Institut für Angewandte und
Numerische Mathematik

PD Dr. Mathias J. Krause
M.Sc. Stefan Karch
M.Sc. Mariia Sukhova

28.11.2022

Einstieg in die Informatik und Algorithmische Mathematik

Aufgabenblatt 7

Bearbeitungszeitraum: 12.12.2022 – 23.12.2022

Aufgabe 1 (Pflichtaufgabe) *Die Matrix-Vektor-Multiplikation*

Sei $A \in \mathbb{R}^{m,n}$ eine *Matrix* mit m *Zeilen* und n *Spalten* und $x \in \mathbb{R}^n$ ein n-dimensionaler Vektor, so ist das *Matrix-Vektor-Produkt* Ax ein m-dimensionaler Vektor. Das Matrix-Vektor-Produkt ist folgendermaßen definiert: Sei $y := Ax$, mit

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix},$$

so gilt

$$y_i := \sum_{j=1}^{n} a_{ij} x_j \qquad \text{für } i = 1, 2, \dots, m.$$

Schreiben Sie ein Java-Programm, in dem die Matrix-Vektor-Multiplikation realisiert wird. Gehen Sie dabei folgendermaßen vor.

(a) Erstellen Sie eine öffentliche Klasse namens `MatrixVektor`. Definieren Sie in dieser Klasse zunächst eine Klassenmethode namens `matrixEinlesen` mit zwei formalen Parametern vom Typ **`int`**. Die Methode soll mittels zweier **`for`**–Schleifen eine Matrix mit m Zeilen und n Spalten von der Konsole einlesen und als Wert vom Typ **`double`**`[][]` zurückgeben. Die Matrixdimensionen m und n sollen dabei über die formalen Parameter übergeben werden. Definieren Sie in analoger Weise eine Klassenmethode namens `vektorEinlesen` mit einem formalen Parameter vom Typ **`int`**, die einen n-dimensionalen Vektor von der Konsole einliest und als Wert vom Typ **`double`**`[]` zurück gibt.

(b) Definieren Sie nun eine Klassenmethoden namens `vektorAusgeben` mit einem formalen Parameter vom Typ **`double`**`[]` ohne Rückgabewert. Die Methode soll das übergebene Feld auf der Konsole ausgeben.

(c) Definieren Sie eine Klassenmethode namens `matrixvektorProdukt` mit zwei formalen Parametern vom Typ **`double`**`[][]` und **`double`**`[]`. Über den ersten Parameter soll eine Matrix A, über den zweiten ein Vektor x übergeben werden. Berechnen Sie in der Methode

das Matrix-Vektor-Produkt Ax, und geben Sie das Ergebnis als Wert vom Typ **`double[]`** zurück.

Hinweis: Vorsicht! Matrix- und Vektorkomponenten werden üblicherweise mit Eins beginnend durchnumeriert. Die Komponenten eines Feldes hingegen werden in Java mit Null beginnend durchnumeriert.

(d) Definieren Sie nun die `main`-Methode des Programms. Lesen Sie in dieser Methode zunächst die Dimensionen m und n von der Konsole ein. Lesen Sie danach eine Matrix $A \in \mathbb{R}^{m,n}$ und einen Vektor $x \in \mathbb{R}^n$ von der Konsole ein. Berechnen Sie das Matrix-Vektor-Produkt Ax, und geben Sie dieses auf der Konsole aus.

(e) Testen Sie ihr Programm mit folgenden Eingaben.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \qquad Ax = \begin{pmatrix} 14 \\ 32 \\ 50 \end{pmatrix}.$$

und

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \qquad Ax = \begin{pmatrix} 5 \\ 11 \\ 17 \end{pmatrix}.$$

Aufgabe 2 *Boothroyd/Dekker-Matrizen*

Die ganzzahligen Elemente einer $n \times n$ Boothroyd/Dekker-Matrix $B = (b_{ij})$ sind gegeben durch

$$b_{ij} = \binom{n+i-1}{i-1} \cdot \binom{n-1}{n-j} \cdot \frac{n}{i+j-1}, \quad 1 \leq i, j \leq n.$$

Dabei bezeichnet $\binom{n}{k}$ den ganzzahligen Binomialkoeffizienten „n über k“ mit

$$\binom{n}{k} := \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot \ldots \cdot k}.$$

Bearbeiten Sie, die folgenden Teilaufgaben in einer Java-Klasse.

- Schreiben Sie eine Methode `ueber` zur Berechnung des Binomialkoeffizienten $\binom{n}{k}$ gemäß folgender Vorschrift:

  $$c_0 := 1, c_i := \frac{c_{i-1} \cdot (n-i+1)}{i}, \qquad \text{für} \quad i = 1, \ldots, k.$$

  Dann gilt $c_k = \binom{n}{k}$. Beachten Sie, dass die `integer`-Division zu verwenden ist. Der Methode sollen zwei ganzzahlige Variablen k und n als Argumente übergeben werden. Der Rückgabewert ist eine Gleitkommazahl.

