Praktikum Wissenschaftliches Rechnen

Lineare Gleichungssysteme

Sebastian Schöps, Timon Seibel

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2025
09. Juni 2025
Übungsblatt 4

Ziel dieser Übung ist es, das Gaußsche Eliminationsverfahren in `Python` zu implementieren. Hierzu wird Ihnen auf `Moodle` das Grundgerüst des abzugebenden `Python`-Projekts zur Verfügung gestellt. Ihre Aufgabe ist es, den Code entsprechend der Aufgabenstellung fertigzustellen.
Sie dürfen weitere (Hilfs-)Methoden hinzufügen, um Ihren Code übersichtlicher zu gestalten. Aus der Sicht des Software Engineerings wird dies sogar empfohlen. Kommentieren Sie Ihren Code, sodass er für Dritte verständlich ist.

Wichtig: Die Übung ist **in Gruppen** zu bearbeiten und die vorgegebenen (noch zu vervollständigenden) Methoden sind zwingend zu verwenden. Insbesondere dürfen weder die Namen der vorgegebenen Methoden noch die Reihenfolge ihrer Argumente abgeändert werden. Ebenso dürfen Sie die Rückgabewerte nicht ändern. Andernfalls könnten die zugehörigen Tests fehlschlagen, was zu Punktverlust führt.
Weiterhin möchten wir Sie daran erinnern, dass Sie, mit Ausnahme von `numpy` und `matplotlib`, keine externen Bibliotheken benutzen dürfen, sofern dies die Aufgabenstellung nicht ausdrücklich erlaubt.

Zur Bearbeitung der Übung stellen wir Ihnen die folgenden Dateien zur Verfügung:

lgs.py Grundgerüst der Implementierung des Gauß-Algorithmus

main.py Von hieraus sollen die geforderten Plots **automatisiert** erstellt und gespeichert werden

beobachtungen.txt Hier sollen sie Ihre Beobachtungen schriftlich festhalten

Abgabe: Komprimieren Sie alle relevanten und Verzeichnisse in `gruppe_xy_uebung_4.zip` (oder `.tar.gz`), wobei `xy` durch Ihre Gruppennummer zu ersetzen ist, und reichen Sie die Archiv-Datei bis **spätestens 22.06.2025, 23:59 Uhr** auf `Moodle` ein. Spätere Abgaben können nicht berücksichtigt werden. Reichen Sie Ihre Abgabe daher frühzeitig ein und melden Sie sich bei Problemen mit dem Upload vor Ablauf der Deadline, damit wir gegebenenfalls eine Lösung finden können.

Aufgabe 4.1: Gauß-Verfahren (10 Punkte)

In dieser Aufgabe sollen Sie den Gaußalgorithmus **ohne** Pivotsuche zur Lösung von linearen Gleichungssystemen $Ax = b$ implementieren. Im gesamten Übungsblatt gehen wir davon aus, dass $A \in \mathbb{R}^{n \times n}$ eine quadratische Matrix ist und $b \in \mathbb{R}^n$. Ein besonderer Fokus wird hierbei darauf liegen, den Input auf Korrektheit zu überprüfen.

4.1a)

Stellen Sie hierzu zunächst die Methode `check_input` fertig. Diese erwartet als Input die Systemmatrix `A: numpy.ndarray[float]` und die rechte Seite `b: numpy.ndarray[float]`. Die Methode gibt nichts zurück, wirft jedoch einen `ValueError`, wenn der Input fehlerhaft ist. Überprüfen Sie den Input auf die folgenden Punkte in folgender Reihenfolge:

- Die Matrix A ist quadratisch
- Die rechte Seite b besitzt die korrekte Länge, sprich, dass die Länge der Anzahl der Zeilen von A entspricht.

Überprüfen Sie die Matrix `A` außerdem darauf, dass sie strikt diagonal dominant ist, da andernfalls nicht garantiert werden kann, dass das System ohne Pivotsuche lösbar ist (siehe 4.2.5 im Skript). Werfen Sie in diesem Fall jedoch keinen Fehler, sondern geben Sie lediglich eine `UserWarning`[1] aus mit dem Hinweis, dass `A` nicht strikt diagonal dominant ist.

4.1b)

Vervollständigen Sie als nächstes die Methode `gauss`. Diese bekommt die Systemmatrix `A` als `numpy.ndarray[float]`, die rechte Seite `b: numpy.ndarray[float]` sowie die Toleranz `tol: float` als optionales Argument[2] übergeben. Verwenden Sie für die Toleranz den Standardwert 10^{-12}, wenn kein Wert übergeben wird. Die Methode soll zunächst den Input mit Hilfe der Methode `check_input` überprüfen und bei erfolgreicher Überprüfung, also wenn kein Fehler geworfen wird, den Lösungsvektor `x: numpy.ndarray[float]` des linearen Gleichungssystems $Ax = b$ mit dem Gauß-Algorithmus **ohne** Pivotsuche berechnen.
Überprüfen Sie immer, wenn Sie in eine neue Matrixzeile gehen, dass das entsprechende Diagonalelement $|a_{ii}| >$ `tol` erfüllt. Lassen Sie andernfalls einen `ValueError` werfen.

