Mathematisches Programmieren

Sommersemester 2022

25.05.2022, 01.06.2022, 08.06.2022 und 15.06.2022

Die Poisson-Gleichung

Sei $\Omega \subset \mathbb{R}^n$ offen, beschränkt und einfach zusammenhängend. Wir betrachten die *Poisson-Gleichung*

$$-\Delta u = f \quad \text{in } \Omega \tag{1}$$

mit homogenen *Dirichlet-Randbedingungen*

$$u = 0 \quad \text{auf } \partial\Omega\,. \tag{2}$$

Hierbei bezeichnet Δ den *Laplace-Operator*

$$\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2}\,. \tag{3}$$

Diese Gleichung beschreibt z.B. die Abhängigkeit des elektrischen Potentials u in Abhängigkeit von einer gegebenen Ladungsverteilung f. Die homogene Dirichlet-Randbedingung drückt aus, dass das Gebiet von einem elektrischen Leiter umschlossen ist, dessen Potential auf Null liegt. Die Poisson-Gleichung taucht aber auch in vielfältigen anderen Zusammenhängen in Mathematik und Physik auf. Sie ist der Prototyp einer *elliptischen partiellen Differentialgleichung.*

Finite Differenzen im Eindimensionalen ($n = 1$)

Hier ist das Gebiet Ω einfach ein Intervall. Mit entsprechender Skalierung können wir $\Omega = (0, \pi)$ wählen. Dann lässt sich das homogene Dirichletproblem schreiben als

$$-u''(x) = f(x) \quad \text{in } (0, \pi)\,, \tag{4a}$$

$$u(0) = u(\pi) = 0\,. \tag{4b}$$

Um diese Gleichung zu diskretisieren, unterteilen wir das Intervall $(0, L)$ in N Teilintervalle der Länge $h = L/N$, wobei hier $L = \pi$, und definieren die Stützstellen

$$x_i = i\,h\,. \tag{5}$$

Sei nun $u_i \approx u(x_i)$, also eine Näherung zur Lösung $u(x)$ an der Stützstelle x_i, die folgendes lineares Gleichungssystem erfüllt:

$$-\frac{u_{i-1} - 2\,u_i + u_{i+1}}{h^2} = f(x_i) \quad \text{für } i = 1, \ldots, N-1\,, \tag{6a}$$

$$u_0 = u_N = 0\,. \tag{6b}$$

Man kann zeigen, dass, falls die exakte Lösung u viermal stetig differenzierbar ist, die linke Seite dieser Näherung gegen $-u''(x)$ konvergiert, und zwar quadratisch, im Grenzfall $h \to 0$.

Finite Differenzen für $n = 2$

In zwei Raumdimensionen kann die Näherung (6) für jede der partiellen Ableitungen verwendet werden. Diese Näherung funktioniert am Besten, wenn die Ränder des Gebiets achsenparallele Geradensegmente sind. Wir beschränken uns daher auf den Fall eines Rechtecks $\Omega = (0, L_x) \times (0, L_y)$. In jeder Raumrichtung nehmen wir N_x bzw. N_y Teilintervalle, so dass die jeweiligen Schrittweiten $h_x = L_x/N_x$ bzw. $h_y = L_y/N_y$ sind. Die Stützstellen sind dann

$$x_{ij} = (i\,h_x, j\,h_y) \tag{7}$$

und die Näherung $u_{ij} \approx u(x_{ij})$ löst

$$-\frac{u_{i-1,j} - 2\,u_{ij} + u_{i+1,j}}{h_x^2} - \frac{u_{i,j-1} - 2\,u_{ij} + u_{i,j+1}}{h_y^2} = f(x_{ij}) \tag{8a}$$

