Heinrich Heine Universität Düsseldorf

Mathematisches Institut
Prof. Dr. Achim Schädle
Melinda Hagedorn

12.11.2026

Computergestützte Mathematik zur linearen Algebra – 5. Übungsblatt

Aufgabe 17: *(Primitive Euler-Ziegel finden)*

Ein primitiver Euler-Ziegel ist definiert als ein Quader mit ganzzahligen Kantenlängen und ganzzahligen Flächendiagonalen, dessen drei Kantenlängen keinen gemeinsamen Teiler ungleich 1 haben. Schreiben Sie einen Python-Code, der sämtliche primitiven Euler-Ziegel mit Kantenlängen von höchstens 1000 findet.

Ihr Programm soll in vertretbarer Zeit sämtliche primitiven Euler-Ziegel finden. Überlegen Sie sich gut, welche Fälle Sie wirklich überprüfen müssen, damit Sie Rechenzeit sparen.

Ein `Python` Programm, das sämtliche primitive Pythagoreische Tripel mit Kantenlängen bis zu 50 berechnet, finden Sie zur Inspiration auf der Vorlesungsseite in der Datei `a17_euler.py`.

Aufgabe 18: *(Umgang mit Vektoren und Matrizen)*

Lösen Sie die folgenden Teilaufgaben mit Hilfe des `numpy`-Moduls in möglichst kurzen Befehlen. Ganze Zahlen sollen hierbei, wenn möglich, als Integer dargestellt werden.

(a) Erstellen Sie eine (5×5)-Matrix mit den Einträgen 1, 2, 3, 4 unterhalb der Diagonalen.

(b) Erstellen Sie eine (6×6)-Matrix mit Einsen in den Einträgen am Rand und Nullen im Inneren.

(c) Erstellen Sie eine (7×7)-Matrix mit Schachbrett-Muster (Einträge 0 und 1).

(d) Erstellen Sie eine Zufallsmatrix der Größe 6×6 mit natürlichen Zahlen zwischen 0 und 99. Vertauschen Sie nun die erste mit der vorletzten Zeile.

(e) Erstellen Sie einen zufälligen Vektor der Länge 10 mit natürlichen Zahlen zwischen 1 und 10. Ändern Sie nun das Vorzeichen jedes Eintrags, der echt zwischen 3 und 8 liegt.

(f) Erstellen Sie einen Zufallsvektor der Größe 10 mit Einträgen aus dem Intervall $[0, 1)$ und ersetzen Sie den größten Eintrag durch 0.

(g) Erstellen Sie einen Zufallsvektor der Größe 10 mit Einträgen aus dem Intervall $[0, 1)$ und finden Sie den Eintrag, der am nächsten an 0.5 liegt.

Hinweis: Für die Zufallsvektoren und -matrizen sind die Befehle `random.randint` und `random.rand` aus dem `numpy`-Modul hilfreich. Sehen Sie sich die Hilfeseiten der Befehle an, um genaueres über die Verwendung zu erfahren.

b.w.

Aufgabe 19: (*Funktionsparameter und Plots*)

(a) Schreiben Sie eine Funktion `plotFunktion_N(f,a,b,N)`, welche die Funktion $f(x)$ auf dem Intervall $[a, b]$ mit N äquidistanten Punkten plottet. `f` ist hierbei das Functionhandle von $f(x)$, `a`, `b` sind Floats und `N` ist ein Integer. Wird kein Wert für `N` übergeben, soll `N=300` gesetzt werden.

(b) Schreiben Sie eine Funktion `plotFunktion_h(f,a,b,h)`, welche die Funktion $f(x)$ auf dem Intervall $[a, b]$ so plottet, dass benachbarte Stellen, an denen geplottet wird, den Abstand h haben. Der letzte Abstand darf diese Bedingung verletzen. `f` ist hierbei das Functionhandle von $f(x)$, `a`, `b` und `h` sind Floats. Wird kein Wert für `h` übergeben, soll `h=1e-3` gesetzt werden.

(c) Testen Sie Ihre Funktionen mit

$$f(x) = 0.3x^2 - 1.7\sin(4x),$$

wobei $a = -1$, $b = 4$ und $N = 100$ bzw. $N = 300$ sowie $h = 2/11$ und $h = 10^{-3}$.

Aufgabe 20: (*Matriz-Vektor Multiplikation*)

Seien $A = (a_{ij})_{i=0,j=0}^{m-1\ n-1}$ eine $m \times n$ Matrix und u ein Vektor der Dimension n. $a_{:,j}$ bezeichnet hier die j-te Spalte von A und $a_{i,:}$ die i-te Zeile. Die elementweise Multiplikation (Hadamard-Produkt) zwischen zwei Matrizen A und B der gleichen Dimension wird im Folgenden mit $A \odot B$ bezeichnet. Implementieren Sie die folgenden Verfahren zur Berechung von $x = Au$ und stellen Sie fest wie schnell diese sind.

(a) Berechne x mit zwei For-Schleifen.

$x = 0$ Initialisierung mit Null
for i ***von*** 0 ***bis*** $m-1$ **:**
  for j ***von*** 0 ***bis*** $n-1$ **:**
    $x_i = x_i + a_{ij}u_j$

(b) Berechne x mit einer For-Schleife als Linearkombination der Spalten von A.

$x = 0$ Initialisierung mit Null
for j ***von*** 0 ***bis*** $n-1$ **:**
  $x = x + a_{:,j}u_j$

(c) Berechne x mit einer For-Schleife. Die i-te Zeile von A wird elementweise mit u multpliziert und das Ergebnis mit `sum` oder `np.sum` aufsummiert.

$x = 0$ Initialisierung mit Null
for i ***von*** 0 ***bis*** $m-1$ **:**
  $x_i = x_i + sum(a_{i,:} \odot u)$

(d) Berechne x durch die NumPy-Matrix-Vektor-Multiplikation von A und u.

Zusatz (Für Physik-Interessierte) Sie dürfen auch die Einstein'sche Summenkonvention benutzen. Dafür steht Ihnen die Funktion `np.einsum` zur Verfügung.

Besprechung in den Übungen vom 16.11.2026 bis 19.11.2026.