Dirk Praetorius
Christoph Überhuber

Wintersemester 2006/07
10. November 2006

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 4

Die Aufgaben mit Stern (*) sind bis zur nächsten Übung vorzubereiten und werden dort abgeprüft. Die übrigen Aufgaben dienen nur Ihrer Übung und mir als zusätzliche Grundlage für den Prüfungsstoff in den schriftlichen Tests. Kopieren Sie bitte den Source-Code in ein Unterverzeichnis `serie04` *Ihres Home-Verzeichnisses. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit dem* `gcc` *compiliert werden können. In den folgenden Aufgaben sollen im wesentlichen* ***dynamische Arrays*** *sowie* ***Zählschleifen*** *geübt werden. Außer Verzweigungen und elementarer Arithmetik sind keine weiteren Elemente von C nötig.*

Aufgabe 29*. Schreiben Sie eine Bibliothek, die für dynamische Vektoren $x \in \mathbb{R}^n$ und Matrizen $A \in \mathbb{R}^{m \times n}$ die folgenden Funktionen beinhaltet: `mallocvector`, `freevector`, `reallocvector`, `mallocmatrix`, `freematrix` und `reallocmatrix`. Ferner sollen Dateifunktion `loadvector` und `loadmatrix` enthalten sein, die dynamische Vektoren und Matrizen aus einer Datei einlesen. An arithmetischen Funktionen soll die Bibliothek Funktionen `matrixmatrix` und `matrixvector` zur Matrix-Matrix- und zur Matrix-Vektor-Multiplikation enthalten. Speichern Sie den Source-Code `bib.c` und das Header-File `bib.h` ins Verzeichnis `serie04`.

Aufgabe 30*. Eine Matrix $L \in \mathbb{R}^{n \times n}$, deren Einträge oberhalb der Hauptdiagonale gleich 0 sind, bezeichnet man als untere Dreiecksmatrix,

$$L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \dots & \dots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

bzw. mathematisch formuliert: $\ell_{jk} = 0$ für $j < k$. Schreiben Sie eine Funktion `loadmatrixL`, die eine dynamische Matrix aus einer Datei einliest, überprüft, ob diese Matrix eine untere Dreiecksmatrix ist, und in diesem Fall L zurückgibt. Anderenfalls werde `NULL` zurückgegeben. Binden Sie bei der Implementierung die Funktionen aus Aufgabe 29 mittels `#include "bib.c"` ein. Speichern Sie den Source-Code der Aufgaben 30–32 in `bibL.c` ins Verzeichnis `serie04`. Testen Sie Ihren Code, indem Sie ein zusätzliches Hauptprogramm schreiben und Beispieldateien anlegen.

Aufgabe 31*. Man schreibe eine Funktion `matrixvectorL`, die das Matrix-Vektor-Produkt $y = Lx$ berechnet. Dabei soll auf die offensichtlichen Nulleinträge von L nicht zugegriffen werden, um den Rechenaufwand gering zu halten. Speichern Sie den Source-Code in `matrixvectorL.c` ins Verzeichnis `serie04`. Testen Sie Ihren Code, indem Sie ein zusätzliches Hauptprogramm schreiben, das die Matrix L und den Vektor x aus Dateien einliest und das Ergebnis y mit dem Ergebnis der Funktion `matrixvector` aus Aufgabe 29 vergleicht.

Aufgabe 32*. Gegeben sei eine untere Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ mit $\ell_{jj} \neq 0$ für alle $j = 1, \dots, n$. Zu gegebenem $y \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Lx = y$. Man schreibe eine Funktion `solveL`, die x berechnet. Dabei soll auf die offensichtlichen Nulleinträge von L nicht zugegriffen werden, um den Rechenaufwand gering zu halten. — Um den Algorithmus herzuleiten, schreibe man das Matrix-Vektor-Produkt $y = Lx$ komponentenweise für y_j mit $j = 1, \dots, n$ als Summe hin. Man überlege, wie die spezielle Gestalt von L die Laufindizes der Summe vereinfacht und löse diese Gleichung nach x_j auf. Testen Sie Ihren Code, indem Sie ein zusätzliches Hauptprogramm schreiben, das die Matrix L und den Vektor y aus Dateien einliest und mit dem berechneten Ergebnis x das Matrix-Vektor-Produkt $\widetilde{y} = Lx$ berechnet. Bei korrekter Implementierung sollte $y \approx \widetilde{y}$ gelten.

