Computerphysik WS 2013/2014

Prof. Dr. Roland Netz, FU Berlin

Übungsblatt 4: Gauß-Seidel / Nullstellenberechnung

Suliman Adam, Klaus Rinne
6. November 2013

Allgemeine Hinweise

Abgabetermin für die Lösungen ist

- **Sonntag, 17.11., 24:00 Uhr.**

Aufgabe 4.1: Iterative Methoden: Das Gauß-Seidel-Verfahren (10 Punkte)

Das Gauß-Seidel-Verfahren kann dazu verwendet werden, ein lineares Gleichungssystem der Form $\mathbf{A}\vec{x} = \vec{b}$ zu lösen. Die Matrix $\mathbf{A}$ wird in eine Diagonalmatrix $\mathbf{D}$ sowie eine untere und eine obere Dreiecksmatrix ($\mathbf{L}$ und $\mathbf{U}$) zerlegt, sodass: $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$.
Ein Startvektor $\vec{x}^{(0)}$ wird gewählt und iterativ durch Lösen der folgenden Gleichung verbessert:

$$(\mathbf{D} + \mathbf{L})\vec{x}^{(k+1)} = -\mathbf{U}\,\vec{x}^{(k)} + \vec{b}, \tag{1}$$

wobei $\vec{x}^{(k+1)}$ durch Vorwärtseinsetzen bestimmt wird. Als Startvektor können Sie im folgenden den Nullvektor verwenden.

- 4.1.1 (3 Punkte): Lösen Sie die folgende Gleichung mithilfe des Gauß-Seidel-Verfahrens:

$$\begin{bmatrix} 3.5 & 3 & -0.5 \\ -1 & 4 & 4 \\ \frac{1}{3} & 3 & 4.5 \end{bmatrix} \vec{x} = \begin{bmatrix} 7.5 \\ -6.5 \\ 1 \end{bmatrix} \tag{2}$$

  Nutzen Sie als Konvergenzkriterium $\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\|_2 < 0.5 \cdot 10^{-4}$ und geben Sie Ihr Ergebnis für $\vec{x}$ auf vier Dezimalstellen gerundet aus. Wieviele Iterationen sind notwendig?

- 4.1.2 (2 Punkte): Das Jacobi-Verfahren ähnelt dem Gauß-Seidel-Verfahren. Allerdings wird der Testvektor $\vec{x}^{(k)}$ erst nach Abschluss der Iteration aktualisiert. Als Iterationsvorschrift ergibt sich:

$$\mathbf{D}\vec{x}^{(k+1)} = (-\mathbf{L} - \mathbf{U})\,\vec{x}^{(k)} + \vec{b} \tag{3}$$

  Lösen Sie Gleichung 2 mithilfe des Jacobi-Verfahrens. Vergleichen Sie die beiden Ergebnisse miteinander. Inwieweit hat sich die Anzahl der Iterationen verändert?

- 4.1.3 (1 Punkt): Für welche Arten von Matrizen ist die Konvergenz des Gauß-Seidel-Verfahrens garantiert?

- 4.1.4 (2 Punkte): Erweitern Sie Ihren Code für das Gauß-Seidel- sowie das Jacobi-Verfahren (soweit noch nicht geschehen), sodass er allgemein zum Lösen von Gleichungen mit einer $n \times n$-Matrix verwendet werden kann. Ergänzen Sie Ihr Programm um ein Iterationslimit von 500 Iterationen. Testen Sie Ihr erweitertes Programm durch Lösen der folgenden Gleichung:

$$\begin{bmatrix} 5 & 3 & -1 & 2 \\ -3 & 7 & 6 & -2 \\ 4 & 4 & 3 & -3 \\ -5 & 2 & 2 & 4 \end{bmatrix} \vec{x} = \begin{bmatrix} 8 \\ 1 \\ 7 \\ 2 \end{bmatrix} \tag{4}$$

  Lässt sich die Gleichung mit beiden Verfahren lösen?

