Numerische Behandlung von linearen Gleichungssystemen

1 Der Gauß'sche Algorithmus

Der Gauß'sche Algorithmus ist schon besprochen worden. Er eignet sich zwar prinzipiell gut zur Bestimmung der Lösung eines linearen inhomogenen Gleichungssystems von der Form

$$A\,x = b\,,$$

bei der praktischen numerischen Behandlung treten aber einige Probleme auf, die durch verfeinerte Methoden vermieden bzw. gemindert werden können. Zum Beispiel ist im Fall $A \in \mathrm{GL}(n; \boldsymbol{R})$ die Anzahl der zur Gewinnung einer Lösung mit Hilfe der inversen Matrix A^{-1} in der Form

$$x = A^{-1}\,b$$

notwendigen Schritte noch relativ hoch. Man beachte, daß man in den meisten Fällen bei der Bestimmung der Lösung eines inhomogenen Gleichungssystems gar nicht alle Elemente der inversen Matrix A^{-1} kennen muß, man interessiert sich ja nur für die Lösung.
Die erweiterte Koeffizientenmatrix wird dabei durch elementare Zeilenumformungen auf Zeilenstufenform gebracht, wobei auch der Vektor b eine Veränderung erfährt. Dies ist zum Beispiel dann sehr arbeitsintensiv, wenn man – wie es in der Praxis oft der Fall ist – eine Reihe von Gleichungssystemen mit derselben Koeffizientenmatix A aber unterschiedlichem Vektor b lösen möchte.

2 Die LR–Zerlegung einer Matrix

Um die Lösung von

$$A\,x = b \quad \text{mit} \quad A \in \mathrm{GL}(n; \boldsymbol{R}), b \in \boldsymbol{R^n} \tag{1}$$

mit Hilfe von A^{-1} berechnen zu können, benötigt man i.a. n^3 wesentliche Rechenoperationen (= Multiplikationen bzw. Divisionen). Mit der im Folgenden beschriebenen Methode kann der Rechenaufwand auf ca. $n^3/3$ Operationen gesenkt werden.

2.1 LR–Zerlegung ohne Zeilenvertauschung

Voraussetzung: Die Matrix

$$A \in \mathrm{M}(m \times n; \boldsymbol{R})$$

kann durch elementare Zeilenumformungen ohne Zeilenvertauschung auf Zeilenstufenform gebracht werden:

$$A \quad \mapsto \quad B_s \cdot \ldots \cdot B_1 \cdot A = R$$

B_i ... Elementarmatrizen vom Typ $Q_i^j(\lambda)$ (entspricht der Addition der λ-fachen j-ten Zeile zur i-ten Zeile)
R ... Matrix von rechter oberer Dreiecksform

Setze

$$\tilde{L} := B_s \cdot \ldots \cdot B_1$$

Bemerkung:
- $\tilde{L}$ ist als Produkt von unteren Dreiecksmatrizen (= untere Δ–Matrizen) wieder eine untere Δ–Matrix
- $\tilde{L}$ ist wieder invertierbar, $\tilde{L}^{-1}$ ist wieder eine untere Δ–Matrix (Beachte: $(Q_i^j(\lambda))^{-1} = Q_i^j(-\lambda)$)

$$\Longrightarrow \quad A = \tilde{L}^{-1} \cdot R = L \cdot R$$

Zerlegung von A in ein Produkt einer linken unteren Δ–Matrix L mit einer rechten oberen Δ–Matrix R.

Bemerkung: Zeilenvertauschungen sind dann notwendig, wenn in einer Spalte an der Stelle, an der ein neue Stufenkante (=Pivot) entstehen soll, eine Null steht.

Definition: Für $A \in \mathrm{M}(m \times n; \boldsymbol{R})$ und $k = 1, \ldots, \min(m, n)$, bezeichnen wir mit $A_k \in \mathrm{M}(k \times k; \boldsymbol{R})$ die linke obere *Teilmatrix* von A.
Ihre Determinante $\det A_k$ heißt *Hauptminor* von A.

Satz 1: Sind alle Hauptminoren von $A \in \mathrm{M}(m \times n; \boldsymbol{R})$ ungleich Null, so besitzt A eine LR–Zerlegung.

Beispiel: Hier sei $A \in \mathrm{M}(3 \times 4; \boldsymbol{R})$

$$A = \begin{pmatrix} 2 & 1 & 0 & 4 \\ 4 & 5 & 1 & 7 \\ 2 & -8 & 1 & 12 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -3 & 1 \end{pmatrix}}_{L} \cdot \underbrace{\begin{pmatrix} 2 & 1 & 0 & 4 \\ 0 & 3 & 1 & -1 \\ 0 & 0 & 2 & 5 \end{pmatrix}}_{R}$$

Beachte: Die negativen Faktoren, die beim Gauß'schen Algorithmus zur Gewinnung von R notwendig waren, treten in den entsprechenden Spalten von L auf!

Begründung: Als Übungaufgabe.

Zusammenfassung: $A \in \mathrm{M}(m \times n; \boldsymbol{R})$ läßt sich unter der in Satz 1 angegebenen Voraussetzung gemäß

$$A = L \cdot R$$

zerlegen, wobei gilt:
$R \in \mathrm{M}(m \times n; \boldsymbol{R})$ ... rechte obere Δ–Matrix
$L \in \mathrm{GL}(m; \boldsymbol{R})$ ... linke untere Δ–Matrix

Bemerkung: Die Faktorisierung ist nicht symmetrisch, L enthält in der Hauptdiagonalen lauter 1er, $R = (r_{ij})$ enthält dort die Pivots d_i, die i.a. $\neq 1$ sind.

Ausweg:

$$R = \underbrace{\begin{pmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \\ 0 & & \cdots & & d_m \end{pmatrix}}_{=:D} \cdot \underbrace{\begin{pmatrix} 1 & r_{12}/d_1 & r_{13}/d_1 & \cdots & r_{1n}/d_1 \\ 0 & 1 & r_{23}/d_2 & \cdots & \\ \vdots & & \ddots & & \\ 0 & & \cdots & & 1 \end{pmatrix}}_{=:\hat{R}}$$

$$\Longrightarrow \quad A = L \cdot D \cdot \hat{R}$$

Bemerkung: Die Matrizen L, D und $\hat{R}$ sind in diesem Fall eindeutig bestimmt.