- Schreiben Sie eine Methode `boDeMatrix`, ohne Rückgabewert, mit einem ganzzahligen Argumenten und einem Argument vom Typ `double[][]`, welche eine $n \times n$ Boothroyd/Dekker-Matrix berechnen soll. Die Berechnung soll nach nach obiger Formel erfolgen. Die berechnete Matrix soll in die übergebene Matrix abgespeichert werden.

- Schreiben Sie eine Methode `ausgabe` ohne Rückgabewert, welche eine Boothroyd/ Dekker-Matrix zeilenweise auf der Konsole ausgeben soll.
- Schreiben Sie eine Methode `norm` vom Gleikommatyp zur Berechnung der *Frobeniusnorm* einer Matrix. Die beiden Argumente der Methode sind vom ganzzahligen Typ und vom Typ `double[][]`. Die Frobeniusnorm einer $n \times n$ Matrix A ist wie folgt definiert

$$\|A\|_F := \sqrt{\sum_{i=1}^{n} \left(\sum_{j=1}^{n} a_{ij}^2 \right)} \; .$$

  Für diese Berechnung eignen sich zwei `for-Schleifen`.
- Schreiben Sie ein Hauptprogramm, in dem mit Hilfe einer `for-Schleife` für die Dimensionen 5 bis 10 eine $n \times n$ Boothroyd/Dekker-Matrix B berechnet wird, und die Matrix und ihre Frobeniusnorm auf der Konsole ausgegeben werden.

Musterlösung: Für $n = 5$ sieht die Boothroyd/Dekker-Matrix folgendermaßen aus:

$$B_5 = \begin{pmatrix} 5 & 10 & 10 & 5 & 1 \\ 15 & 40 & 45 & 24 & 5 \\ 35 & 105 & 126 & 70 & 15 \\ 70 & 224 & 280 & 160 & 35 \\ 126 & 420 & 540 & 315 & 70 \end{pmatrix} \text{ mit } \|A\|_F = 886.7102119632998$$

Aufgabe 3 *Cramersche Regel*

Gegeben seien eine $n \times n$ Matrix A sowie die Vektoren x und b der Länge n. Die Lösung x des linearen Gleichungssystems

$$A \cdot x = b$$

kann dann mit Hilfe der Cramerschen Regel wie folgt berechnet werden

$$\text{(*)} \qquad \det(A) \neq 0 \implies x_i = \frac{\det(A_i)}{\det(A)}, \quad i = 1(1)n.$$

Hierbei bezeichnet A_i diejenige Matrix, die aus A hervorgeht, indem man die i-te Spalte von A durch die rechte Seite b ersetzt; $\det(A)$ bezeichnet die Determinante von A.

Schreiben Sie eine Java-Klasse zur Lösung eines 3×3 Systems (d. h. $n = 3$) mit Hilfe der Cramerschen Regel. Das Programm soll folgende Teilschritte enthalten

- zwei Methoden `READVEKTOR` und `WRITEVEKTOR` für die Ein- bzw. Ausgabe eines Vektors,
- eine Eingabemethode `READMATRIX`, in der eine Matrix zeilenweise eingelesen wird,
- eine Methode `DET` mit einem formalen Argument A vom Typ `MATRIX` zur Berechnung der zugehörigen 3×3 Determinante nach folgender Vorschrift (Sarrus-Regel):

$$\begin{aligned} det(A) = \quad & a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} \\ - \quad & a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} \; , \end{aligned}$$

- eine Methode `SWAP` zur Berechnung der Matrix A_i,
- eine `main`-Methode, in der mit Hilfe der oben formulierten Unterprogramme
  - die Matrix A und die rechte Seite b eingelesen werden,
  - die Determinante $d = \det(A)$ berechnet wird,
  - falls $d = 0$ ist, die Meldung

    ```
    Das System hat keine Loesung
    ```

    ausgegeben wird,
  - ansonsten ($d \neq 0$) die Lösung x gemäß (*) berechnet und ausgegeben wird.

Achtung !
In den Unterprogrammen sind keine globalen Variablen, sondern nur Argumente aus der formalen Argumentliste oder lokale Variablen zu verwenden.

Beachten Sie, dass für größeres n die Cramersche Regel wegen des enorm wachsenden Aufwandes unpraktikabel ist.

Testen Sie Ihr Programm mit $A = \begin{pmatrix} 1 & -5 & -1 \\ 5 & -2 & 10 \\ 8 & 1 & -3 \end{pmatrix}$ und $B = \begin{pmatrix} -12 \\ 1 \\ -9 \end{pmatrix}$ mit $L = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}$.

