Institut für Analysis und Numerik
Universität Magdeburg
Thomas Richter

22.05.2020

Lösung zu: Übung Nr. 6 zur Algorithmische Mathematik I
Wintersemester 2019/2020

Abgabe bis zum 29. Mai 2020

Störungstheorie Man lese den Abschnitt und gebe kurze Antworten:

- Was ist die Konditionszahl einer Matrix?

Die Konditionszahl ist definiert als

$$\mathrm{cond}(A) = \|A\| \cdot \|A^{-1}\|,$$

wobei $\|\cdot\|$ eine beliebige Matrixnorm ist.

- Was ist die Konditionszahl der Matrix

$$A = \begin{pmatrix} 2 & 0 \\ 0 & -4 \end{pmatrix}$$

gemessen in der $\|\cdot\|_\infty$-Norm?

In der $\|\cdot\|_\infty$-Norm, d.h. der maximalen Zeilensumme ist

$$\|A\|_\infty = 4, \quad A^{-1} = \begin{pmatrix} 0.5 & 0 \\ 0 & -0.25 \end{pmatrix}, \quad \|A^{-1}\|_\infty = 0.5,$$

also

$$\mathrm{cond}_\infty(A) = 2.$$

- Mit welcher relativen Fehler im Ergebnis, d.h. $\|\delta x\|/\|x\|$ ist zu rechnen, wenn bei dieser Matrix die rechte Seite mit eine Fehler von 5% versehen ist? (Die Matrix sei fehlerfrei).

Der rel. Fehler wird gemäß der Konditionszahl verstärkt, es muss also von bis zu 10% Fehler im Ergebnis ausgegangen werden.

Von der Gauss-Elimination zur LR-Zerlegung Man gebe kurze Antworten

1. Was ist der Aufwand zum Lösen eines LGS mit einer Dreiecksmatrix, egal ob linke untere oder rechte obere?

Vorwärts- und Rückwärtselimination benötigen je

$$\frac{n^2}{2} + O(n)$$

Operationen

2. Was ist die Ausgabe von

```
for i in reversed(range(2,4)):
    print (i)
```

Es werden die Zahlen `3,2` ausgegeben.

3. Was ist der Aufwand zur Berechnung der euklidischen Norm eines Vektors $\mathbb{R}^n$

$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}}$$

Es müssen n Quadrate und $n-1$ Summen gebildet werden. Es folgt eine Wurzel. Der Aufwand ist somit $O(n)$ oder genauer

$$n + O(1),$$

wenn man Summe und Produkt (Quadrat) zusammen als eine arithmetische Operation zählt.

4. Man beschreibe eine Frobenius-Matrix

Eine linke untere Dreiecksmatrix mit 1en auf der Diagonale und von Null verschiedenen Einträgen nur in einer Spalte unterhalb der Diagonalen.

Aufgabe 6.1:

Man beweise *Satz 7.26* (*Frobenius-Matrix*).

Hinweis: Beweisen ist ein großes Wort. Es geht hier um das Nachrechnen. Man schreibe die zu untersuchenden Produkte komponentenweise, d.h. z.B. $(AB)_{ij} = \sum_k A_{ik}B_{kj}$ *und nutze dann jeweils das Wissen über die Matrizen aus, d.h. das viele Einträge entweder Null oder 1 sind.*

Wir nutzen für 2 Matrizen jeweils den Zusammenhang

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}. \tag{1}$$

Jetzt sei $F = F^{(s)}$ eine Frobeniusmatrix mit Einträgen in der s-ten Spalte, d.h.

$$F_{ii} = 1, \quad F_{ij} = 0 \quad \forall i < j, \quad F_{ij} = 0 \quad \forall i \neq s > j$$

(i) Wir zeigen zunächst das die Inversie eine Forbeniusmatrix ist mit umgekehrtem Vorzeichen in den Unter-Diagonalelementen. Für die Inverse gilt

$$F^{-1}F = I,$$

also mit (1) gilt für i, j mit dem Kronecker-Symbol

$$\delta_{ij} = I_{ij} = (F^{-1}F)_{ij} = (FF^{-1})_{ij} = \sum_{k=1}^{n} F^{-1}_{ik}F_{kj} = \sum_{k=1}^{n} F_{ik}F^{-1}_{kj}$$

Durch Ausnutzen, dass F Frobenius-Matrix ist, können die Summen verkürzt werden (Dreiecks-Gestalt mit 1en)

$$\delta_{ij} = F^{-1}_{ij} + \sum_{k=j+1}^{n} F^{-1}_{ik}F_{kj} = F^{-1}_{ij} + \sum_{k=1}^{i-1} F_{ik}F^{-1}_{kj}. \tag{2}$$

