PRAKTISCHE UMSETZUNG DES SOR-VERFAHRENS

Praktische Aufgabe P1 im Modul Math-Ba-NUM

Lars Ortscheidt & Eric Kunze

1. Darstellung der Besetzung

Die Darstellung der Besetzung der jeweiligen Matrizen ist Abb. 1 zu entnehmen.

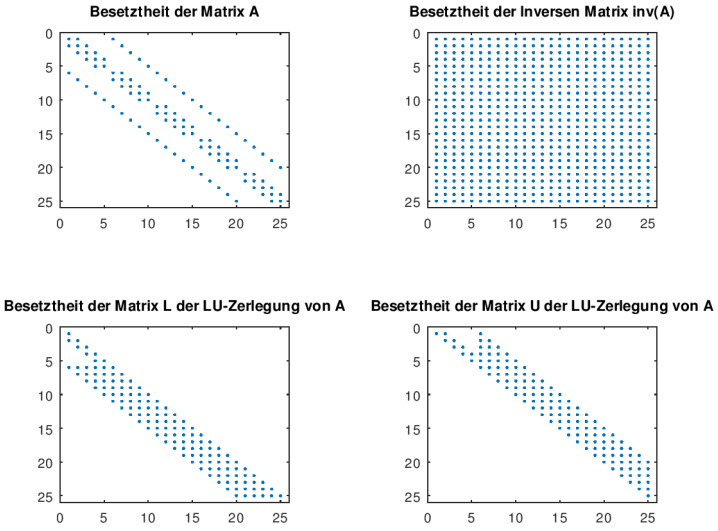

Abbildung 1: Besetzung der einzelnen Matrizen

Anhand der Darstellung ist erkennbar, dass die Eigenschaft der Dünnbesetztheit nicht invariant unter Inversion ist, d.h. die Inverse einer dünnbesetzten Matrix kann wie hier durchaus (nahezu) voll besetzt sein. Damit beeinflussen Inversionen auch die Geschwindigkeit eines Iterationsverfahrens negativ.

Die LU-Zerlegung der Matrix A existiert stets, da alle Matrizen A unabhängig von ihrer Dimension immer regulär, symmetrisch und positiv definit sind.

2. Iterationsverfahren

Wir beschreiben hier kurz unsere Lösungsstrategie im SOR-Verfahren, welches im File `SOR.m` zu finden ist. Zu lösen sei das Gleichungssystem $Ax = b$.

Die erste Idee war die Implementierung mittels der Iterationsvorschrift $Mx = c$, wobei

$$M := \left(L + \frac{1}{\omega}D\right)^{-1}\left(\frac{1-\omega}{\omega}D - R\right) \quad \text{und} \quad c := \left(L + \frac{1}{\omega}D\right)^{-1} \cdot b$$

Jedoch ist insbesondere die notwendige Inversion stets negativ auffallend, da aus den dünn besetzten Matrizen dann wieder (nahezu) vollbesetzte Matrizen entstehen.

Aus diesem Grund verwenden wir im Folgenden die Darstellung ohne Inversion und damit die Iterationsvorschrift

$$x^{k+1} := x^k - \left(L + \frac{1}{\omega}D\right)^{-1}(Ax^k - b)$$

Dabei kann die Inversion in der konkreten Implementierung durch den Operator \ ersetzt werden, sodass also die eigentliche Iteration mit $B = L + \frac{1}{\omega} \cdot D$ sich durch

```
1 while(n<=nmax && fehler>=tol)
2         X       = X - B \ (A*X-b); % Iterationsschritt
3         n       = n + 1;           % Iterationszaehler
4         fehler = norm(A * X - b) / norm(b);
5 endwhile
```

3. Poisson-Gleichung

Wir betrachten die Lösung $u(x, y) = x^2(1-x)y(1-y)$ der Poisson-Gleichung $-\Delta u(x, y) = g(x, y)$. Dann ist die Funktion g durch

$$g(x, y) = -\operatorname{div}\operatorname{grad}\left(x^2(1-x)y(1-y)\right) = 2\left(-x^2 + x^3 - (-1+y)y + 3x(-1+y)y\right)$$

bestimmt. Dies ist im Programm die Funktion `poisson`.

4. Dokumentation der Iterationszahlen

Sei

$$f(x, y) = \sin(\pi \cdot x) \cdot \sin(\pi \cdot y)$$

und

$$g(x, y) = 2\left(-x^2 + x^3 - (-1+y)y + 3x(-1+y)y\right)$$

Die Werte für den Relaxationsparameter $\omega \in \{1, 1.8, \omega^*\}$ beziehen sich jeweils auf die Berechnung mit `SOR`, für das CG-Verfahren wurde die Funktion `pcg` verwendet.

N \ ω	1	1.8	ω^*	CG
5	49	66	17	1
10	168	69	32	1
100	1500	1500	299	1
200	1500	1500	595	1

Tabelle 1: Iterationszahlen für f

N \ ω	1	1.8	ω^*	CG
5	49	66	17	11
10	165	69	33	22
100	1500	1500	300	221
200	1500	1500	598	447

Tabelle 2: Iterationszahlen für g

Aus diesen Werten sind mehrere Eigenschaften ersichtlich. Offensichtlich konvergiert das CG-Verfahren schneller als das SOR-Verfahren, was aber auch so zu erwarten war. Insbesondere bei der Betrachtung der Funktion g lässt sich aber erahnen, dass bei noch größeren $N > 200$ sich die Iterationszahlen des CG-Verfahrens und des SOR-Verfahrens mit ω^* weiter annähern.