Aufgabe 33. Das Produkt $L = AB$ zweier unterer Dreiecksmatrizen $A, B \in \mathbb{R}^{n \times n}$ ist wieder eine untere Dreiecksmatrix. Man beweise diese Aussage zunächst mathematisch, indem man sich die Formel für das Matrix-Matrix-Produkt hinschreibe und mittels der Voraussetzung an A und B die Indizes vereinfache. Danach schreibe

man eine Funktion `matrixmatrixL`, die die Produktmatrix berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von L, d.h. L_{jk} für $j \geq k$, berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. man verwende die anfangs hergeleitete Formel.

Aufgabe 34. Eine Matrix $U \in \mathbb{R}^{n\times n}$, deren Einträge unterhalb der Hauptdiagonale gleich 0 sind, bezeichnet man als obere Dreiecksmatrix,

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix}$$

bzw. mathematisch formuliert: $u_{jk} = 0$ für $j > k$. Man schreibe eine Funktion `matrixvectorU`, die das Matrix-Vektor-Produkt $y = Ux$ berechnet. Dabei soll auf die offensichtlichen Nulleinträge von U nicht zugegriffen werden, um den Rechenaufwand gering zu halten.

Aufgabe 35. Gegeben sei eine obere Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$ mit $u_{jj} \neq 0$ für alle $j = 1, \dots, n$. Zu gegebenem $y \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Ux = y$. Man schreibe eine Funktion `solveU`, die x berechnet. Dabei soll auf die offensichtlichen Nulleinträge von U nicht zugegriffen werden, um den Rechenaufwand gering zu halten.

Aufgabe 36. Das Produkt $U = AB$ zweier oberer Dreiecksmatrizen $A, B \in \mathbb{R}^{n\times n}$ ist wieder eine obere Dreiecksmatrix. Man beweise diese Aussage zunächst mathematisch, indem man sich die Formel für das Matrix-Matrix-Produkt hinschreibe und mittels der Voraussetzung an A und B die Indizes vereinfache. Danach schreibe man eine Funktion `matrixmatrixU`, die die Produktmatrix berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von U, d.h. U_{jk} für $j \leq k$, berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. man verwende die anfangs hergeleitete Formel.

Aufgabe 37. Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \dots, n, \quad k = i, \dots, n,$$

$$\ell_{ki} = \frac{1}{u_{ii}} \left(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \right) \quad \text{für } i = 1, \dots, n, \quad k = i+1, \dots, n,$$

$$\ell_{ii} = 1 \quad \text{für } i = 1, \dots, n,$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null. Man schreibe eine Funktion `computeLU`, die die LU-Zerlegung von A berechnet und zurückgibt. Dazu überlege man, in welcher Reihenfolge man die Einträge von L und U berechnen muss, damit die angegebenen Formeln wohldefiniert sind (d.h. alles was benötigt wird, ist bereits zuvor berechnet worden).

Aufgabe 38. Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$, und es sei vorausgesetzt, dass das lineare Gleichungssystem $Ax = b$ für jede rechte Seite $b \in \mathbb{R}^n$ eine eindeutige Lösung besitzt. Wir nehmen ferner an, dass die LU-Zerlegung von A existiert (vgl. Aufgabe 37). Man überlege sich einen Algorithmus, der für gegebenes $b \in \mathbb{R}^n$ die Lösung von $Ax = b$ berechnet. Dazu gehe man von der LU-Zerlegung $A = LU$ aus. Man realisiere diesen Algorithmus in Form einer Funktion `solve`.