Aus $j \neq s$ folgt dann (hier ist $F_{kj} = 0$ für $k \neq j$)

$$\delta_{ij} = F^{-1}_{ij} \quad \forall j \neq s$$

F^{-1} kann also höchstens in der s-ten Spalte Einträge außer der Diagonalen haben. Für $s = j$ gilt mit dem 2ten Teil der Gelichung (2)

$$\delta_{is} = F^{-1}_{is} + \sum_{k=1}^{i-1} F_{ik}F^{-1}_{ks}$$

Im Fall $i = s$ folgt hiermit $F^{-1}_{ss} = 0$. Für $i < s$ verschwindet die Summe, da dann $F_{ik} = 0$ (da F ja nur in s-ter Spalte Einträge hat). Also für $i > s$ reduziert sich die Summe zu

$$\delta_{is} = F^{-1}_{is} + F_{is}F^{-1}_{ss} \quad \Rightarrow \quad 0 = F^{-1}_{is} + F_{is},$$

woraus genau $F_{is}^{-1} = -F_{is}$ folgt.
(*ii*) Jetzt sei G eine zweite Frobeniusmatrix mit Einträge in Spalte r mit $r < s$. Dann ist wieder

$$(GF)_{ij} = \sum_{k=1}^{n} G_{ik}F_{kj} = \sum_{k=j}^{i} G_{ik}F_{kj},$$

wobei die Summe verkürzt wurde wegen der Dreiecksgestalt von G und F. Es folgt $(GF)_{ij} = 0$ für alle $j > i$, d.h. GF hat wieder linke untere Dreiecksgestalt. Für $i = j$ gilt

$$(GF)_{ii} = G_{ii}F_{ii} = 1,$$

d.h. die Diagonalen sind alle eins. Es bleiben die Einträge unterhalb der Diagonalen. Nun sei also $i > j$

$$(GF)_{ij} = \sum_{k=j}^{i} G_{ik}F_{kj} = G_{ij} + \sum_{k=j+1}^{i-1} G_{ik}F_{kj} + F_{ij}$$

Für $r = j$ und $s \neq j$ folgt $F_{ij} = 0$ und $G_{ij} = 0$ aber auch $F_{kj} = 0$ innerhalb der Summe, somit

$$(GF)_{ij} = 0.$$

Für $r = j$ ist $F_{ij} = 0$ und $F_{kj} = 0$ aber es bleibt

$$(GF)_{ir} = G_{ir}$$

Entsprechend ist für $r < s = j < i$ dann $G_{ij} = 0$ aber auch $G_{ik} = 0$ für alle $k > j + 1 > r$ also (**hier benötigt man** $r < s$**!**)

$$(GF)_{is} = G_{is}$$

Programmieraufgabe 6.2:

Wir betrachten die Gauss-Elimination gemäß

```
1  def gauss_elimination(A,b):
2      n = len(A)    #
3      assert len(b)==n,'A und b passen nicht!'
4
5      for i in range(n-1):
6          for j in range(i+1,n):
7              s=A[j,i]/A[i,i]
8              for k in range(i+1,n):
9                  A[j,k]=A[j,k]-s*A[i,k]
10             b[j]=b[j]-s*b[i]
11             A[j,i]=0
12     for i in reversed(range(n)):
13         b[i]=b[i]/A[i,i]
14         A[i,i]=1.0
15         for j in range(i):
16             b[j]=b[j]-b[i]*A[j,i]
17             A[j,i]=0.0
18     return b
```

a) Man verwende den numpy-slice Operator zur Beschleunigung von Zeilen 8-9 sowie 15-17. Für die angegebenen Testmatrix stelle man die Laufzeit beider Methoden (ohne und mit Beschleunigung) doppelt-logarithmisch graphisch dar. Hierbei sollen beide Zeitreihen in einem plot gezeigt werden.

b) Man erstelle eine Funktion zum Rückwärtseinsetzen. Als Testmatrix verwende man einfach die Matrizen aus Aufgabe **a**) und setze alle Einträge links unterhalb der Diagonalen auf Null. Man vergleiche wieder eine einfache und eine mit numpy-slices beschleunigte Variante. Hier kommt es nicht auf die Lösung eines LGS an. D.h., wir können die Rückwärtselimination zu einem beliebigem Vektor bei beliebiger Matrix durchführen. Es geht nur um die Messung des Aufwandes.

c) Man stelle die erreichte Beschleunigung in für beide Algorithmen, Gauss-Elimination und Rückwärtseinsetzen einfach-logarithmisch dar. Hierbei soll die x-Achse logarithmisch gewählt werden.

Hinweis: alles hierfür Notwendige ist Teil des auf der Homepage verlinkten Python-Kurses in den Abschnitten `vectors.ipynb` *sowie* `plot.ipnyb`. *Darüber hinaus finden Sie Algorithmen zum Rückwärtseinsetzen und auch Tipps zur Beschleunigung mit slices im Skript.*