Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Dr. Felix Schindler

28. September 2017

Übung zum Kompaktkurs

# Einführung in die Programmierung zur Numerik mit Python

Sommersemester 2017 — Blatt 4

**Aufgabe 1** (Potenz-Methode)

Schreiben Sie ein Programm, das für eine gegebene positiv definite Matrix $A \in \mathbb{R}^{n \times n}$ den größten Eigenwert $\sigma > 0$ mit zugehörigem Eigenvektor $r \in \mathbb{R}^n$ näherungsweise bestimmt. Verwenden Sie dazu die iterative *Potenz-Methode* zur Bestimmung des größten Eigenvektors:

- Sei ein zufälliger Startvektor $r^{(0)} \in \mathbb{R}^n$ gegeben, welcher nicht im Kern von $A$ liegt ($Ar^{(0)} \neq 0$).
- Für $n > 0$ ist in jedem Iterationsschritt die neue Annäherung durch $r^{(n)} := \frac{Ar^{(n-1)}}{||Ar^{(n-1)}||}$ gegeben, wobei $||v|| := \sum_{i=1}^{n} v_i^2$ die euklidische Norm von $v \in \mathbb{R}^n$ sei.

Eine Approximation des zugehörigen Eigenwertes ist durch den *Rayleigh*-Quotienten $\sigma(A, r) := \frac{<r,Ar>}{<r,r>}$ gegeben.

Gehen Sie dazu in folgenden Schritten vor:

- Schreiben Sie eine Funktion `rayleigh_quotient(A, r)`, welche den Rayleigh-Quotienten $\sigma(A, r)$ berechnet und zurück gibt.
- Schreiben Sie eine Funktion `find_initial_value(A)`, welche zu $A$ einen geeigneten Startwert $r^{(0)}$ findet und zurück gibt.
  *Hinweis:* Suchen Sie dazu im `numpy.random` Modul eine geeignete Funktion, die einen zufälligen Vektor passender Größe erstellt.
- Schreiben Sie eine Funktion `power_method(A, n)`, welches die Approximation des Eigenvektors $r^{(n)}$ iterativ bestimmt und in jedem Schritt den Rayleigh-Quotienten $\sigma(A, r^{(n)})$ ausgibt.
- Schreiben Sie eine Funktion `test(A, r, sigma)`, welche den Approximationsfehler $Ar - \sigma r$ ausgibt.

Testen Sie ihr Programm für die folgenden Matrizen mit $n = 20$ Iterationen.

$$B = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad C = \begin{pmatrix} 4 & -1 & -1 \\ 0.5 & 2 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

**Aufgabe 2** (Sylvester-Kriterium)

Es sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix. Wir sagen, dass $A$ *positiv definit* ist, wenn $x^\top Ax > 0$ für alle $x \in \mathbb{R}^n \setminus \{0\}$ gilt. Nach dem *Kriterium von Sylvester* ist dies genau dann der Fall, wenn $\det(A_k) > 0$ für alle sogenannten *Hauptminoren* $A_k := (A_{ij})_{i,j=1}^k \in \mathbb{R}^{k\times k}$ gilt.

Schreiben Sie ein Programm, das für eine gegebene Matrix den Test auf positive Definitheit durchführt. Gehen Sie in folgenden Schritten vor:

(a) Implementieren Sie eine Funktion `determinante(A)`, welche die Determinante von `A` ausrechnet.

(b) Schreiben Sie eine weitere Funktion `sylvester(A)`, die für die gegebene Matrix `A` das Sylvester-Kriterium überprüft und je nach Ergebnis einen entsprechenden Wahrheitswert zurückgibt.

Testen Sie das Programm an den folgenden beiden Matrizen:

$$A = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -2 & 0 \\ 0 & -2 & 5 & -3 \\ 0 & 0 & -3 & 7 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 1 & 5 & 7 \\ 1 & 2 & 6 & 8 \\ 5 & 6 & 1 & 4 \\ 7 & 8 & 4 & 0 \end{pmatrix}$$

*Hinweis:* *Die Determinante einer Matrix $A \in \mathbb{R}^{n\times n}$ können Sie zum Beispiel mit dem* Laplace'schen Entwicklungssatz *berechnen: Sei $M_{ij} \in \mathbb{R}^{(n-1)\times(n-1)}$ die Matrix, die aus $A$ durch Streichen der $i$-ten Zeile und $j$-ten Spalte hervorgeht (auch* Minor *genannt). Dann gilt*

$$det(A) = \sum_{i=1}^{n} (-1)^{i+j} \cdot a_{ij} \cdot det(M_{ij})$$

*für ein fest gewähltes $j \in \{1, \ldots, n\}$. Schreiben Sie dazu eine weitere Hilfsfunktion* `minor(A,i,j)`*, welche den zu $A$ gehörigen Minor $M_{ij}$ liefert.*

**Aufgabe 3** (Conway's Game of Life)

Der britische Mathematiker John Conway hat sich folgendes "Spiel" ausgedacht: Wir betrachten ein Gitter mit $n \times n$ Zellen, wobei jede Zelle entweder leer (tot) oder besetzt (lebendig) ist. Hat man eine bestimmte Verteilung von leeren und besetzten Zellen vorliegen, so bestimmt sich die nächste Verteilung (welche die aktuelle ersetzt) nach folgenden Regeln: Für jede Zelle werden die acht Nachbarzellen betrachtet. Es gilt dann:

- Eine leere Zelle mit genau drei besetzten Nachbarn ist in der nächsten Verteilung besetzt.
- Eine besetzte Zelle bleibt in der nächsten Verteilung genau dann besetzt, wenn sie 2 oder 3 besetzte Nachbarn hat.

Schreiben Sie ein Programm, welches das Game of Life simuliert. Gehen Sie dazu so vor:

(a) Legen Sie für eine von Ihnen gewählte Größe $n$ das Spielfeld als `numpy` array an.

(b) Schreiben Sie eine Funktion `naechste_generation(feld)`, welche für ein gegebenes Spielfeld `feld` die nächste Konfiguration berechnet und zurückliefert. Beachten Sie dabei nur das Innere des Spielfeldes, d.h. setzen Sie die Randfelder stets auf leer.

(c) Implementieren Sie außerdem eine Funktion `ausgeben(feld)`, die das Spielfeld in geeigneter Weise auf dem Terminal ausgibt.

(d) Initialisieren Sie ein Spielfeld mit einer von Ihnen gewählten Anfangskonfiguration und lassen Sie es sich über einige Iterationen entwickeln.