Computerphysik WS 2015/2016

Prof. Dr. Roland Netz, FU Berlin

Übungsblatt 4: Iterative Lösung von Gleichungssystemen

Julian Kappler, Markus Miettinen
5. November 2015

Allgemeine Hinweise

Abgabetermin für die Lösungen ist

- **Sonntag, 16.11., 24:00 Uhr.**

Die Lösungen sollten in Form eines IPython Notebooks (`*.ipynb`) abgegeben werden. Bitte achten Sie dabei auf eine sinnvolle Benennung aus der Ihr Name hervorgeht. Zur Abgabe schicken Sie die Lösungsdatei im Anhang einer Email an Ihren Tutor.

Aufgabe 4: Iterative Lösung von Gleichungssystemen (20 Punkte)

Wir betrachten im Folgenden ein lineares Gleichungssystem der Form $\boldsymbol{A}\vec{x} = \vec{b}$. Zerlegt man die Matrix $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{R}$ in eine Diagonalmatrix $\boldsymbol{D}$ sowie eine untere und eine obere Dreiecksmatrix ($\boldsymbol{L}$ und $\boldsymbol{U}$), so kann man ein Fixpunktverfahren für dessen Lösung verwenden.

Beim Jacobi Verfahren fängt man mit einem beliebiger Startvektor $\vec{x}^{(0)}$ an und löst dann iterativ

$$\vec{x}^{(k+1)} = \boldsymbol{D}^{-1}\left[(-\boldsymbol{L} - \boldsymbol{R})\,\vec{x}^{(k)} + \vec{b}\right]. \tag{1}$$

Gleichung (1) besagt, dass die Komponenten $x_i^{(k+1)}$ von $\vec{x}^{(k+1)}$ aus den Komponenten $x_j^{(k)}$ von $\vec{x}^{(k)}$ durch

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(-\sum_{j\neq i} a_{ij}\cdot x_j^{(k)} + b_i\right), \qquad i = 1, ..., n \tag{2}$$

berechnet werden, wobei a_{ij}, b_i die Komponenten von $\boldsymbol{A}$, $\vec{b}$ bezeichnen.

Als Konvergenzkriterium kann der Abstand zweier aufeinanderfolgender Vektoren gewählt werden, $\left(\vec{x}^{(k+1)} - \vec{x}^{(k)}\right)^2 < \epsilon^2$, wobei ϵ die gewünschte Genauigkeit ist.

Bemerkung: Wir verzichten in dieser Aufgabe darauf, eine Pivotisierung durchzuführen.

- 4.1 (3 Punkte): Implementieren Sie eine Funktion für das Jacobi-Verfahren. Ihre Funktion sollte als Argumente $\boldsymbol{A}$, $\vec{b}$ nehmen, sowie als optionale Argumente $\vec{x}^{(0)}$, ϵ und eine maximale Anzahl an Iterationsschritten $N_{\text{iter,max}}$. Verwenden Sie $\vec{x}^{(0)} = \vec{0}$, $\epsilon = 10^{-9}$, $N_{\text{iter,max}} = 100$ als Standardwerte für die optionalen Argumente. Die Funktion sollte ein Tupel bestehend aus dem Lösungsvektor sowie der benötigten Anzahl an Iterationsschritten zurückgeben. Wurde keine Konvergenz in $N_{\text{iter,max}}$ Iterationsschritten erzielt, soll eine Fehlermeldung ausgegeben werden.

  Hinweis: Optionale Argumente mit Standardwerten werden hier erklärt: `https://docs.python.org/release/3.4.3/tutorial/controlflow.html#default-argument-values`

Das Gauss-Seidel Verfahren ähnelt dem Jacobi Verfahren. Allerdings nutzt man nun die Tatsache, dass beim komponentenweisen Lösen der Zeile i im Schritt $k+1$ schon $i-1$ Komponenten von $\vec{x}^{(k+1)}$ bestimmt worden sind. Als Iterationsvorschrift ergibt sich

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} \cdot x_j^{(k)} \right), \qquad i = 1, ..., n, \tag{3}$$

wobei a_{ij}, $x_i^{(k+1)}$, $x_i^{(k)}$, b_i die Komponenten von $\boldsymbol{A}$, $\vec{x}^{(k+1)}$, $\vec{x}^{(k)}$, $\vec{b}$ bezeichnen.

- 4.2 (3 Punkte): Implementieren Sie analog zur vorherigen Aufgabe eine Funktion für das Gauss-Seidel Verfahren, mit denselben Argumenten, Standard- und Rückgabewerten wie in 4.1.
- 4.3 (2 Punkte): Lösen Sie das lineare Gleichungssytem

$$\begin{pmatrix} 7 & -2 & 1 & 2 \\ 2 & 8 & 3 & 1 \\ -1 & 0 & 5 & 2 \\ 0 & 2 & -1 & 4 \end{pmatrix} \vec{x} = \begin{pmatrix} 3 \\ -2 \\ 5 \\ 4 \end{pmatrix} \tag{4}$$

  mithilfe Ihrer in Teilaufgaben 4.1 und 4.2 geschriebenen Funktionen, und vergleichen Sie ihre Lösunsvektoren mit dem Ergebnis von `numpy.linalg.solve`. Welches der beiden selbstimplementierten Verfahren benötigt mehr Iterationsschritte?