Aufgabe 4 (Weihnachtsaufgabe) *Der Weihnachtsmann ist unterwegs...*

Am 24. Dezember hat der Weihnachtsmann viel zu tun. Mit seinem Schlitten muss er in jedem Dorf von Schornstein zu Schornstein fliegen, um Geschenke für die Kinder zu verteilen. Zum Glück muss er jeden Schornstein in einem Dorf nur genau einmal besuchen. Dennoch stellt sich für ihn die Frage, in welcher Reihenfolge er die Schornsteine besuchen sollte, um einen möglichst geringen Flugweg zurückzulegen.

Diese Fragestellung ist unter dem Begriff *TSP* (*Travelling Santa-Claus Problem**) bekannt. Bei einem TSP sind n Zielpunkte mit den Koordinaten (x_0, y_0), (x_1, y_1), ..., $(x_{n-1}, y_{n-1}) \in \mathbb{R} \times \mathbb{R}$ gegeben. Ziel ist es, Routen mit minimaler Weglänge zu finden, bei der jeder Zielpunkt genau einmal erreicht wird. Ausgangs- und Endpunkt jeder Route ist dabei (o.b.d.A.) der erste Zielpunkt mit den Koordinaten (x_0, y_0). Eine Route kann man als Permutation der endlichen Folge $(1, 2, \ldots, n-1)$ darstellen. Für n Zielpunkte ergeben sich daher $(n-1)!$ verschiedene Routen. Schreiben Sie ein Java-Programm, welches dem Weihnachtsmann helfen kann, eine optimale Flugroute zu finden. Gehen Sie dabei folgendermaßen vor.

(a) Erstellen Sie eine öffentliche Klasse namens `Weihnachtsmann`. Definieren Sie für diese Klasse eine Klassenmethode namens `fakultaet` mit einem formalen Parameter vom Typ **`int`**. Die Methode soll für eine übergebene Zahl $n \in \mathbb{N}$ die Fakultät $n!$ *rekursiv* berechnen und als Wert vom Typ **`int`** zurückgeben.

(b) Definieren Sie für die Klasse `Weihnachtsmann` die nachfolgende Klassenmethode

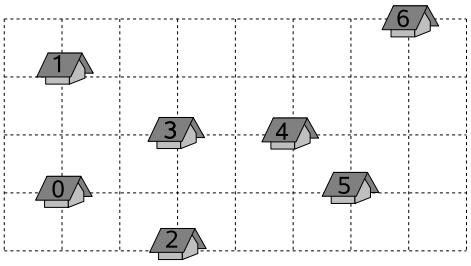
```
static int[] permutation(int[] route, int index) {
    int[] neueRoute = new int [route.length];
    System.arraycopy(route, 0, neueRoute, 0, route.length);
    int fakul = fakultaet(neueRoute.length-1);
```

```
        for (int i = 0; i < neueRoute.length - 1; i++) {
            int k = (index / fakul) % (neueRoute.length - i);
            int z = neueRoute[i + k];
            for (int j = i + k; j > i; j--) {
                neueRoute[j] = neueRoute[j - 1];
            }
            neueRoute[i] = z;
            fakul /= (neueRoute.length - (i + 1));
        }
        return neueRoute;
    }
```

Die Methode gibt die p-te Permutation ($0 \leq p < m!$) einer Folge von m natürlichen Zahlen zurück. Die Folge wird dabei über den Parameter `route`, der Index p über den Parameter `index` übergeben.

(c) Definieren Sie eine Klassenmethode namens `weglaenge` mit einem formalen Parameter vom Typ **`int`**`[]`, über den eine Route (d.h. eine Permutation von $(1, 2, \ldots, n-1)$) an die Methode übergeben werden kann, sowie zwei weiteren Parametern vom Typ **`double`**`[]`. Die Methode soll die Gesamtweglänge der übergebenen Route berechnen und als Wert vom Typ **`double`** zurückgeben. Die Koordinaten der Zielpunkte sollen über die letzten zwei Parameter an die Methode übergeben werden. Bedenken Sie, dass jede Route am ersten Zielpunkt mit den Koordinaten (x_0, y_0) beginnt und endet.

(d) Erstellen Sie nun die `main`-Methode Ihres Programms. Lesen Sie in dieser zunächst die Anzahl der Zielpunkte von der Konsole ein. Lesen Sie anschließend die x-Koordinaten $x_0, x_1, \ldots, x_{n-1}$ sowie die y-Koordinate $y_0, y_1, \ldots, y_{n-1}$ aller Zielpunkte von der Konsole ein, und speichern Sie diese in zwei geeigneten Feldern ab. Bestimmen Sie alle möglichen Routen und berechnen Sie für jede Route die Weglänge. Geben Sie die Routen mit der minimalen Weglänge auf der Konsole aus. Testen Sie Ihr Programm für das folgende Dorf:



* Tatsächlich steht TSP für *Travelling Salesman Problem*. Frohe Weihnachten!