2.2 Anwendung

Praktisches Lösen von Gleichungssystemen der Form

$$A\,x = b$$

mit $A \in \mathrm{M}(m \times n; \boldsymbol{R})$:

$$A\,x = b \quad \Longrightarrow \quad L \underbrace{R\,x}_{=:y} = b$$

$\Longrightarrow$ 2 Gleichungssysteme

$$L\,y = b \quad \text{und} \quad R\,x = y \tag{2}$$

2 Schritte: (i) Löse $L\,y = b$ beginnend bei der ersten Gleichung, dann
(ii) löse $R\,x = y$ beginnend bei der letzten Gleichung.

Vergleich des Rechenaufwandes für ein System (1) mit $A \in \mathrm{GL}(n; \boldsymbol{R})$:

- Gauß'scher Algorithmus direkt: $\frac{1}{2}n^3 + \frac{1}{2}n^2$ wesentliche Operationen,
- Mit Hilfe der LR–Zerlegung: $\frac{1}{3}n^3 - \frac{1}{3}n$ wesentliche Operationen für die LR–Zerlegung und n^2 Operationen für das Lösen der Systeme (2).

Hinweise:

1. Für L und R muß nicht extra Speicherplatz reserviert werden. Ihre Elemente können in einer Matrix abgespeichert werden, da ja nur 2 Δ–Matrizen benötigt werden.

2. Dieses Lösungsverfahren ist besonders dann interessant, wenn es sich um p Systeme von Gleichungen der Form

   $$A\,x_i = b_i\,, \quad i = 1, \ldots, p$$

   handelt, bei denen sich die Matrix A nicht ändert. Die LR–Zerlegung ist dann nur einmal notwendig!
   Bei 10 Systemen mit $n = 100$ liegt der Unterschied schon bei $\approx 4.6\ 10^6$ Operationen.

2.3 LR–Zerlegung mit Zeilenvertauschung

Beispiel:

$$A = \begin{pmatrix} 1 & 3 & 2 \\ 2 & 6 & 9 \\ 1 & 5 & 6 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 3 & 2 \\ 0 & 0 & 5 \\ 0 & 2 & 4 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 3 & 2 \\ 0 & 2 & 4 \\ 0 & 0 & 5 \end{pmatrix} = R$$

$$R = P_3^2 \cdot Q_3^1(-1) \cdot Q_2^1(-2) \cdot A$$

Hätte man im System $Ax = b$ von vornerherein die 2. und die 3. Gleichung vertauscht, so wäre der Gauß'sche Algorithmus ohne Zeilenvertauschung möglich gewesen.
P ... Permutationsmatrix, die die notwendigen Zeilenvertauschungen bewirkt.

$$\Longrightarrow \quad P \cdot A = L \cdot R$$

d.h. PA kann jetzt ohne Zeilenvertauschung in ein Produkt LR zerlegt werden.
Für das Gleichungssystem bedeutet dies:

$$P\cdot \mid \quad Ax = b \qquad \Longrightarrow \qquad \underbrace{PA}_{LR} \cdot x = Pb \quad \Longrightarrow \quad LRx = Pb$$

Man hat also die Systeme

$$Ly = Pb \quad \text{und} \quad Rx = y$$

sukzessive zu lösen.

Satz 2: Zu jeder regulären Matrix A existiert eine Permutationsmatrix P, so daß PA in das Produkt LR zerlegt werden kann.

2.4 Cholesky – Zerlegung

Für eine quadratische, symmetrische und positiv definite Matrix $A \in M(n \times n)$ kann man auch eine Zerlegung in der Form

$$A = L \cdot L^T$$

angeben, wobei $L \in M(n \times n)$ eine linke untere Dreiecksmatrix ist und L^T die zu L transponierte Matrix bezeichnet.
Die Elemente l_{ik} von L können der Reihe nach spaltenweise berechnet werden, wobei man bei der ersten Spalte beginnt. Für die Berechnung der Elemente der k-ten Spalte lauten die Formeln

$$l_{kk} = \begin{cases} \sqrt{a_{11}} & \text{für } k = 1 \\ \sqrt{a_{kk} - \sum_{\mu=1}^{k-1} l_{k\mu}^2} & \text{für } k = 2, \ldots, n \end{cases}$$

$$l_{ik} = \begin{cases} \dfrac{a_{i1}}{l_{11}} & \text{für } \begin{matrix} k = 1 \\ i = 2, \ldots, n \end{matrix} \\ \dfrac{a_{ik} - \sum_{\mu=1}^{k-1} l_{i\mu} l_{k\mu}}{l_{kk}} & \text{für } \begin{matrix} k = 2, \ldots, n \\ i = k+1, \ldots, n \end{matrix} \end{cases}$$

3 Anwendungen

Ziele:

1. An Hand eines einfachen Beispiels aus dem Bereich der Differentialgleichungen soll die Herkunft großer Gleichungssystem aufgezeigt werden.
2. Es soll gezeigt werden, welche speziellen Eigenschaften die Koeffizientenmatrizen oft haben (z.B. Bandmatrizen).

Beispiel: Gesucht ist eine Funktion $u \in \mathcal{C}^2[0,1]$ mit der Bedingung

$$-\frac{d^2\,u}{d\,x^2} = f(x) \quad \text{für} \quad x \in [0,1] \tag{3}$$

wobei f eine vorgelegte Funktion sei.

Bemerkung:

1. Dieses Problem kann natürlich sofort mit den Mitteln der Analysis gelöst werden, es soll aber hier als einfaches Beispiel für die Gewinnung eines diskreten Problems (= lineares Gleichungssystem) dienen.
2. Ist $u(x)$ eine Lösung von (3), so ist auch

   $$v(x) = u(x) + C + Dx\,, \quad C, D \in \boldsymbol{R}$$

   Lösung. Mit den Randwerten

   $$u(0) = 0, \quad u(1) = 0$$

   wird u eindeutig festgelegt (= *Zwei–Punkt–Randwertproblem*).
3. Dieses RWP entspricht dem Wärmeleitungsproblem in einem Stab der Länge 1 mit den beiden Enden auf der Temperatur 0^o und vorgegebener Wärmequelle $f(x)$.