für $i = 1, \ldots, N_x - 1$, $j = 1, \ldots, N_y - 1$, mit Randbedingungen

$$u_{i,0} = u_{i,N_y} = 0 \quad \text{für } i = 0, \ldots, N_x\,, \tag{8b}$$

$$u_{0,j} = u_{N_x,j} = 0 \quad \text{für } j = 0, \ldots, N_y\,. \tag{8c}$$

Aufgaben

1. Schreibe das lineare Gleichungssystem (6) in Matrixform. Erzeuge, für gegebenes N, diese Matrix in Python (Hinweis: verwende `diag`), berechne ihre Eigenwerte (Hinweis: `eig` oder `eigvals` – weil die Matrix symmetrisch ist, kann man auch `eigh` oder `eigvalsh` verwenden) und trage die Eigenwerte als Funktion ihres Index in ein Koordinatensystem zusammen mit den ersten $N - 1$ exakten Eigenwerten des Laplaceoperators, k^2 für $k = 1, \ldots, N - 1$, auf.

2. Das lineare Gleichungssystem (6) hat eine Tridiagonalstruktur. Weil das Gauß'sche Eliminationsverfahren diese Tridiagonalstruktur erhält, kann man das Gleichungssystem leicht in $O(N)$ Recheneoperationen lösen. (Das Gauß'sche Verfahren benötigt $O(N^3)$ Rechenoperationen im allgemeinen Fall und ist damit für sehr große Matrizen nicht geeignet.) Dieses vereinfachte Verfahren wird oft als „Thomas-Algorithmus“ bezeichnet.

Implementiere den Thomas-Algorithmus in Python (Hinweis: Folge der Notation im Wikipedia-Artikel über den Thomas-Algorithmus). Überprüfe die Richtigkeit des Lösers im Vergleich mit dem Scipy-Löser `scipy.linalg.solve_banded` und vergleiche die benötigte Rechenzeit für eine sehr große Tridiagonalmatrix.

3. Verwende einen der Löser aus Aufgabe 2, um die Poisson-Gleichung (4) zu lösen. Teste den Löser in folgenden Schritten:

   (a) Wähle eine Funktion $u(x)$ mit $u(0) = u(\pi) = 0$, die sich explizit differenzieren lässt und setze diese in die Poisson-Gleichung (4) ein, um $f(x)$ zu berechnen.

   (b) Löse das diskrete System (6) zu dem jetzt gegebenen $f(x)$ für verschiedene Werte von N. Trage die exakte Lösung $u(x)$ sowie die berechneten Näherungslösungen u_i in ein Koordinatensystem auf.

   (c) Berechne den Diskretisierungsfehler

   $$E(N) = \max_{i=1,\ldots,N-1} |u_i - u(x_i)|$$

   und trage $E(N)$ gegen N in ein doppelt-logarithmisches Koordinatensystem auf. Was stellen Sie fest?

4. Bei der Poisson-Gleichung in zwei Raumdimensionen bekommt man eine Matrix mit 4 Nebendiagonalen. Allerdings sind zwei dieser Nebendiagonalen $O(N)$ weit von der Hauptdiagonalen entfernt. Damit funktioniert der Thomas-Algorithmus nicht mehr: Ein normaler Gauß'sches Eliminationsprozess würde im Allgemeinen alle Einträge, die zwischen diesen Nebendiagonalen liegen, mit von Null verschiedenen Werten füllen. Glücklicherweise gibt es in Scipy effiziente Löser für lineare Gleichungssysteme mit dünnbesetzten Matrizen.

   Im Folgenden ist die Poisson-Gleichung auf dem Rechteck $\Omega = (0, 1) \times (0, 1)$ zu lösen. Der Einfachheit halber kann $N_x = N_y \equiv N$ gewählt werden.

   (a) Erzeuge die Matrix auf der linken Seite der Gleichung (8) im CSR-Format (Hinweis: `scipy.sparse.csr_matrix`).

   (b) Definiere ein geeignetes Testproblem, wie im Eindimensionalen, und löse dieses numerisch (Hinweis: `scipy.sparse.linalg.spsolve`).

   (c) Wie skaliert die Rechenzeit von `spsolve` mit der Größe des Gleichungssystems, das $(N_x - 1) * (N_y - 1)$ Variablen enthält?

Geben Sie ihren lauffähigen Code per Email in einem ZIP-Archiv bis zum 24.06.2022 ab.