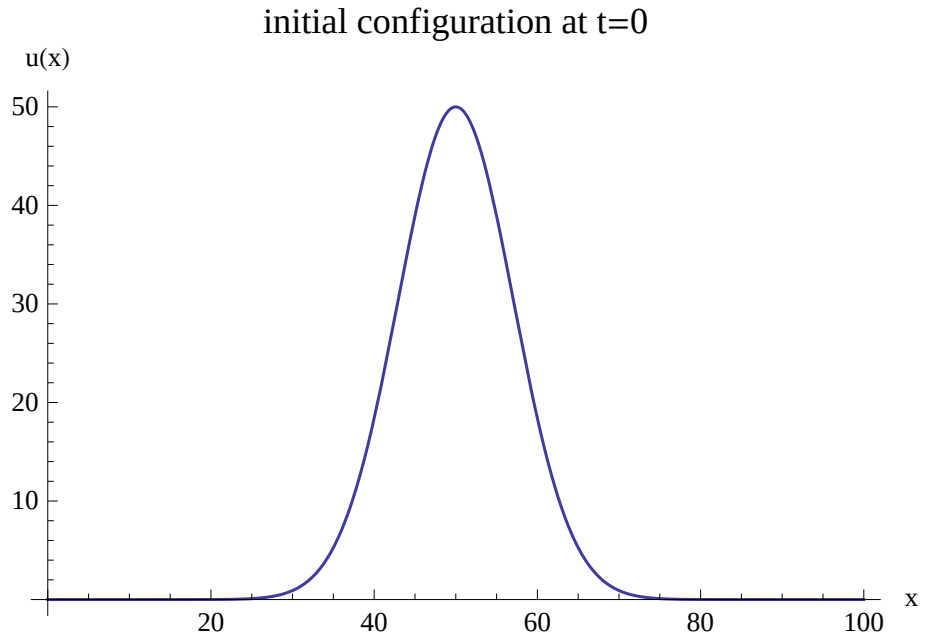


Abbildung 12: Die gewählte Anfangsverteilung

wurden im Anschluss mit dem Programm `gnuplot` in eine 3d-Graphik prozessiert. Die 3 Spalten beinhalten die Werte x, y, z für den 3d-Plot, mit den Parametern $x, \tau, u(x, \tau)$. Die Syntax für `gnuplot` ist im folgenden gelistet:

```
set style data lines
set surface
set xrange [0:100]
set yrange [0:100]
set zrange [0.0:50.0]
set xtics 10
set ytics 10
set ztics 10
set xlabel "x (arb. units)"
set ylabel "t (arb. units)"
set zlabel "u(x,t)"
set title "Solution to diffusion equation"
set view 50,10,1,1
set dgrid3d 40,51,3
```

```
set terminal postscript eps
set out 'diffusion.eps'
splot 'diffusion.dat'
```

Dies erzeugt den gezeigten 3d-Plot.

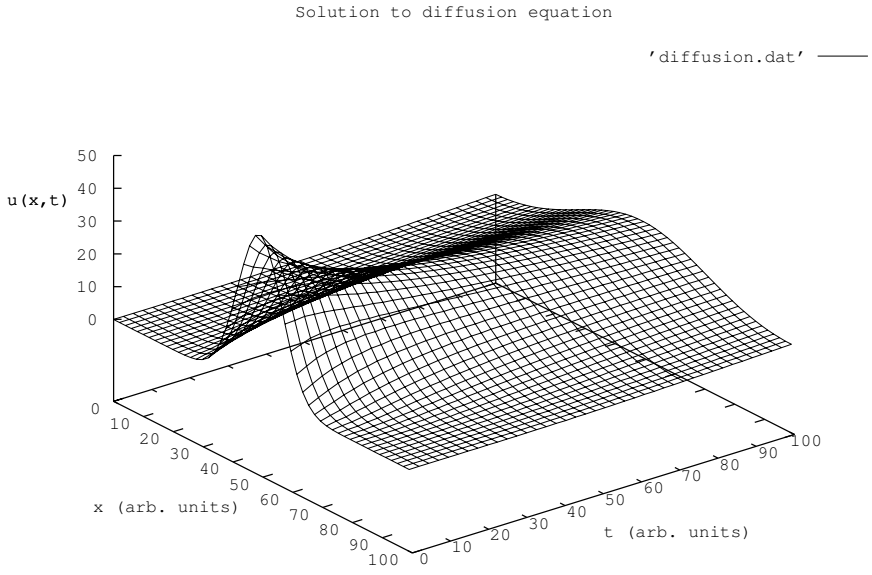


Abbildung 13: Zeitentwicklung des Anfangsprofiles, gerechnet mit der Diffusionsgleichung und dem Crank-Nicholson-Schema, Plot erzeugt mit `gnuplot`