Wir approximieren dieses Problem durch ein diskretes Problem:
Wir suchen nicht nach den Funktionswerten von u in allen Punkten von $[0,1]$ (dies würde jeden Computer überfordern), sondern nur nach den Funktionswerten an diskreten Stellen des Intervalls.
Dazu wird das Intervall $[0,1]$ in $n+1$ Teilintervalle unterteilt:

$$[0,1] = [0,h] \cup [h,2h] \cup \ldots \cup [n\,h, (n+1)h]$$

Die Funktion f sei an den Zwischenstellen (= Stützstellen)

$$x_k = k\,h\,,\ k = 0, \ldots, n+1, \quad \text{mit} \quad x_0 = 0, x_{n+1} = (n+1)h = 1$$

gegeben.
Wir können nun näherungsweise die wahren Werte von u an diesen Stellen berechnen.
Bezeichnungsweise:

$$\begin{aligned} u_0 &= u(0) = 0 \\ u_1 &= u(1 \cdot h) \\ &\vdots \\ u_n &= u(n\,h) \\ u_{n+1} &= u((n+1)h) = 0 \end{aligned}$$

Approximation der Ableitung $\frac{d^2\,u}{d\,x^2}$:

Jede Ableitung ist der Grenzwert eines Differenzenquotienten:

$$\frac{d\,u}{d\,x} = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}$$

Wir approximieren nun:

$$\frac{d\,u}{d\,x} \approx \frac{u(x+h) - u(x)}{h}$$

oder

$$\frac{d\,u}{d\,x} \approx \frac{u(x) - u(x-h)}{h} \quad \text{oder} \quad \frac{d\,u}{d\,x} \approx \frac{u(x+h) - u(x-h)}{2h}$$

Wegen ihrer Symmetrie wählen wir die letzte Näherung.
Für die Approximation der zweiten Ableitung sei nur eine der vielen Möglichkeiten angegeben:

$$\begin{aligned} \frac{d^2u}{dx^2} &= \frac{d}{dx}\left(\frac{d\,u}{d\,x}\right) \\ &\approx \frac{u'(x+h) - u'(x)}{h} \\ &\approx \left(\frac{u(x+h) - u(x)}{h} - \frac{u(x) - u(x-h)}{h}\right) \Big/ h \\ &= \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \end{aligned}$$

An der Stelle $x_k = k\,h$ wird die Differentialgleichung (3) jetzt durch die diskrete Gleichung

$$-\frac{u(x_k+h) - 2u(x_k) + u(x_k-h)}{h^2} = f(x_k)$$

approximiert, d.h. man erhält

$$-u_{k+1} + 2u_k - u_{k-1} = h^2 f(x_k)\,, \quad k = 1, \ldots, n\,.$$

Das sind nun n lineare Gleichungen für die n Unbestimmten $u_1, \ldots, u_n$. Die erste und die letzte Gleichung enthalten noch u_0 bzw. u_{n+1} , die aber bekannt sind.

Speziell: Wähle $n = 5$, d.h. $h = 1/6$

$$\begin{aligned}
k = 1: \quad & -u_2 + 2u_1 - \overbrace{u_0}^{=0} = \frac{1}{6^2} f(\frac{1}{6}) \\
k = 2: \quad & -u_3 + 2u_2 - u_1 = \frac{1}{6^2} f(\frac{2}{6}) \\
& \vdots \\
k = 5: \quad & -\underbrace{u_6}_{=0} + 2u_5 - u_4 = \frac{1}{6^2} f(\frac{5}{6})
\end{aligned}$$

In Matrizenschreibweise:

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{pmatrix} = \frac{1}{6^2} \begin{pmatrix} f(\frac{1}{6}) \\ \\ \vdots \\ \\ f(\frac{5}{6}) \end{pmatrix}$$

Die Koeffizientenmatrix A hat (auch für den Fall $n \in \boldsymbol{N}$) folgende Eigenschaften:

1. $A = (a_{ij})$ ist *tridiagonal*, d.h. alle Elemente außerhalb der Hauptdiagonalen und der beiden anschließenden Nebendiagonalen sind Null (= *Bandmatrix*), d.h.
$$a_{ij} = 0 \quad \text{für} \quad |i - j| > 1$$
2. A ist *symmetrisch*, d.h. ${}^tA = A$. Dies ist nicht der Fall, wenn in der Differentialgleichung Ableitungen ungerader Ordnung (u' bzw. u''' etc.) auftreten.
3. A ist *positiv definit*.

Bemerkung: (Siehe Lineare Algebra 2)
Eine symmetrische Matrix $A \in \mathrm{M}(n \times n; \boldsymbol{R})$ heißt *positiv definit* genau dann, wenn eine der folgenden Bedingungen erfüllt ist:

1. ${}^t x\, A\, x > 0 \quad \forall\, x \in \boldsymbol{R^n}, x \neq 0$.
2. Alle *Eigenwerte* λ_i von A sind > 0.
(Der Begriff Eigenwert wird in LA2 behandelt werden.)
3. Alle Teilmatrizen A_k, $k = 0, \ldots, n$, besitzen eine positive Determinante, d.h. alle Hauptminoren sind > 0.
4. Alle Pivots d_i (ohne Zeilenvertauschung) sind positiv, d.h. $d_i > 0$.

Gauß'sches Verfahren auf A angewandt:

$$A \mapsto \begin{pmatrix} 2 & -1 & & & \\ & 3/2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

ist wieder tridiagonal!
Weitere Eliminationsschritte liefern wieder Marizen mit derselben Eigenschaft:

$$\Longrightarrow \quad A = \underbrace{\begin{pmatrix} 1 & & & & \\ -\frac{1}{2} & 1 & & & \\ & -\frac{2}{3} & 1 & & \\ & & -\frac{3}{4} & 1 & \\ & & & -\frac{4}{5} & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} \frac{2}{1} & & & & \\ & \frac{3}{2} & & & \\ & & \frac{4}{3} & & \\ & & & \frac{5}{4} & \\ & & & & \frac{6}{5} \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} 1 & -\frac{1}{2} & & & \\ & 1 & -\frac{2}{3} & & \\ & & 1 & -\frac{3}{4} & \\ & & & 1 & -\frac{4}{5} \\ & & & & 1 \end{pmatrix}}_{R}$$

$$A = L \cdot D \cdot R \qquad \ldots \quad LR\text{–Zerlegung von } A$$

Anmerkung:

- L, R sind bidiagonal, sie liegen in derselben Bandbreite, wie A
- ${}^tL = R$
- Die Pivots d_i in D sind alle positiv, ferner gilt

$$\prod_{i=1}^{n} d_i = \det A = 6$$

- $\lim_{i \to 0} d_i = 1$

Das sind Eigenschaften, die für numerische Verfahren sehr vorteilhaft sind.

