Institut für Numerische Simulation der Universität Bonn
Prof. Dr. Mario Bebendorf

Praktikum im Sommersemester 2012

Programmierpraktikum numerische Algorithmen (P2E1)
(Numerische Lösung der Wärmeleitungsgleichung)

Betreuer: Christian Kuske[1]

Blatt 4

1 Einführung

Auf den ersten drei Übungsblättern wurden zahlreiche Methoden vorgestellt, lineare Gleichungssysteme zu lösen. Jedes dieser Verfahren besitzt Vor- und Nachteile. Einige der Nachteile konnten durch Verbesserung der Algorithmen behoben werden. Allerdings bleibt allen Verfahren gemein, dass die Konvergenzgeschwindigkeit von der Kondition der Systemmatrix abhängt. Dieses Problem tritt bei der Mehrgitter-Methode nicht auf.

Auf diesem Blatt werden die grundlegenden Eigenschaften der Mehrgitter-Methode beschrieben. Da sich die Ideen des Verfahrens bereits anhand zweier Gitter erklären lässt, beschränken wir uns zunächst auf die Zweigitter-Methode.

2 Zweigitter-Methode

Die Idee der Zweigitter-Methode besteht darin, zwei Gitter mit unterschiedlicher Maschenweite zu verwenden, wobei Lösungen auf dem groben Gitter Ω_H mit der Maschenweite $H = 2h$ genutzt werden um Lösungen auf dem feinen Gitter Ω_h mit der Maschenweite h zu ermitteln. Ein Vorteil ergibt sich durch die verringerte Maschenweite auf dem groben Gitter, wodurch Berechnungen mit weniger Aufwand durchgeführt werden können.

Im folgenden Algorithmus sind I_H^h und I_h^H Prolongations- beziehungsweise Restriktions-Operatoren. Die mit H indizierten Variablen stellen jeweils Operationen auf dem groben Gitter dar, wohingegen die mit h indizierten auf dem feinen Gitter arbeiten. Für die Anzahl der Glättungen am Anfang und Ende des Verfahrens genügen ein bis drei Iterationen.

Glätte $L_h u_h = f_h$ auf Ω_h
Berechne das Residuum $r_h = f_h - L_h u_h$
Bestimme $r_H = I_h^H r_h$
Löse $L_H e_H = r_H$ auf Ω_H
Bestimme $e^h = I_H^h e^H$
Setze $u_h = u_h + e_h$
Glätte $L_h u_h = f_h$ auf Ω_h

Algorithmus 2.1: Zweigitter-Methode

[1] We6 4.002; Tel.: 0228/73-60454; Email: kuske@ins.uni-bonn.de

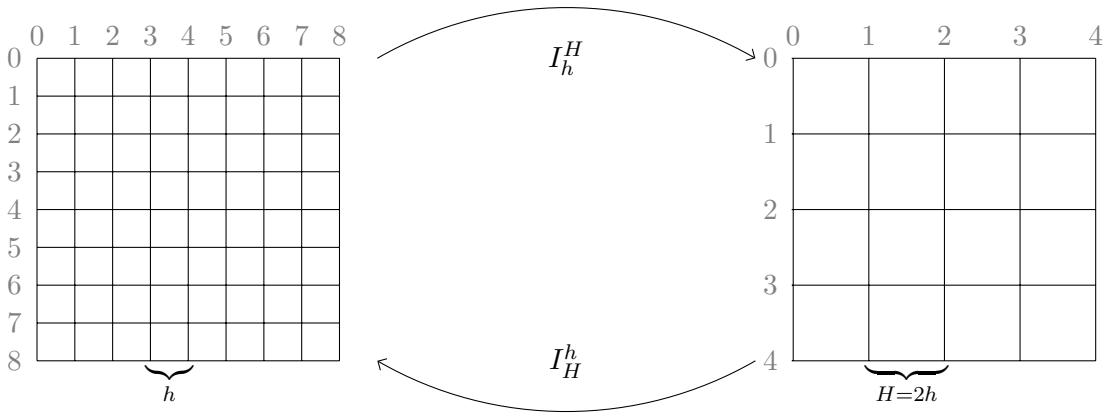

Abbildung 1: Gitterwechsel mit Hilfe des Prolongations-Operators I_H^h und des Restriktions-Operators I_h^H zwischen einem 9×9 beziehungsweise 5×5 Gitter.

Im Algorithmus wird zunächst der hochfrequente Fehleranteil auf einem feinen Gitter geglättet. Das dadurch entstehende Residuum wird mittels eines Restriktions-Operators auf ein grobes Gitter übertragen und exakt gelöst. Anschließend wird die Lösung auf dem groben Gitter durch einen Prolongations-Operator auf das feine Gitter zurückgeführt und dient dort als Korrektur für die Lösung auf dem feinen Gitter.

3 Gitterwechsel

Das Wechseln zwischen feinen und groben Gitter ist für das Zweigitter-Verfahren wichtig. Für die Prolongation sowie Restriktion stehen dabei mehrere Möglichkeiten zur Verfügung. Für eine schematische Darstellung des Gitterwechsels sei Abbildung 1 zu betrachten.