8.12.1 Sourcecode-Uebung12.f90

```
PROGRAM main
        IMPLICIT NONE
        INTEGER, PARAMETER :: NN=100     !Groesse des Simulationsbereiches
        INTEGER, PARAMETER :: tmax=100   !Endzeit
        INTEGER :: tau=0                 !Zeitschritte
        REAL(KIND=8), PARAMETER :: PI=3.14159265358979323846264
        COMPLEX, PARAMETER :: ii=(0.,1.) !Imaginaere Einheit
        REAL(KIND=8) :: x(NN), xold(NN) !Simulationsbereich
        REAL(KIND=8) :: alpha = 1.
!alpha=D*deltat/(2*deltax^2), deltax(t)...Schrittweiten, D Diffusionskonst
        REAL(KIND=8) :: sum = 0.         !Summenparameter

        INTEGER :: i, j, k, m, n         !Schleifenparameter
        PRINT *, "Dieses Programm loest eine 1+1-dim. Diffusionsgleichung"
        PRINT *, "Uebung 12 aus Computational Physics 2, WS 09/10,
                  Prof. Gattringer, Programmiert von A. Windisch"
        PRINT *

!Beschreibe nun den Simulationsbereich mit einem lokalisierten Anfangsprofil.
Der Sim-Bereich umfasst 100 Werte von 1...100, das lokalisierte Anfangsprofil
ist ein um 50 gecenterter Gauss der Breite 100.

!Bereite Schreiben der Konfiguration vor
OPEN (UNIT=10, FILE="diffusion.dat", ACTION="write")


DO i=1, NN
  xold(i) = NN/2*EXP(-(i-NN/2.)**2/NN)
END DO


DO i=1, NN
 WRITE (10,'(I5)',ADVANCE="NO")(i-1)
 WRITE (10,'(I5)',ADVANCE="NO")(tau)
 WRITE (10,'(F8.3)')(xold(i))
END DO


!Berechne Zeitschritte

DO tau=1, tmax
 DO n=1, NN
  DO k=1, NN
   DO m=1, NN
   sum = sum + 1./NN*REAL(EXP(ii*2.*PI/NN*k*(n-m)))
*(1.-2.*alpha*(1-COS(2.*PI/NN*k)))/(1.+2.*alpha
*(1-COS(2.*PI/NN*k)))*xold(m)
   END DO !Ende m
  END DO !Ende k, also Ende der Summe
 x(n) = sum
 sum = 0.          !Summe ruecksetzen

 END DO            !Ende fuer Wert n
 xold = x          !Konfigurationen tauschen

 DO i=1, NN
  WRITE (10,'(I5)',ADVANCE="NO")(i-1)
  WRITE (10,'(I5)',ADVANCE="NO")(tau)
  WRITE (10,'(F8.3)')(xold(i))
 END DO

END DO
CLOSE (UNIT=10)
END PROGRAM main
```

8.12.2 Output des Programmes

```
 Dieses Programm loest eine 1+1-dim. Diffusionsgleichung
 Uebung 12 aus Computational Physics 2,
WS 09/10, Prof. Gattringer, Programmiert von A. Windisch
```

Ein Vergleich mit der in `Mathematica` gewonnenen Lösung mittels der Funktion `NDSolve` zeigt, dass die Lösungen übereinstimmen.

```
In[39]:= NDSolve[{D[u[t, x], t] == 2 * D[u[t, x], x, x], u[0, x] == 50 * e^(-1/100 (-50+x)^2)},
          u, {t, 0, 100}, {x, 0, 100}]

         NDSolve::bcart :
          Warning: An insufficient number of boundary conditions have been specified
               for the direction of independent variable x. Artificial
               boundary effects may be present in the solution. >>

Out[39]= {{u → InterpolatingFunction[{{0., 100.}, {0., 100.}}, <>]}}

In[40]:= Plot3D[Evaluate[u[t, x] /. %], {t, 0, 100}, {x, 0, 100},
          PlotRange → All, AxesLabel → {"t (arb. units)", "x (arb. units)", "u(x,t)"},
          PlotLabel → "Solution to diffusion equation, Mathematica's NDSolve"]
```

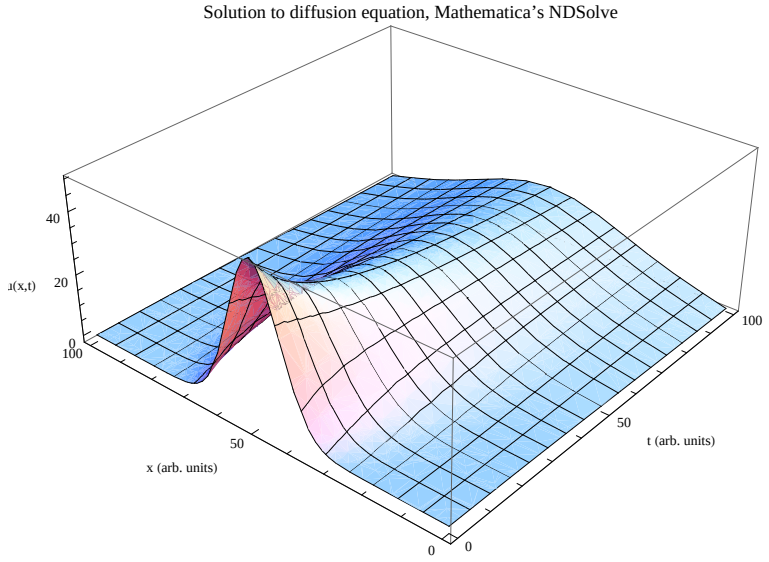


Abbildung 14: Lösung der Diffusionsgleichung zum gegebenen Anfangsprofil mit `Mathematica`

Abbildungsverzeichnis