Rechenaufwand: Will man ein lineares Gleichungssystem mit einer tridiagonalen Matrix lösen, so ist der Rechenaufwand

$$N_{\text{Band}} = \frac{2}{3}(3n - 2) \quad \text{Mult./Div.}$$

hingegen benötigt man bei einer voll besetzte Matrix

$$N_{\text{Voll}} = \frac{1}{3}(n^3 + 3n^2 - n) \quad \text{Mult./Div.}$$

Für ein typisches Problem mit $n = 150$ hat man

$$N_{\text{Voll}} \approx 1.1 \cdot 10^6 \quad \text{gegenüber} \quad N_{\text{Band}} \approx 300$$

Beispiel:

$$\begin{aligned} -u'' &= 4\pi^2 \sin 2\pi x \\ u(0) &= u(1) = 0 \end{aligned}$$

a) Wähle $n = 3$, d.h. $h = \frac{1}{4}$

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \frac{2\pi^2}{4^2} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

$$\Longrightarrow \qquad u_1 = \frac{\pi^2}{8}, u_2 = 0, u_3 = -\frac{\pi^2}{8}$$

Im linken Bild sind die drei Funktionswerte durch gerade Stecken verbunden.
Vergleich mit der exakten Lösung (rechtes Bild):

$$u(x) = \sin 2\pi x$$

b) Wähle $n = 9$, d.h. $h = \frac{1}{10}$. Die Berechnung führt auf die in obigem mittleren Bild strichliert gezeichnete Kurve.

4 Rundungsfehler, Pivotstrategien

In einem Rechner wird jede reelle Zahl a in Form einer Gleitkommazahl abgespeichert, z.B.

$$a = \underbrace{-0.1234567}_{\text{Mantisse}} \cdot \underbrace{10^{-9}}_{\text{Exponent}}$$

Dabei ist die Mantisse eine vorzeichenbehaftete t–stellige Zahl, wobei
Taschenrechner: $t = 10$
PC (Matlab): $t = 16$

I.a. wird jede Zahl $a \in \boldsymbol{R}$ gerundet angegeben, speziell Meßdaten.

Definition: $[a + b]$ bezeichnet die auf t Stellen gerundete Gleitkommazahl der Summe von a und b. Differenz, Produkt, Quotient analog.

$\Longrightarrow$ Rundungsfehler
Beispiel: $t = 2$

$$\text{Berechne:} \qquad \left(\frac{2}{3} + \frac{2}{3}\right) - \frac{1}{3}$$

$$\left[\frac{1}{3}\right] = 0.33\,, \ \left[\frac{2}{3}\right] = 0.67\,, \ [0.67 + 0.67] = [1.34] = 1.3$$

$$[1.3 - 0.33] = [0.97] = 0.97 \text{ !!}$$

4.1 Auswahl geeigneter Pivots

Bemerkung: Manche Matrizen sind extrem sensibel gegenüber kleinen Änderungen ihrer Elemente, andere wiederum nicht.

Beispiel:

$$A = \begin{pmatrix} 1. & 1. \\ 1. & 1.0001 \end{pmatrix}, \; B = \begin{pmatrix} 0.0001 & 1. \\ 1. & 1. \end{pmatrix}$$

A ist sensibel, man sagt auch *schlecht konditioniert*
B ist gut konditioniert

A ist fast«ßingulär ($\det A = 0.0001$), ändert man a_{22} auf 1.0 ab, so ist $\det A = 0$
B ist regulär.
Ein Maß dafür, wie gut eine Matrix konditioniert ist, stellt die *Konditionszahl* $\mathrm{cond}(A)$ dar (siehe Lineare Algebra 2):

$$\mathrm{cond}(A) \approx 4000\,, \; \mathrm{cond}(B) \approx 4$$

Betrachte zwei verschieden rechte Seiten für das Gleichungssystem $A \cdot x = b$:

$$b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad b' = \begin{pmatrix} 2 \\ 2.0001 \end{pmatrix}$$

1. System:

$$\begin{array}{r} u + v = 2 \\ u + 1.0001\, v = 2 \end{array} \quad \Longrightarrow \quad \begin{array}{l} u = 2 \\ v = 0 \end{array}$$

2. System:

$$\begin{array}{r} u + v = 2 \\ u + 1.0001\, v = 2.0001 \end{array} \quad \Longrightarrow \quad \begin{array}{l} u = 1 \\ v = 1 \end{array}$$

Bemerkung: Eine Änderung in der 5. Stelle der 2. Komponente von b hat eine Vergrößerung der Änderung in der ersten Stelle von u und v zur Folge.

Keine numerische Methode kann diese Sensibilität gegenüber kleinen Änderungen in den Daten verkleinern!

Bemerkung: Auch eine gut konditionierte Matrix kann durch einen schlechten Algorithmus ruiniert werden!

Beispiel:

$$B\,x = c \quad \text{mit} \quad c = {}^t(1, 2)$$

Schlecht: Gauß'scher Algorithmus unmittelbar angewandt:

$$\begin{array}{r} 0.0001\, u + v = 1 \\ u + v = 2 \end{array} \quad \Longrightarrow \quad \left(\begin{array}{cc|c} 0.0001 & 1 & 1 \\ 0 & -9\,999 & -9\,998 \end{array} \right) \quad \Longrightarrow \quad v_{\text{exakt}} = 0.99990$$

Rundung auf 3 Dezimalen liefert:

$$-10\,000\, v = -10\,000 \quad \Longrightarrow \quad v_{\text{ger}} = 1$$

d.h. v_{ger} ist korrekt auf 3 Dezimalen.
1. Gleichung ungerundet:

$$0.0001\,u + 0.9999 = 1 \quad \Longrightarrow \quad u_{\text{exakt}} = 1$$

1. Gleichung gerundet:

$$0.0001\,u + 1 = 1 \quad \Longrightarrow \quad u_{\text{ger}} = 0 \;\; !!$$

u_{ger} ist gänzlich falsch!
Obwohl B gut konditioniert ist, ist das direkte Verfahren extrem instabil!
Zerlegung von B:

$$B = \underbrace{\begin{pmatrix} 1 & 0 \\ 10\,000 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 0.0001 & 0 \\ 0 & -9\,999 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} 1 & 10\,000 \\ 0 & 1 \end{pmatrix}}_{R}$$

Achtung: Die Pivots sind von extrem unterschiedlicher Größenordnung!
Der *kleine* Pivot 0.0001 brachte das Malheur.

Abhilfe: Zeilenvertauschung.