Prolongation

Der Prolongations-Operator führt durch Interpolation das grobe Gitter in ein feines über. Hierbei werden die Punkte des feines Gitters, die nicht auf dem groben liegen, durch lineare Interpolation, das heißt durch Gewichtung der Nachbarpunkte, ermittelt. Es ist

$$\begin{aligned} v_h^{(2i,2j)} &= v_H^{(i,j)}, \\ v_h^{(2i+1,2j)} &= \frac{1}{2}(v_H^{(i,j)} + v_H^{(i+1,j)}), \\ v_h^{(2i,2j+1)} &= \frac{1}{2}(v_H^{(i,j)} + v_H^{(i,j+1)}), \\ v_h^{(2i+1,2j+1)} &= \frac{1}{4}(v_H^{(i,j)} + v_H^{(i+1,j)} + v_H^{(i,j+1)} + v_H^{(i+1,j+1)}), \end{aligned} \tag{1}$$

mit $0 \leq i, j \leq \frac{n-1}{2}$.

Restriktion

Im Gegensatz zur Prolongation wird die Restriktion dazu verwendet, ein feines Gitter auf ein grobes Gitter zu übertragen. Dabei bietet sich neben der Gewichtung durch Nachbarpunkte ebenfalls die Injektion an, das bedeutet

$$v_H^{(i,j)} = v_h^{(2i,2j)}.$$

Die gewichtete Restriktion ergibt sich zu:

$$\begin{aligned} v_H^{(i,j)} \quad = \quad & \frac{1}{16}(v_h^{(2i-1,2j-1)} + v_h^{(2i-1,2j+1)} + v_h^{(2i+1,2j-1)} + v_h^{(2i+1,2j+1)} \\ & + 2(v_h^{(2i,2j-1)} + v_h^{(2i,2j+1)} + v_h^{(2i-1,2j)} + v_h^{(2i+1,2j)}) \\ & + 4v_h^{(2i,2j)}), \end{aligned} \tag{2}$$

mit $1 \leq i, j \leq \frac{n-1}{2} - 1$.

Da sich allerdings zeigen lässt, dass für diesen Restriktions-Operator der Prolongations-Operator von oben – bis auf eine Konstante – transponiert ist, wird dieser für das Zweigitter-Verfahren herangezogen.

4 Glätter und Löser

Für die Zweigitter-Methode werden die, auf den vorherigen Aufgabenblättern erarbeiteten, Verfahren benutzt, die den hochfrequenten Fehler glätten und die Lösung eines Gleichungssystem schnell bestimmen können.

Glätter

Hierfür eignen sich das Jacobi- und das Gauss-Seidel-Verfahren. Diese Verfahren konvergieren langsam, da der niederfrequente Anteil des Fehlers nur langsam reduziert wird. Allerdings kann durch wenige Iterationsschritte der hochfrequente Anteil geglättet werden.

Beim SOR-Verfahren konnte durch einen Relaxationsparameter die Konvergenz beschleunigt werden, indem der Parameter optimal gewählt wurde. Bei schlechter Wahl wurde das SOR-Verfahren langsamer als das unmodifizierte Verfahren. Durch eine leichte Unterrelaxation kann ein Oszillieren des Fehlers gedämpft und so eine weitere Glättung erzielt werden.

Löser

Es sei zunächst angemerkt, dass auf einem groben Gitter der Lösungsvektor lediglich ein Viertel der Einträge des ursprünglichen Vektors besitzt, wodurch die Lösung schneller bestimmt werden kann.

Für das Zweigitter-Verfahren bietet sich das Lösen durch das CG-Verfahren an. Auch die Verwendung des SOR-Verfahrens kann, aufgrund der kleineren Problemgröße, herangezogen werden.

Für das Mehrgitter-Verfahren kann auf ein CG- oder SOR-Verfahren verzichtet werden, da bei sehr groben Gittern ein direktes Lösen effizienter und schneller ist.

5 Aufgaben

a) Implementiere das SOR-Verfahren für das lineare Gleichungssystem $Ax = b$ und verwende dazu:

```
- void SOR(const unsigned int n, double * const x,
           const double * const b, const double omega,
           const unsigned int maxIt),
```

wobei die Matrix A die symmetrische Laplace-Matrix L_h ist. Das Verfahren soll nach `maxIt` Schritten abbrechen.

b) Implementiere Prolongations- und Restriktionsoperator und verwende die beiden folgenden Funktionen:

- `void prolong(const unsigned int n, const double * const xH, double * const xh)`,
  wobei $x_h = I_H^h x_H$ berechnet wird und I_H^h der in Gleichung (1) definierte Prolongations-Operator ist.
- `void restrict(const unsigned int n, const double * const xh, double * const xH)`,
  wobei $x_H = I_h^H x_h$ berechnet wird und I_h^H der in Gleichung (2) definierte Restriktions-Operator ist. (Hinweis: Die Restriktion ist lediglich auf dem Inneren des Gebiets definiert, am Rand verwendet man entweder die Injektion oder eine passende Gewichtung.)

c) Implementiere die Zweigitter-Methode zur Lösung von $L_h u_h = f_h$ mit Hilfe der Funktion:

- `unsigned int TGM(const unsigned int n, double * const u, const double * const f, const double eps)`,
  wobei der Rückgabewert, der Anzahl der benötigten Iterationen entspricht. Setze $n = 2^L + 1$ und beende das Verfahren, falls $\|r_h\|_2 < \varepsilon$ gilt. Verwende als Glätter das SOR-Verfahren mit `maxIt` $= 5$ und als Löser das CG-Verfahren mit einer Genauigkeit von 10^{-14}.

d) Erstelle eine Tabelle für wachsendes n, bei dem die Anzahl der Iterationsschritte und die benötigte Zeit gemessen werden!

e) Verwende das PCG-Verfahren als Löser in der Zweigitter-Methode und stelle die Ergebnisse ebenfalls tabellarisch dar!