

Hinweise zur Bearbeitung der Programmieraufgabe 15. in Java

- Für die Implementierung von Matrix- und Vektor-Strukturen für die numerische lineare Algebra gibt es mehrere Möglichkeiten.

– Die einfachste ist, Vektoren und Matrizen als 1- bzw. 2-dimensionale Arrays in der Form

```
double[] x = new double[3];  
double[][] M = new double[2][2];
```

oder bei der Initialisierung

```
double[] y = {1.0, 3.4, -1e5};  
double[][] N = {{1.0, 2.0}, {-1.5, 3.14}};
```

zu implementieren. Algebraische Operationen wie die Berechnung von Normen oder das Matrix-Vektor-Produkt muss man dann von Hand mit entsprechenden Methoden implementieren. Sofern man dabei Arrays als Argumente an Methoden übergibt, ist darauf zu achten, dass in Java hierfür nur die *call by reference*-Technik zur Verfügung steht. Ändert man also innerhalb einer Methode das übergebene Objekt, z.B.

```
public void do_something(double[] v)  
{  
    v[0] = 3.14;  
}
```

so bleiben die Änderungen innerhalb der Methode auch nach deren Beendigung erhalten, es wird *keine* lokale Kopie des Arguments erzeugt.

– Als Alternative zu Arrays werden auf der Numerik I-Homepage die Java-Klassen `Vector` und `Matrix` zur Verfügung gestellt, die schon eine gewisse Infrastruktur enthalten. Zur Handhabung dieser Klassen in eigenen Programmen sei auf die Kommentare im entsprechenden Quellcode verwiesen.

- Die im Rahmen einer Pivotstrategie auftauchenden Permutationsmatrizen

$$P_\pi := (\delta_{\pi(i),j})_{i,j=1,\dots,n}, \quad \pi \in S_n$$

sollte man natürlich *nicht* als Matrix abspeichern, sondern die folgende *indirekte Adressierung* verwenden. Zu simulieren sind letztendlich *Zeilenvertauschungen* einer Matrix M . Nummeriert man die Zeilen der ursprünglichen Matrix M von 1 bis n durch und speichert dann in einem Array

```
int[] P = new int[n];
```

an der i -ten Stelle die Zeilennummer *nach* der Permutation, so gilt $(P_\pi M)_{i,j} = M_{\pi(i),j}$, man kann also mit $M[P[i]][j]$ auf das entsprechende Matrixelement zugreifen, *ohne* M physikalisch umzusortieren.

- Falls mehrere Gleichungssysteme mit ein und derselben Matrix A gelöst werden müssen, muss die LR -Zerlegung natürlich nur einmal berechnet werden. Für die verschiedenen rechten Seiten b müssen dann nur noch die gestaffelten Systeme $Ly = b$ und $Rx = y$ berechnet werden. Ihr Algorithmus sollte dies möglichst berücksichtigen.
- Für die Matrizen L und R wird im Prinzip kein zusätzlicher Speicherplatz benötigt. Deren Einträge können in A abgelegt werden. Man sagt auch die Berechnung wird *am Platz* durchgeführt.

Bitte wenden!

Gauß-Algorithmus mit Spaltenpivotisierung

Eingabe: Matrix $A = (a_{i,j})_{i,j=1}^n$, rechte Seite $b = (b_j)_{j=1}^n$

Ausgabe: Lösung $x = (x_j)_{j=1}^n$

Sei $p = (p_j)_{j=1}^n$ ein Vektor zur Speicherung der Zeilenvertauschungen

Initialisiere: $p_j = j, j = 1, \dots, n$

// Berechnung der LR-Zerlegung und Auflösen von $Ly = b$

für $k = 1, 2$ bis $n - 1$

bestimme Index m , so dass $|a_{p_m,k}| = \max_{k \leq i \leq n} |a_{p_i,k}|$

falls $|a_{p_m,k}| > |a_{p_k,k}|$

// tausche implizit Zeile k und Zeile m :

setze $tmp = p_k, p_k = p_m, p_m = tmp$

für $i = k + 1, k + 2$ bis n

$a_{p_i,k} = a_{p_i,k} / a_{p_k,k}$

für $j = k + 1, k + 2$ bis n

$a_{p_i,j} = a_{p_i,j} - a_{p_i,k} a_{p_k,j}$

$b_{p_i} = b_{p_i} - a_{p_i,k} b_{p_k}$

// Löse $Lz = y$

für $i = n, n - 1$ bis 1

$z_{p_i} = (b_{p_i} - \sum_{k>i} a_{p_i,k} z_{p_k}) / a_{p_i,i}$

// Rücktausch

für $j = 1, 2$ bis n

$x_i = z_{p_i}$

gib x zurück