Bemerkung: So wie ein Null-Pivot«ëine Zeilenvertauschung erfordert, so erfordert ein sehr kleiner Pivot eine praktische Zeilenvertauschung. Ein gutes Computerprogramm sollte alle möglichen Pivots in einer Spalte miteinander vergleichen. Wählt man den größten der Kandidaten und vertauscht man dann die Zeilen entsprechend, so spricht man von einem *partiellen Pivotieren.*

Beispiel:

$$B = \begin{pmatrix} 0.0001 & 1. \\ 1. & 1. \end{pmatrix}, \quad B' = \begin{pmatrix} 1. & 1. \\ 0.0001 & 1. \end{pmatrix}$$

$$B' = \underbrace{\begin{pmatrix} 1. & 0 \\ 0.0001 & 1. \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 1. & 0 \\ 0 & 0.9999 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} 1 & 0.0001 \\ 0 & 1 \end{pmatrix}}_{R}$$

Die Piovots sind alle von der selben Größenordnung.
Eine mögliche Erweiterung der Strategie bildet eine geeignete

4.2 Skalierung der Gleichungen

Beispiel: $2 \cdot x_1 + x_2 + x_3 = 1$
$x_1 + \varepsilon \cdot x_2 + \varepsilon \cdot x_3 = 2 \cdot \varepsilon \qquad \varepsilon \ll 1$
$x_1 + \varepsilon \cdot x_2 - \varepsilon \cdot x_3 = \varepsilon$

$$\left(A, \vec{b}\right) = \left(\begin{array}{ccc|c} 2 & 1 & 1 & 1 \\ 1 & \varepsilon & \varepsilon & 2\varepsilon \\ 1 & \varepsilon & -\varepsilon & \varepsilon \end{array}\right) \begin{array}{l} \\ \cdot(-1/2) \\ \cdot(-1/2) \end{array}$$

$$\rightsquigarrow \left(\begin{array}{ccc|c} 2 & 1 & 1 & 1 \\ 0 & \varepsilon - \frac{1}{2} & \varepsilon - \frac{1}{2} & 2\varepsilon - \frac{1}{2} \\ 0 & \varepsilon - \frac{1}{2} & -\varepsilon - \frac{1}{2} & \varepsilon - \frac{1}{2} \end{array}\right) \cdot(-1)$$

$$\rightsquigarrow \begin{pmatrix} 2 & 1 & 1 & \mid 1 \\ 0 & \varepsilon-\frac{1}{2} & \varepsilon-\frac{1}{2} & \mid 2\varepsilon-\frac{1}{2} \\ 0 & 0 & -2\varepsilon & \mid -\varepsilon \end{pmatrix} \qquad \text{exakte Lösung:} \quad \begin{array}{l} x_1 = \frac{-\varepsilon}{2\varepsilon-1} \\ x_2 = \frac{6\varepsilon-1}{(2\varepsilon-1)\cdot 2} \\ x_3 = \frac{1}{2} \end{array}$$

Rechner: $\varepsilon = 0.01\ ,\ t = 2$:

$$\left(A, \vec{b}\right) \rightsquigarrow \begin{pmatrix} 2 & 1 & 1 & 1 \\ 0 & -0.5 & -0.5 & -0.5 \\ 0 & -0.5 & -0.5 & -0.5 \end{pmatrix} \quad \Rightarrow \quad \begin{array}{l} x_1 = 0 \\ x_2 = 1-\sigma \\ x_3 = \sigma\ ,\ \sigma \in \boldsymbol{R} \end{array} \qquad \text{keine eindeutige Lösung!}$$

Läßt man hingegen in der exakten Lösung $\varepsilon \to 0$ gehen, so folgt:

$$x_1 = 0, x_2 = 1/2, x_3 = 1/2\,.$$

Dies erhält man in der Gleitkommadarstellung mit $\sigma = 1/2$!
D.h. diese Lösung ist unbefriedigend.

Ausweg: Skalierung der einzelnen Gleichungen
d.h. jede Gleichung wird zunächst mit einem geeigneten Faktor multipliziert und anschließend wird eine Variablentransformation durchgeführt.

$$\begin{array}{l} 2\cdot x_1 + x_2 + x_3 = 1 \\ \frac{x_1}{\varepsilon} + x_2 + x_3 = 2 \\ \frac{x_1}{\varepsilon} + x_2 - x_3 = 1 \end{array} \qquad \text{Transf.:} \quad \begin{array}{l} \frac{x_1}{\varepsilon} = z_1 \\ x_2 = z_2 \\ x_3 = z_3 \end{array}$$

$$\left(\tilde{A}, \tilde{\vec{b}}\right) = \begin{pmatrix} 2\varepsilon & 1 & 1 & \mid 1 \\ 1 & 1 & 1 & \mid 2 \\ 1 & 1 & -1 & \mid 1 \end{pmatrix} \overset{\text{Zeilenvert.}}{\rightsquigarrow} \begin{pmatrix} 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 2 \\ 2\varepsilon & 1 & 1 & 1 \end{pmatrix} \mid\cdot(-1) \mid\cdot(-2\varepsilon)$$

$$\begin{pmatrix} 1 & 1 & -1 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & [1-2\varepsilon] & [1+2\varepsilon] & [1-2\varepsilon] \end{pmatrix} \overset{\text{Zeilenvert.}}{\rightsquigarrow} \begin{pmatrix} 1 & 1 & -1 & 1 \\ 0 & [1-2\varepsilon] & [1+2\varepsilon] & [1-2\varepsilon] \\ 0 & 0 & 2 & 1 \end{pmatrix}$$

exakte Lösung: $z_3 = 1/2, z_2 = \frac{6\varepsilon-1}{4\varepsilon-2}, z_1 = \frac{-1}{2\varepsilon-1}$

$$\Rightarrow \quad x_3 = 1/2, x_2 = \frac{6\varepsilon-1}{4\varepsilon-2}, x_1 = \frac{-\varepsilon}{2\varepsilon-1}$$

Mit $t = 2$ und $\varepsilon = 0.01$ liefert der Rechner:

$$\begin{pmatrix} 1 & 1 & -1 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \quad \Rightarrow \quad \begin{array}{l} z_1 = 1 \\ z_2 = \frac{1}{2} \\ z_3 = \frac{1}{2} \end{array} \quad \Rightarrow \quad \begin{array}{l} x_1 = \varepsilon \\ x_2 = \frac{1}{2} \\ x_3 = \frac{1}{2} \end{array}$$

Günstiger Weg: Skaliere das Gleichungssystem so, daß im neuen System $\tilde{A}\cdot\vec{z} = \tilde{\vec{b}}$ gilt:
$\max_{1\le j\le n}\{\mid \tilde{a}_{ij} \mid\} = 1$ mit $i = 1, 2, \ldots, n$

