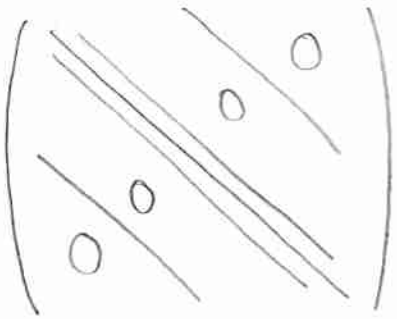
§39: ITERATIVE VERFAHREN FÜR LINEARE GLEICHUNGSSYSTEME

39.1. Motivation

- Viele praktische Probleme führen auf sehr große lineare Gleichungssysteme, bei denen die Systemmatrix dünn besetzt ist, d.h. nur wenige Einträge von 0 verschieden sind.

Beispiel:



Pentadiagonalmatrix bei 2-D Diffusionsfiltern in der Bildverarbeitung ◁

- Aus Speicherplatzgründen will man oft nur die von 0 verschiedenen Elemente abspeichern.

Beispiel: Ein Grauwertbild mit 512×512 Pixeln führt zu $512^2 = 262.144$ Gleichungen mit ebenso vielen Unbekannten. Bei 8 Byte / Eintrag und vollem Abspeichern benötigt die Pentadiagonalmatrix $8 \cdot 512^4 \approx 550$ GB, bei effizientem Abspeichern nur $5 \cdot 512^2 \approx 1{,}3$ MB!

- Direkte Verfahren wie der Gauß-Algorithmus können die Nullen auffüllen und so zu einem enormen Speicherplatzbedarf führen.
  Zudem ist ihr Rechenaufwand oft zu hoch.
  $O(n^3)$ Operationen für ein $(n \times n)$-Gleichungssystem.

- Daher verwendet man oft iterative Näherungsverfahren, die kaum zusätzlichen Speicherplatz benötigen und nach wenigen Schritten eine brauchbare Approximation liefern

39.2. Grundstruktur klassischer iterativer Verfahren

Geg.: $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$

Ges.: $x \in \mathbb{R}^n$ mit $Ax = b$

Falls $A = S - T$ mit einer einfach zu invertierenden Matrix S (z.B. Diagonalmatrix, Dreiecksmatrix), so kann man $Ax = b$ umformen in

$$Sx = Tx + b$$

und mit einem Startvektor $x^{(0)} \in \mathbb{R}^n$ die Fixpunktiteration

$$Sx^{(k+1)} = Tx^{(k)} + b \qquad (k = 0, 1, 2, \dots)$$

anwenden.

Wir wollen nun drei verschiedene Aufspaltungen $A = S - T$ untersuchen.

Sei hierzu $A = D - L - R$ eine Aufspaltung in eine Diagonalmatrix D, eine strikte untere Dreiecksmatrix L und eine strikte obere Dreiecksmatrix R.

$$\underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix}}_{D} - \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ * & 0 & 0 \\ * & * & 0 \end{bmatrix}}_{L} - \underbrace{\begin{bmatrix} 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{bmatrix}}_{R}$$

39.3. Das Jacobi-Verfahren (Gesamtschrittverfahren)

Hier wählt man $S := D$ und $T := L+R$.

In jeder Iteration wird also nur das Diagonalsystem

$$D\, x^{(k+1)} = (L+R)\, x^{(k)} + b$$

nach $x^{(k+1)}$ aufgelöst, d.h. nur durch die Diagonalelemente dividiert:

$$x^{(k+1)} = D^{-1}\left((L+R)\, x^{(k)} + b \right)$$

explizit:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(- \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}\, x_j^{(k)} + b_i \right) \qquad (i = 1, \ldots, n)$$

39.4. Beispiel

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, \quad b = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

führt auf das Diagonalsystem

$$2\, x_1^{(k+1)} = x_2^{(k)} + 3$$

$$2\, x_2^{(k+1)} = x_1^{(k)} + 4$$

Bei vierstelliger Genauigkeit und Startvektor $x^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ erhält man

k	$x_1^{(k)}$	$x_2^{(k)}$
0	0	0
1	1,5	2
2	2,5	2,75
3	2,875	3,25
4	3,125	3,438
5	3,219	3,568
6	3,282	3,610
7	3,305	3,641
8	3,321	3,653
9	3,327	3,661
10	3,331	3,664
11	3,332	3,666
12	3,333	3,666

exakte Lösung: $x_1 = 3\frac{1}{3}$, $x_2 = 3\frac{2}{3}$.

