Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Mario Ohlberger
Dr. Christoph Lehrenfeld

16. März 2016

Übung zum Kompaktkurs

Einführung in die Programmierung zur Numerik mit Python

Wintersemesterferien 2015/2016 — Blatt 3

Aufgabe 1 (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 auch, ob die gegebene Matrix überhaupt symmetrisch ist.

Testen Sie das Programm an den folgenden beiden Matrizen:

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

Hinweise: *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 (Sie wird 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 $j \in \{1, \dots, n\}$. Schreiben Sie dazu eine weitere Hilfsfunktion `minor(A,i,j)`*, welche den zu A gehörigen Minor M_{ij} liefert.*

Aufgabe 2 (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 das Spielfeld `feld` mit der nächsten Konfiguration überschreibt. 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.

Wenn Sie Zeit und Lust haben, können Sie die folgenden Bonusaufgaben erledigen:

- Machen Sie die Ränder des Spielfeldes periodisch, d.h. lassen sie alles was das Feld über einen Rand verlässt auf der gegenüber liegenden Seite wieder hereinkommen. Dazu müssen Sie die Funktion `naechste_generation(feld)` um die entsprechende Behandlung der Randfelder erweitern.