4.1c)

Abschließend sollen Sie Methode `frobenius_norm` implementieren. Diese bekommt eine Matrix `A` als `numpy.ndarray[float]` übergeben und gibt die Frobenius-Norm[3] von `A` als `float` zurück.
Überprüfen Sie, dass `A` eine quadratische Matrix ist und werfen Sie andernfalls einen `ValueError`.

Aufgabe 4.2: Beispiele & Plots (10 Punkte)

In dieser Aufgabe sollen Sie ein paar Beispiele durchrechnen, und Ihre Beobachtungen in `beobachtungen.txt` festhalten oder Ihre Ergebnisse plotten. Denken Sie daran, dass `main.py` bei Ausführung alle geforderten Plots automatisch und ohne äußeres Zutun erstellen und speichern soll. Sie können die Plots zusätzlich auch anzeigen lassen.

4.2a)

Verwenden Sie die Methode `gauss`, um folgendes lineares Gleichungssystem zu Lösen:

$$\begin{pmatrix} 3 & 1 & 0 & 0 & 0 \\ 1 & 3 & 1 & 0 & 0 \\ 0 & 1 & 3 & 1 & 0 \\ 0 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 1 & 3 \end{pmatrix} x = \begin{pmatrix} 4 \\ 5 \\ 5 \\ 5 \\ 4 \end{pmatrix}.$$

Lassen Sie Ihre Lösung `x` auf der Konsole ausgeben und kopieren Sie diese zusätzlich in `beobachtungen.txt`.

[1] Verwenden Sie hierzu die Methode `warn` aus dem Package `warnings`.
[2] Da `tol` ein optionales Argument ist, reicht es, `gauss(A, b)` aufzurufen, solange man keine andere Toleranz verwenden möchte. Überprüfen Sie, dass Sie in diesem Fall **keinen** Fehler erhalten.
[3] Die Frobenius-Norm einer Matrix $A \in \mathbb{R}^{n \times n}$ ist definiert als $\|A\|_F := \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$.

4.2b)

Wenden Sie `gauss` auf folgendes System an:

$$\begin{pmatrix} 1 & 2 & 2 \\ 2 & 4 & 6 \\ 1 & -1 & 1 \end{pmatrix} x = \begin{pmatrix} 2 \\ 5 \\ -3 \end{pmatrix}.$$

Welches Verhalten stellen Sie fest? Wieso tritt dieses auf? Beantworten Sie diese Fragen schriftlich in zwei bis drei Sätzen in `beobachtungen.txt`.

4.2c)

Abschließend sollen Sie sich mit der Abhängigkeit der Störungsanfälligkeit eines linearen Gleichungssystems von der Kondition der Systemmatrix befassen. Wir betrachten hierzu das folgende System mit parametrischer Systemmatrix:

$$A(t) := \begin{pmatrix} 2 & 1 \\ 1 & 2 \cdot 10^t \end{pmatrix} x(t) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} =: b.$$

Sei weiterhin $\hat{b} = (10^{-5}, 0)^\top$ eine kleine Störung und $\tilde{x}(t)$ die Lösung des gestörten Systems $A(t)\tilde{x}(t) = b + \hat{b}$.
Berechnen Sie nun für jedes $t \in \{0, 1, \dots, 9\}$ die Kondition von $A(t)$ bezüglich der Frobenius-Norm und lösen Sie sowohl das gestörte als auch das ungestörte Gleichungssystem. Berechnen Sie nun mit den zugehörigen Lösungen $\tilde{x}(t)$ und $x(t)$ den relativen Fehler $\frac{\|\tilde{x}(t)-x(t)\|_2}{\|x(t)\|_2}$. Plotten Sie den relativen Fehler über der Kondition in einem doppellogarithmischen Plot und exportieren Sie diesen als `Rel_Error_VS_Cond.pdf`. Denken Sie an an die Achsenbeschriftungen, einen Titel und eine Legende.
Was stellen Sie fest? Halten Sie Ihre Beobachtungen in `beobachtungen.txt` in ein bis zwei Sätzen schriftlich fest.
Hinweis: Zur Berechnung der Kondition von $A(t)$ benötigen Sie auch die Norm von $A^{-1}(t)$. Berechnen Sie hierzu am besten die Inverse $A^{-1}(t)$ per Hand und übergeben Sie diese zur Normberechnung an `frobenius_norm`.

Aufgabe 4.3: Tests

Damit Sie selbst einen Eindruck davon erhalten können wie automatisierte Tests in `Python` aussehen, stellen wir Ihnen die Datei `test.py` zur Verfügung, welche bereits Beispieltests beinhaltet. Diese überprüfen lediglich, dass Ihre Implementierung den korrekten Input entgegennimmt und der Output das richtige Format hat. Gerne dürfen Sie für sich selbst weitere Tests ergänzen, um Ihre Implementierung hiermit zu überprüfen. Das wird aber nicht bewertet.