  Hinweis: Verwenden Sie als Konvergenzkriterium das in 4.1. gegebene ϵ sowie den dort gegebenen Startvektor.

- 4.4 (2 Punkte): Versuchen Sie nun, das lineare Gleichungssytem

$$\begin{pmatrix} 1 & 4 & 2 \\ 5 & 3 & 6 \\ 9 & 8 & 7 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \tag{5}$$

  mithilfe Ihrer in Teilaufgaben 4.1 und 4.2 geschriebenen Funktionen sowie mit `numpy.linalg.solve` zu lösen. Was stellen Sie fest? Ist die Koeffizientenmatrix $\boldsymbol{A}$ hier diagonaldominant?

  Hinweis: Verwenden Sie auch hier als Konvergenzkriterium das in 4.1. gegebene ϵ sowie den dort gegebenen Startvektor. Verwenden Sie weiterhin $N_{\text{iter,max}} = 300$. Eine Matrix heisst diagonaldominant, wenn in jeder Zeile der Betrag des Diagonalelements größer ist als die Summe der Beträge der restlichen Einträge derselben Zeile.

Nun wollen wir die Laufzeiten von Jacobi Verfahren, Gauss-Seidel Verfahren und `numpy.linalg.solve` vergleichen. Dazu wollen wir die Zeit messen, die gebraucht wird, um zufällig generierte Gleichungssysteme verschiedener Dimension N zu lösen.

Für gegebenes N soll $\boldsymbol{A}$ die Form $\boldsymbol{A} = \boldsymbol{R} + N \cdot \mathbf{1}_N$ haben, wobei $\boldsymbol{R}$ eine $(N \times N)$ Matrix mit in [0,1) gleichverteilten, zufälligen Einträgen ist, und $\mathbf{1}_N$ die $(N \times N)$ Einheitsmatrix. $\vec{b}$ soll ein N-dimensionaler Vektor mit in [0,1) gleichverteilten, zufälligen Einträgen sein.

- 4.5 (1 Punkt): Ist eine nach obigem Algorithmus zufällig generierte Matrix $\boldsymbol{A}$ diagonaldominant?

- 4.6 (4 Punkte): Implementieren Sie Funktionen, welche für Jacobi Verfahren, Gauss-Seidel Verfahren und `numpy.linalg.solve` jeweils die zum Lösen von zufällig generierten Gleichungssystemen benötigte Zeit bestimmen. Jede Funktion soll über die Resultate für M zufällig generierte Gleichungssysteme mitteln.

  Ihre Funktionen sollen als Eingabeparameter die Dimension N des Gleichungssystems sowie die Anzahl M der Wiederholungen nehmen. Rückgabewerte der Funktionen sollen die gemittelte Laufzeit und für die iterativen Verfahren auch die gemittelte Anzahl an benötigten Iterationsschritten sein.

  Hinweis: `numpy.arrays` *mit in* [0,1) *gleichverteilten, (pseudo)zufälligen Einträgen können mit* `numpy.random.random()` *erzeugt werden. Für* ϵ, $\vec{x}^{(0)}$, $N_{\text{iter,max}}$ *können die Standardwerte aus 4.1 verwendet werden. Zur Zeitmessung können Sie* `time.perf_counter()` *verwenden, um die Zeitdifferenz zwischen vor und nach dem Aufruf des jeweiligen Gleichungssystemlösers zu bestimmen.*

- 4.7 (2 Punkte): Berechnen Sie, wie viel Speicher `numpy.array`s vom Datentyp 64 bit float und Dimensionen $(10^4 \times 10^4)$, $(10^5 \times 10^5)$ jeweils ungefähr benötigen. Würden Sie das Ausführen Ihrer Funktionen aus 4.6 mit $N = 10^5$ auf einem handelsüblichen Desktoprechner empfehlen?

- 4.8 (3 Punkte): Berechnen Sie mithilfe Ihrer in 4.4 implementierten Funktionen für

$$N \in \{10, 31, 100, 316, 1000, 3162, 10000\} \tag{6}$$

  die gemittelte Laufzeiten und die gemittelte Anzahl an Iterationsschritten für Jacobi Verfahren und Gauss-Seidel Verfahren, sowie die gemittelte Laufzeit für `numpy.linalg.solve`. Lassen Sie ihre Funktionen jeweils über $M = 5$ Wiederholungen mitteln.

  Stellen Sie die gemittelte Laufzeiten und die gemittelte Anzahl benötigter Iterationsschritte als Funktionen von N in jeweils einem gemeinsamen doppellogarithmischen Plot dar.

  Wie verhalten sich die Laufzeiten und durchschnittlich benötigten Iterationsschritte von Jacobi Verfahren und Gauss-Seidel Verfahren zueinander, wie die Laufzeiten der selbstimplementierten Funktionen relativ zu `numpy.linalg.solve`?