Das Jacobi-Verfahren ist gut geeignet für Parallelrechner, da $x_i^{(k+1)}$ nicht von $x_j^{(k+1)}$ abhängt.

39.5. Das Gauß-Seidel-Verfahren (Einzelschrittverfahren)

$$S := D - L, \quad T := R$$

In jeder Iteration wird das Dreiecksystem

$$(D-L)\, x^{(k+1)} = R\, x^{(k)} + b$$

durch einfache Vorwärtssubstitution gelöst:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(- \sum_{j=1}^{i-1} a_{ij}\, x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}\, x_j^{(k)} + b_i \right) \qquad (i = 1, \dots, n)$$

d.h. neue Werte werden weiterverwendet, sobald sie berechnet wurden.

39.6. Beispiel

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 4 \end{pmatrix}, \quad x^0 := \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$x_1^{(k+1)} = \frac{1}{2} \left(x_2^{(k)} + 3 \right)$$

$$x_2^{(k+1)} = \frac{1}{2} \left(x_1^{(k+1)} + 4 \right)$$

k	$x_1^{(k)}$	$x_2^{(k)}$
0	0	0
1	1,5	2,75
2	2,875	3,348
3	3,174	3,587
4	3,294	3,647
5	3,324	3,662
6	3,331	3,666

konvergiert etwa doppelt so schnell wie Jacobi-Verfahren.

39.7. Das SOR-Verfahren (SOR = successive overrelaxation)

Beschleunigung des Gauß-Seidel-Verfahrens durch Extrapolation:

$$x^{(k+1)} = x^{(k)} + \omega \left(\tilde{x}^{(k+1)} - x^{(k)} \right)$$

wobei $\omega \in (1,2)$ und $\tilde{x}^{(k+1)}$ die Gauß-Seidel-Iteration zu $x^{(k)}$ ist.

Für große Gleichungssysteme kann damit die Iterationszahl für ein geeignetes ω oft um eine Zehnerpotenz gesenkt werden.

39.8. Konvergenzresultate

a) Die Jacobi-, Gauß-Seidel- und SOR-Verfahren können als Fixpunktiterationen interpretiert werden.

Nach 26.6 liegt Konvergenz vor, wenn die Abbildung kontrahierend ist. Dies ist nicht in allen Fällen erfüllt.

b) Für wichtige Spezialfälle existieren jedoch Konvergenzaussagen; z.B.:

Ist die Systemmatrix $A = (a_{ij}) \in \mathbb{R}^{n \times n}$ streng diagonaldominant (d.h. $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \forall i$), so konvergieren das Jacobi- und das Gauß-Seidel-Verfahren.

39.9 Effizienz

a) Die vorgestellten Verfahren sind die einfachsten, allerdings nicht die effizientesten iterativen Verfahren zum Auflösen linearer Gleichungssysteme.

b) Effizientere, aber kompliziertere Verfahren:

- ADI-Methoden (ADI = alternating direction implicit)
- PCG-Verfahren (PCG = preconditioned conjugate gradients)
- Mehrgitterverfahren (multigrid)

Siehe numerische Spezialliteratur.

c) Hocheffiziente Ansätze wie die Mehrgitterverfahren verwenden oft das Jacobi-Seidel-Verfahren als Grundbaustein.

Mit ihnen ist es z.T. möglich, lineare Gleichungssysteme in optimaler Komplexität (d.h. $O(n)$) zu lösen.