Betrachten wir das SOR-Verfahren isoliert, so nimmt die Iterationszahl für die Folge der ω i.d.R. kontinuierlich ab. $\omega = 1$ beschreibt ja bekanntermaßen das Gauß-Seidel-Verfahren. Dieses kommt ohne den Relaxationsparameter aus und erreicht damit schlechtere Konvergenzeigenschaften. Das ω^* beschreibt den optimalen Relaxationsparameter, was man auch zeigen kann. Allgemein kann folgende Darstellung gefunden werden

$$\omega^* = \frac{2}{1 + \sqrt{1 - \rho(D^{-1}(L + R))}}$$

Damit wird dann die schnellste Konvergenz erreicht, was beispielsweise auch dazu führt, dass die maximale Iterationszahl nur mit ω^* unterschritten wird (betrachte z.B. f und $N = 200$).

Vergleicht man nun beide Funktionen f und g, so stellt man fest, dass f im Allgemeinen relativ ähnliche Iterationszahlen im SOR-Verfahren liefert wie g. Ein deutlicher Unterschied ist dagegen bei der Verwendung des CG-Verfahrens zu erkennen, was für f stets nach einer Iteration konvergiert, dagegen für g mitunter fast so viele Iterationen benötigt wie das SOR-Verfahren mit ω^*.

Besonders ist anhand dieser Aufgabe zu erkennen, dass nicht alle Verfahren gleich schnell konvergieren und auch für verschiedene Parameter innerhalb der Verfahren sich unterschiedliche Konvergenzeigenschaften ergeben.

5. Fehlerdarstellung

Im Folgenden ist die grafische Darstellung der Fehler in Abhängigkeit der Iterationszahl zu finden. Der (relative) Fehler wurde mithilfe von

$$\frac{\|A \cdot x - b\|}{\|b\|}$$

berechnet. Dabei haben wir eine dem SOR-Verfahren gleiche Funktion `fehlervec` erstellt, die das SOR-Verfahren simuliert, aber lediglich ein Vektor zurückgibt, dessen i-ter Eintrag der Fehler im i-ten Iterationsschritt ist.

Es ist deutlich zu erkennen, dass die Konvergenz mit optimiertem ω^* schneller erreicht wird als im Gauß-Seidel-Verfahren oder mit $\omega = 1.8$. Zwischen den beiden Gleichungen f und g sind wie auch in den Iterationszahlen nur geringfügige Unterschiede zu erkennen.

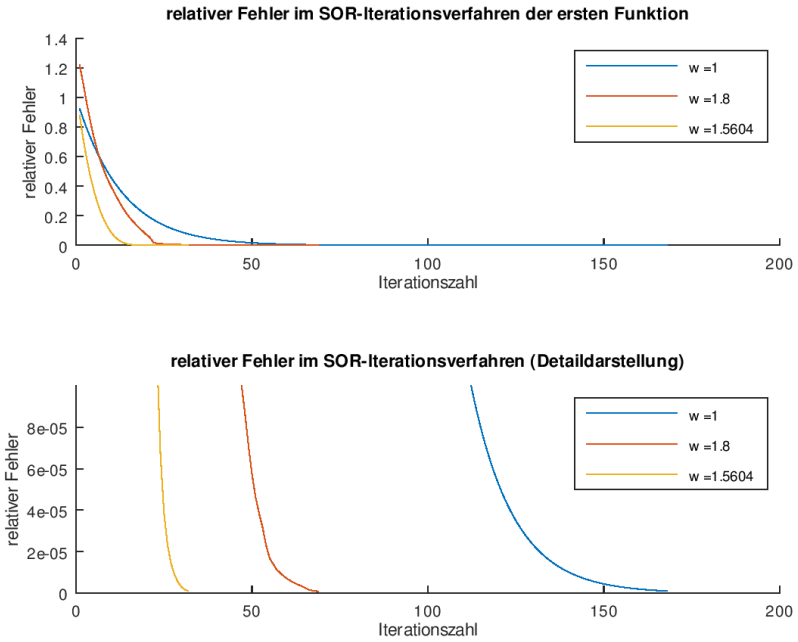

Abbildung 2: Fehler in Abhängigkeit der Iteration von f

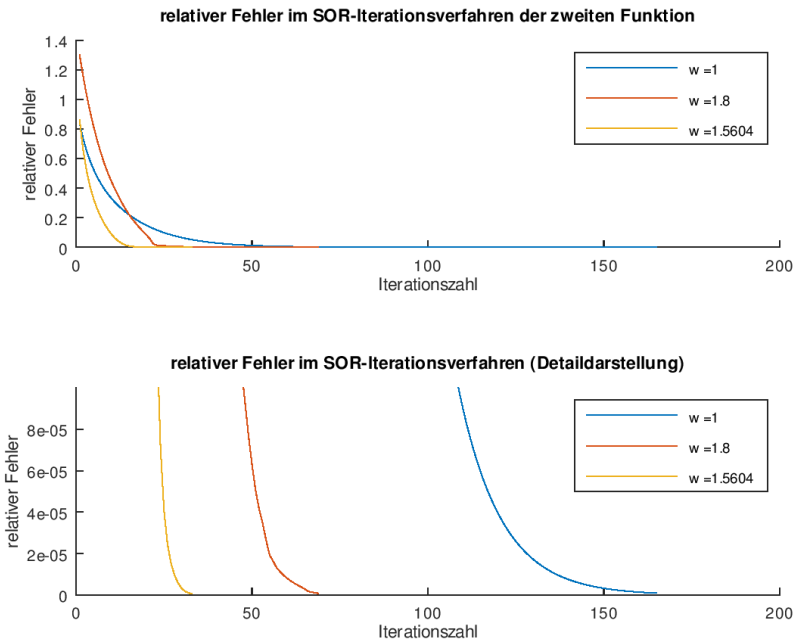

Abbildung 3: Fehler in Abhängigkeit der Iteration von g