Heinrich Heine Universität Düsseldorf

Mathematisches Institut
Prof. Dr. Achim Schädle
Melinda Hagedorn

19.12.2024

Computergestützte Mathematik zur linearen Algebra – 11. Übungsblatt

Aufgabe 41: (*Lineare Unabhängigkeit von Polynomen*)

Lösen Sie diese Aufgabe unter Verwendung der Klasse `Polynom`, deren aktuelle Version Sie ab Donnerstag Nachmittag auf der Vorlesungshomepage finden. Unter anderem können Polynome nun auch mit natürlichen Zahlen potenziert werden.

(a) Schreiben Sie eine Funktion `LinUnabh_Poly`, die als Eingabe eine Liste mit beliebig vielen Polynomen erhält und diese auf lineare Unabhängigkeit untersucht. Sind alle Polynome linear unabhängig, soll `True` ausgegeben werden, andernfalls `False`.

Hinweis: Der Rang einer Matrix, den Sie mit der Funktion `zeilenstufen` aus Aufgabe 31 bestimmt haben, ist hier hilfreich. Sollten Sie die Aufgabe nicht bearbeitet haben, können Sie für dessen Berechnung auch `np.linalg.matrix_rank` verwenden.

(b) Testen Sie Ihre Funktion anhand der beiden Listen $[x^2-1,\ x-1]$ und $[x^2-1,\ 2x^2-2]$, sowie anhand der Bernstein-Polynome vom Grad 4, die definiert sind als

$$B_{i,4}:\ \mathbb{R}\to\mathbb{R},\quad x\mapsto \binom{4}{i}x^i\,(1-x)^{4-i}\quad \text{für}\quad i=0,\dots,4.$$

Aufgabe 42: (*Lineare Gleichungssysteme*)

(a) Schreiben Sie eine Funktion `x=lgsloeser(A,b)`, welche das lineare Gleichungssystem $Ax=b$ mit $A\in\mathbb{R}^{n\times n}$ und $b\in\mathbb{R}^n$ mit Hilfe der LR-Zerlegung und Vorwärts-/Rückwärtssubstitution löst.
Hinweis: Sie können für die LR-Zerlegung Ihre korrigierte Version des Codes aus Aufgabe 35 verwenden oder die Python-Funktion `scipy.linalg.lu`. Beachten Sie, dass diese Funktion Arrays `P, L, R` liefert, sodass `A` $=$ `PLR` gilt. (In Aufgabe 35 gilt stattdessen $PA=LR$.)
Für die Vorwärtssubstitution können Sie Ihre Implementierung von Aufgabe 33 benutzen oder deren Musterlösung von der Homepage herunterladen. In dieser finden Sie auch die Implementierung der Rückwärtssubstitution.

(b) Auf der Vorlesungsseite finden Sie das Modul `Aufgabenkontrolle11`. Laden Sie es in Ihr Arbeitsverzeichnis in denselben Ordner, in dem Sie auch das Blatt bearbeiten, herunter und importieren Sie es mittels `import Aufgabenkontrolle11`. Testen Sie Ihren Gleichungssystemlöser mit Hilfe der Funktion `Aufgabenkontrolle11.Aufgabe42(lgsloeser)`. Lesen Sie hierfür den Hilfetext zu dieser Funktion durch.

b.w.

Aufgabe 43: (*LR-Zerlegung mit Brüchen*)

Auf der Homepage finden Sie das Modul `bruch.py` mit einer erweiterten und verbesserten Version der Klasse `Bruch` aus Aufgabe 15. Wir wollen diese nun mit der *LR*-Zerlegung aus Aufgabe 35 verbinden, sodass wir die Zerlegung einer Matrix $A \in \mathbb{Q}^{n\times n}$ exakt berechnen können. Die Musterlösung zu Aufgabe 35 finden Sie ab Donnerstag Nachmittag ebenfalls auf der Homepage.

(a) Sehen Sie sich die aktuelle Version der Klasse `Bruch` an. Welche Methoden sind dazugekommen und wozu dienen sie?

(b) Für einen Array mit Einträgen aus der Klasse `Bruch` wollen wir `max` und `np.argmax` bestimmen können, d.h. die Brüche müssen sortierbar sein. Das lässt sich erreichen, indem Sie die `Bruch`-Klasse um die passenden Methoden für die Operationen `<`, `<=`, `>`, `>=` ergänzen. Implementieren Sie auch die passende Methode zur Bestimmung des Absolutbetrags, so dass `abs(b)` für einen `Bruch b` funktioniert.
Hinweis: Die Seite https://www.python-kurs.eu/python3_magische_methoden.php hilft bei der Suche nach den passenden Methoden-Namen.

(c) Passen Sie nun die Funktion `LR_kompakt(A)` aus Aufgabe 35 soweit an, dass die Eingabe ein Array `A` ist, dessen Einträge sowohl `Integer` als auch `Bruch`-Objekte sein können. Als Pivotelement soll das betragsgrößte Element jeder Spalte gewählt werden, vgl. 35(c).

(d) Testen Sie Ihre neue *LR*-Zerlegung mit Hilfe der Funktion `TestLR_Bruch(testfunktion)` aus dem Modul `Aufgabenkontrolle11`.

Aufgabe 44: (*Gram-Schmidt-Verfahren für Vektoren*)

In den Folien zur Wiederholung der Linearen Algebra finden Sie die Pseudocodes für das klassische und das modifizierte Gram-Schmidt-Verfahren zur Orthogonalisierung von Vektoren $a_1, a_2, \ldots, a_n$.

(a) Schreiben Sie eine Funktion `GramSchmidt(A)`, in der Sie das klassische Gram-Schmidt-Verfahren implementieren und auf die Basisvektoren $\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_n$ anwenden, die der Funktion als die Spalten eines Arrays `A` übergeben werden.

(b) Implementieren Sie nun auch das modifizierte Gram-Schmidt-Verfahren in einer Funktion `GramSchmidt_mod(A)`.

(c) Testen Sie Ihre Funktionen an geeigneten Zufallsmatrizen.

Besprechung in den Übungen vom 06.01.2025 bis 10.01.2025.