Zusammenfassung:
Geeignete Skalierung des Systems führt i.a. zu einer Verbesserung des Ergebnisses.
Gute Programme zur numerischen Lösung von Gleichungssystemen skalieren die Gleichungen automatisch in bestimmter Weise, z.B. unter der Bedingung

$$\max_{1\le j\le n}\{\mid \tilde{a}_{ij} \mid\} = 1 \quad \text{mit} \quad i = 1, 2, \ldots, n$$

5 Iterative Methoden zur Lösung von linearen Gleichungssystemen

Will man mit Hilfe des Gauß'schen Algorithmus bzw. der LR–Zerlegung ein System von der Form

$$Ax = b\,, \quad A \in \mathrm{GL}(n; \boldsymbol{R})$$

lösen, so wächst die Anzahl der erforderlichen wesentlichen Rechenschritte mit der dritten Potenz von n was natürlich viel Rechenzeit kosten kann. Exakte Methoden können aber i.a. nicht mehr wesentlich verbessert werden.
Gesucht sind also schnellere Verfahren zur Bestimmung einer Näherungslösung. Man startet bei einem anfänglich irgendwie gewählten Vektor x_0, berechnet dann x_1 aus x_0 und allgemein eine verbesserte Lösung x_{k+1} aus der vorangegangenen Approximation x_k. Man erhält so eine Folge von Approximationen $\{x_k\}$.
Nun zur Vorgangsweise: Man spaltet die Matrix A auf in eine Summe von zwei Matrizen

$$A = S - T \tag{4}$$

und erhält dann die Gleichung

$$Sx = Tx + b$$

Damit kann man eine Iteration formulieren:

$$Sx_{k+1} = Tx_k + b\,, \quad k = 0, 1, 2, \ldots \tag{5}$$

Es gibt aber keine Garantie dafür, daß dieser Weg zum Ziel führt. Die Aufspaltung (4) muß 2 Bedingungen erfüllen:

- Der neue Vektor x_{k+1} soll leicht berechenbar sein. Deshalb soll S einfach aufgebaut und invertierbar sein, S sollte z.B. diagonal oder tridiagonal sein.
- Die Folge $\{x_k\}$ von Vektoren soll gegen die wahre Lösung konvergieren:

$$\left.\begin{array}{c} Sx = Tx + b \\ Sx_{k+1} = Tx_k + b \end{array}\right\} - \qquad S(x - x_{k+1}) = T(x - x_k)$$

  Mit dem *Fehlervektor* $v_k := x - x_k$ gilt also

$$Sv_{k+1} = Tv_k\,, \quad k = 0, 1, 2, \ldots$$

$$v_k = S^{-1}Tv_{k-1} = \ldots = \left(S^{-1}T\right)^k v_0$$

$$\text{Konvergenz für } k \to \infty: \qquad x_k \;\to\; x_0 \quad \Longleftrightarrow \quad v_k \;\to\; 0$$

Satz 3: Die Iteration (5) ist genau dann konvergent, wenn für jeden *Eigenwert* λ von $S^{-1}T$ gilt:

$$|\lambda| < 1$$

Die Konvergenzgüte hängt von der maximalen Größe von $|\lambda|$ ab, die als *Spektralradius* ρ von $S^{-1}T$ bezeichnet wird:

$$\rho(S^{-1}T) := \max_i |\lambda_i|$$

Die Konvergenzrate r ist z.B. gegeben durch

$$r = -\log_{10} \rho$$

Anmerkung: Die Begriffe *Eigenwert* und *Spektralradius* werden in der Vorlesung Lineare Algebra 2 behandelt.

Unter anderem hat man zwei (extreme) Möglichkeiten, A aufzuspalten:

- $A = A + 0$, d.h. $S = A, T = 0$:
  Die Iteration lautet dann

$$Ax_1 = b$$

$$\Longrightarrow \qquad S^{-1}T = 0\,, \quad \rho(S^{-1}T) = 0\,, \quad r = -\log_{10} 0 = \infty$$

  Aber: $S = A$ kann u.U. schwer invertiert werden. Das war ja das ursprüngliche Problem!

- $$S = \mathrm{diag}(a_{11}, \ldots, a_{nn})$$
  d.h. man nimmt die Elemente aus der Diagonalen von A.
  Aus dem Gleichungssystem gewinnt man jetzt mit der Bezeichnung:

  $(x_i)_k$ ... i–te Komponente des Vektors x_k, der im k–ten Schritt berechnet worden ist

  folgendes System

$$\begin{aligned} a_{11}(x_1)_{k+1} &= (-a_{12}x_2 - \ldots - a_{1n}x_n)_k + b_1 \\ &\vdots \\ a_{nn}(x_n)_{k+1} &= (-a_{n1}x_2 - \ldots - a_{n,n-1}x_{n-1})_k + b_n \end{aligned}$$

  Ist $a_{ii} \neq 0, i = 1, \ldots, n$, und ist A *dünn* besetzt (d.h. nur *wenige* Elemente von A sind $\neq 0$), dann funktioniert dieses Verfahren (= JACOBI–*Approximation*) recht gut.

  Frage: • Konvergenz
  • Güte der Konvergenz (=Konvergenzrate)

Beispiel:

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}\,, \quad S = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}\,, \quad T = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\,, \quad S^{-1}T = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}$$

$$\text{Mit} \quad x = \begin{pmatrix} v \\ w \end{pmatrix} \qquad \text{gilt jetzt} \qquad \begin{aligned} 2v_{k+1} &= w_k + b_1 \\ 2w_{k+1} &= v_k + b_2 \end{aligned} \qquad \text{oder}$$

$$\begin{pmatrix} v \\ w \end{pmatrix}_{k+1} = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}_k + \begin{pmatrix} b_1/2 \\ b_2/2 \end{pmatrix}$$

Die Eigenwerte von $S^{-1}T$ sind $\lambda_{1/2} = \pm 1/2$, der Spektralradius ist demnach $\rho = 1/2$; mit $r = -\log_{10} \rho \approx 0.3$ liegt eine gute Konvergenz vor.
Der Fehler halbiert sich bei jedem Schritt, eine weitere binäre Stelle wird korrekt.

Nachteil: Bei großen Systemen muß immer der gesamte Vektor x_k abgespeichert werden.