- 4.1.5 (2 Punkte): Durch die Integration eines Relaxationsfaktors $\omega \in (0, 2)$ ist es möglich die Konvergenz des Gauß-Seidel-Verfahrens zu beschleunigen:

$$\left(\mathbf{L} + \frac{1}{\omega}\mathbf{D}\right) \vec{x}^{(k+1)} = \left(\frac{1}{\omega}\mathbf{D} - \mathbf{D} - \mathbf{U}\right) \vec{x}^{(k)} + \vec{b} \tag{5}$$

  Dieser Algorithmus wird als das Successive Over-Relaxation-Verfahren (Überrelaxationsverfahren) bezeichnet. Lösen Sie Gleichung 4 für verschiedene Werte von ω und plotten Sie die Anzahl der benötigten Iterationsschritte als Funktion von ω. Für welche Werte von ω konvergiert das Verfahren in diesem Fall?

Aufgabe 4.2: Potentiallandschaft (10 Punkte)

Gegeben ist auf dem Gebiet $U = (-1, 1) \times (-1, 1)$ das Potential

$$\phi(x, y) = x^4 - x^2 + y^4 - 0.2y^3 - y^2 + 0.2xy^3$$

- 4.2.1 (1 Punkt): Plotten Sie $\phi(x, y)$ auf dem Gebiet U in einem 3D Plot.

  Sie können dazu das Modul **mpl_toolkits.mplot3d.axes3d** verwenden:

```
from mpl_toolkits.mplot3d import Axes3D
x=np.arange(-1.0,1.01,0.01)
y=np.arange(-1.0,1.01,0.01)
x, y = np.meshgrid(x,y)
Axes3D(plt.figure()).plot_wireframe(x,y,np.vectorize(phi)(x,y))
```

- 4.2.2 (4 Punkte): Programmieren Sie das mehrdimensionale Newtonverfahren für Funktionen $f : \mathbb{R}^2 \to \mathbb{R}^2$. Finden Sie die lokalen Minima und das lokale Maximum von ϕ auf U, indem Sie das Newtonverfahren für geeignete Startwerte auf das Kraftfeld $\vec{F}(x, y) = -\vec{\nabla}\phi(x, y)$ anwenden. Die Ableitungen und die Jacobi-Matrix von ϕ können analytisch berechnet und dem Programm übergeben werden. Zum Lösen des linearen Gleichungssystems können Sie np.linalg.solve verwenden.

- 4.2.3 (1 Punkt): Schreiben Sie ein Programm Zuordnung(x,y), welches jedem Startwert (x,y) die Nummer des lokalen Extremums von ϕ zuordnet: Wenn Konvergenz gegen lokales Minimum k: return k; wenn Konvergenz gegen lokales Maximum: return 5; wenn Konvergenz gegen einen anderen Punkt: return 0.

  Die Nummerierung der lokalen Minima können Sie selbst festlegen. Plotten Sie Ihr Ergebnis aus 4.2.3 in 2D:

```
Zuordnungvec=np.vectorize(Zuordnung)
plt.imshow(Zuordnungvec(x,y),extent=[-1,1,1,-1])
plt.colorbar()
```

- 4.2.4 (2 Punkte): Implementieren Sie folgende Iteration:

$$(x_{n+1}, y_{n+1}) = (x_n, y_n) + \epsilon \vec{F}(x_n, y_n)$$

  Verwenden Sie $\|\vec{F}(x_n, y_n)\|_2 < \delta$ als Abbruchbedingung. Geben Sie ein Wertepaar für ϵ und δ an, welches zur Minimasuche von ϕ auf U geeignet ist.

- 4.2.5 (1 Punkt): Plotten Sie die Zuordnung in die Minima analog zu 4.2.3 für den Algorithmus aus 4.2.4.

- 4.2.6 (1 Punkt): Vergleichen Sie das 2D-Newtonverfahren und den Algorithmus 4.2.4 inbesondere im Hinblick auf die Untersuchung einer Potentiallandschaft. Wie müsste der Algorithmus 4.2.4 modifiziert werden, um Maxima suchen zu können?