Neue Idee: Man verwendet jede Komponente des neu berechneten Vektors x_{k+1} sobald sie ermittelt worden ist (= GAUSS–SEIDEL–*Iteration*):

1. Gleichung (wie vorher):

$$a_{11}(x_1)_{k+1} = (-a_{12}x_2 - \ldots - a_{1n}x_n)_k + b_1$$

2. Gleichung (verwendet schon den neuen Wert von x_1):

$$a_{22}(x_2)_{k+1} = -a_{21}(x_1)_{k+1} + (-a_{23}x_3 - \ldots - a_{2n}x_n)_k + b_2$$

$\vdots$

n–te Gleichung:

$$a_{nn}(x_n)_{k+1} = (-a_{n1}x_1 - \ldots - a_{n,n-1}x_{n-1})_{k+1} + b_n$$

Beispiel:

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, \quad S = \begin{pmatrix} 2 & 0 \\ -1 & 2 \end{pmatrix}, \quad T = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad S^{-1}T = \begin{pmatrix} 0 & 1/2 \\ 0 & 1/4 \end{pmatrix}$$

$$\begin{aligned} 2v_{k+1} &= w_k + b_1 \\ 2w_{k+1} &= v_{k+1} + b_2 \end{aligned}$$

$$\begin{pmatrix} 2 & 0 \\ -1 & 2 \end{pmatrix} x_{k+1} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x_k + b$$

Die Eigenwerte von $S^{-1}T$ sind $\lambda_1 = 0, \lambda_2 = 1/4$, der Spektralradius ist demnach $\rho = 1/4$, die Konvergenzrate $r = -\log_{10}\rho \approx 0.6$.
Der Fehler wird bei jedem Schritt mit dem Faktor $1/4$ multipliziert. Ein Gauss–Seidel–Schritt ist soviel wert wie zwei Jacobi–Schritte.

Satz 4: Für eine symmetrische Matrix $A \in \mathrm{M}(n \times n; \boldsymbol{R})$ konvergiert die Gauss–Seidel–Iteration genau dann, wenn A positiv definit ist.

Beispiel:

$$Ax = b \quad \text{mit} \quad A = \begin{pmatrix} 3 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \end{pmatrix} \tag{6}$$

$$\begin{array}{lcccccc} x_1 & = & & 0.333x_2 & & & +0.333 \\ x_2 & = & 0.333x_1 & & +0.333x_3 & & +0.667 \\ x_3 & = & & 0.333x_2 & & +0.333x_4 & \\ x_4 & = & & & 0.333x_3 & & +0.333 \end{array}$$

Jacobi–Iteration für das System (6):

Iterations-schritt	x_1	x_2	x_3	x_4
0	0.333	0.667	0.000	0.333
1	0.556	0.778	0.333	0.333
2	0.593	0.963	0.370	0.444
3	0.654	0.988	0.469	0.457
4	0.662	1.041	0.481	0.490
5	0.680	1.048	0.510	0.494
⋮	⋮	⋮	⋮	⋮
10	0.690	1.072	0.526	0.509
11	0.691	1.072	0.527	0.509
12	0.691	1.073	0.527	0.509
13	0.691	1.073	0.527	0.509

GAUSS–SEIDEL–Iteration für das System (6):

Iterations-schritt	x_1	x_2	x_3	x_4
0	0.333	0.667	0.000	0.333
1	0.556	0.852	0.395	0.465
2	0.617	1.004	0.490	0.497
3	0.668	1.052	0.516	0.505
4	0.684	1.067	0.524	0.508
5	0.689	1.071	0.526	0.509
6	0.690	1.072	0.527	0.509
7	0.691	1.073	0.527	0.509
8	0.691	1.073	0.527	0.509

Gegenbeispiel:

$$\begin{array}{rcrcrcl} x & + & 2y & & & = & 1 \\ 2x & + & y & + & 2z & = & 0 \\ & & 2y & + & z & = & 1 \end{array} \quad , \quad A = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix} \tag{7}$$

Exakte Lösung:

$$x = -\frac{1}{7}\,,\ y = \frac{4}{7}\,,\ z = -\frac{1}{7}$$

JACOBI– und GAUSS–SEIDEL–Iteration für das System (7):

Iterations-schritt	Jacobi			Gauß–Seidel		
	x	y	z	x	y	z
1	1	0	1	1	0	1
2	1	-4	1	1	-4	9
3	9	-4	9	9	-36	73
4	9	-36	9	73	-292	585
5	73	-36	73	585	-2340	4681
6	73	-292	73			

Die Iteration ist stark divergent!

Definition: $A \in \mathrm{M}(n \times n; \boldsymbol{R})$ heißt *streng diagonal dominant (s.d.d.)*, wenn

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \forall\, i = 1, \ldots, n\,.$$

Bemerkung: A ist streng diagonal dominant

$$\Longrightarrow \quad A \in \mathrm{GL}(n; \boldsymbol{R})\,.$$

Satz 5: Die JACOBI– und die GAUSS–SEIDEL–Iterationen für $Ax = b$ konvergieren, falls A streng diagonal dominant ist.

Hinweis: Für (6) war A s.d.d., für (7) nicht!

Eine unter Umständen wesentliche Verbesserung des Konvergenzverhaltens liefer die sogenannte

SOR–Methode:
(= Methode der sukzessiven Überrelaxation = successive overrelaxation)
Man zerlegt die Matrix A in 3 Summanden:

$$A = L + D + R$$

- $L \ldots$ strikt linke untere Δ–Matrix
- $R \ldots$ strikt rechte obere Δ–Matrix
- $D \ldots$ Diagonalmatrix

Die JACOBI–Iteration liefert in diesem Fall für die i–te Komponente von x

$$\begin{aligned} a_{ii}(x_i)_{k+1} &= (-a_{i1}x_1 - \ldots - a_{in}x_n)_k + b_i \\ &= -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}(x_j)_k + b_i \,, \quad i = 1, \ldots, n \end{aligned}$$

In Matrizenschreibweise:

$$Dx_{k+1} = -(L+R)x_k + b$$

Wegen der Voraussetzung $a_{ii} \neq 0\,, i = 1, \ldots, n\,,$ ist $D = \operatorname{diag}(a_{11}, \ldots, a_{nn}) \in \mathrm{GL}(n; \boldsymbol{R})\,.$

$$\Longrightarrow \quad x_{k+1} = \underbrace{-D^{-1}(L+R)}_{T_J} x_k + D^{-1}b$$

$$\boxed{x_{k+1} = T_J x_k + D^{-1}b}$$

JACOBI–Verfahren (Gesamtschrittverfahren)

$T_J \ldots$ JACOBI–Iterationsmatrix
Früher hatten wir $S = D, T = -L - R\,.$
Das GAUSS–SEIDEL–Verfahren lieferte

$$Dx_{k+1} = -Lx_{k+1} - Rx_k + b$$

$$\Longrightarrow \quad (D+L)x_{k+1} = -Rx_k + b$$

$D + L$ ist linke untere Δ–Matrix, deren Diagonalelemente $\neq 0$ sind.

$$\Longrightarrow \quad D + L \in \mathrm{GL}(n; \boldsymbol{R})$$

$$\Longrightarrow \quad x_{k+1} = \underbrace{-(D+L)^{-1}R}_{T_{\mathrm{GS}}} x_k + (D+L)^{-1}b$$

$$\boxed{x_{k+1} = T_{\mathrm{GS}} x_k + (D+L)^{-1} b}$$

GAUSS–SEIDEL–Verfahren (Einzelschrittverfahren)

$T_{\mathrm{GS}} \ldots$ GAUSS–SEIDEL–Iterationsmatrix

Die Korrektur im $(k+1)$–ten Schritt ist im GAUSS–SEIDEL–Verfahren

$$\begin{aligned}
(\Delta x_i)_{k+1} &:= (x_i)_{k+1} - (x_i)_k \\
&= -\frac{1}{a_{ii}} \left[\sum_{j=1}^{i-1} a_{ij}(x_j)_{k+1} + \sum_{j=i+1}^{n} a_{ij}(x_j)_k \right] + \frac{b_i}{a_{ii}} - (x_i)_k \\
&= -\frac{1}{a_{ii}} \left[\sum_{j=1}^{i-1} a_{ij}(x_j)_{k+1} + \sum_{j=i}^{n} a_{ij}(x_j)_k \right] + \frac{b_i}{a_{ii}}
\end{aligned}$$

Die Idee ist nun $(\Delta x_i)_{k+1}$ durch

$$\omega \cdot (\Delta x_i)_{k+1}\,,\ \omega \in \boldsymbol{R}\,,$$

zu ersetzen.
$\omega \ \ldots$ Relaxationsfaktor
$\omega > 1$ Überrelaxation

$$\begin{aligned}
(x_i)_{k+1} &= (x_i)_k + \omega(\Delta x_i)_{k+1} \\
&= (x_i)_k - \frac{\omega}{a_{ii}} \left[\sum_{j=1}^{i-1} a_{ij}(x_j)_{k+1} + \sum_{j=i}^{n} a_{ij}(x_j)_k - b_i \right]
\end{aligned}$$

Nach einer Multiplikation mit a_{ii} erhalten wir in Matrizenschreibweise

$$Dx_{k+1} = Dx_k - \omega L x_{k+1} - \omega(D+R)x_k + \omega b$$

oder

$$(D + \omega L)x_{k+1} = [(1-\omega)D - \omega R]\, x_k + \omega b$$

$\omega = 1$ liefert wieder GAUSS–SEIDEL–Iteration, d.h. keine Beschleunigung.

Beschleunigung erst mit $\omega > 1$!
Beispiel:

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix}}_{L} + \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix}}_{R}$$

$$D + \omega L = \begin{pmatrix} 2 & 0 \\ -\omega & 2 \end{pmatrix}$$

$$(1-\omega)D - \omega R = \begin{pmatrix} 2(1-\omega) & \omega \\ 0 & 2(1-\omega) \end{pmatrix}$$

Die Iteration lautet jetzt:

$$\begin{pmatrix} 2 & 0 \\ -\omega & 2 \end{pmatrix} x_{k+1} = \begin{pmatrix} 2(1-\omega) & \omega \\ 0 & 2(1-\omega) \end{pmatrix} x_k + \omega b$$

$$x_{k+1} = \underbrace{\begin{pmatrix} 1-\omega & \frac{\omega}{2} \\ \frac{\omega}{2}(1-\omega) & 1-\omega+\frac{\omega^2}{4} \end{pmatrix}}_{T_\omega} x_k + \frac{\omega}{4}\begin{pmatrix} 2 & 0 \\ \omega & 2 \end{pmatrix} b$$

Frage: Wie kann man ω optimal wählen?
Bedingung: $\rho(T_{\omega_{\text{opt}}}) \leq \rho(T_\omega)\,,\ 0 < \omega < 2\,.$
Die Berechnung von ω wird schwierig! Hier:

$$\omega_{\text{opt}} = 4(2-\sqrt{3}) \approx 1.07 \quad \Longrightarrow \quad \rho \approx 0.07 \quad \Longrightarrow \quad r = 1.15$$

Definition: Eine Matrix $A \in \mathrm{M}(n \times n; \boldsymbol{R})$ mit der speziellen tridiagonalen Gestalt

$$A = \begin{pmatrix} D_1 & H_1 & & & & \\ K_1 & D_2 & H_2 & & \mathcal{O} & \\ & K_2 & D_3 & & & \\ & & K_3 & \ddots & & \\ & \mathcal{O} & & \ddots & \ddots & H_{s-1} \\ & & & & K_s & D_s \end{pmatrix}$$

D_i ... quadratische Diagonalmatrizen
H_i, K_i ...i.a. rechteckig und beliebig
heißt T–Matrix.

Satz 6: Ist A eine T–Matrix mit $a_{ii} \neq 0$ und besitzt die JACOBI–Iterationsmatrix

$$T_J = D^{-1}(L+R)$$

nur reelle Eigenwerte λ_j und gilt

$$\rho(T_J) < 1\,,$$

so ist der optimale Relaxationsparameter ω_{opt} des SOR–Verfahrens gegeben durch

$$\omega_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho(T_J)^2}}$$

Beispiel: Differentialgleichung $-u'' = f(x)$ von oben mit $n = 21\,, h = \frac{1}{22}$:

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & \\ -1 & 2 & -1 & 0 & \cdots & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & -1 & 2 & -1 \\ & & & & 0 & -1 & 2 \end{pmatrix} \in \mathrm{M}(21 \times 21; \boldsymbol{R})$$

SOR: $\omega_{\text{opt}} \approx 1.75$.
In diesem Fall ist $\rho_{\text{SOR}} \approx 0.75$ im Vergleich zu
$\rho_{\text{J}} \approx 0.99$ und
$\rho_{\text{GS}} \approx 0.98$

Mit SOR wird bei jedem Schritt der Fehler um 25% verbessert.

Da $\rho_{\text{J}}^{30} \approx \rho_{\text{SOR}}$
$\Longrightarrow$ Ein einziger SOR–Schritt ist so viel wert wie 30 JACOBI–Schritte.