Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Sommersemester 2009
22. April 2009

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Die Aufgaben mit Stern (*) sind bis zur Übung in der kommenden Woche vorzubereiten und werden dort abgeprüft. Kopieren Sie bitte die Source-Codes auf Ihren Account auf der `lva.student.tuwien.ac.at` *in ein Unterverzeichnis* `serie06`*. Überprüfen Sie vor der Übung, ob Ihre Source-Codes mit dem* `gcc` *compiliert werden können.*

In dieser Übungsserie sollen zur Speicherung die geeigneten Strukturen aus den entsprechenden Aufgaben verwendet werden.

Aufgabe 51*. Man schreibe eine Struktur `vector` zur Speicherung von `double`-Vektoren der Länge n. Die Struktur enthalte neben der Dimension n den dynamischen Datenvektor. Ferner schreibe man die zugehörigen Funktionen `newVector`, `delVector`, `getVectorLength`, `getVectorEntry`, `setVectorEntry`. Speichern Sie den Source-Code unter `vektor.c` in das Verzeichnis `serie06`.

Aufgabe 52*. Man schreibe eine Struktur `matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `0`) auch untere (Typ `'L'`) und obere (Typ `'U'`) Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & \ddots & & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix} \qquad L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \vdots & & \ddots & & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

als obere bzw. untere Dreiecksmatrix. Mathematisch formuliert, gilt also $u_{jk} = 0$ für $j > k$ bzw. $\ell_{jk} = 0$ für $j < k$. Eine vollbesetzte Matrix werde im Fortran-Format spaltenweise als dynamischer Vektor der Länge $n \cdot n$ gespeichert. Dreiecksmatrizen sollen in einem Vektor der Länge $\sum_{j=1}^{n} j = n(n+1)/2$ gespeichert werden. Man schreibe die Funktionen, um mit dieser Struktur arbeiten zu können (`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Speichern Sie den Source-Code unter `matrix.c` in das Verzeichnis `serie06`.

Aufgabe 53*. Man schreibe eine Funktion `matrixvektor` zur Berechnung des Matrix-Vektor-Produkts $Ax \in \mathbb{R}^n$, wobei die Matrix $A \in \mathbb{R}^{n \times n}$ und der Vektor $x \in \mathbb{R}^n$ in den Datenstrukturen aus Aufgabe 51–52 gespeichert seien. Schreiben Sie die Funktion möglichst effizient, d.h. eventuelle Struktur (Dreiecksmatrix!) von A soll ausgenutzt werden. Speichern Sie den Source-Code unter `matrixvector.c` in das Verzeichnis `serie06`.

Aufgabe 54*. Man schreibe eine Struktur `cvector`, zur Speicherung von Vektoren mit komplexwertigen Koeffizienten. Benutzen Sie hierzu den Strukturdatentyp `cdouble` aus Aufgabe 43. Ferner schreibe man die zugehörigen Funktionen `newCVector`, `delCVector`, `getCVectorLength`, `getCVectorEntry`, `setCVectorEntry`. Speichern Sie den Source-Coode unter `cvector.c` in das verzeichnis `serie06`.

Aufgabe 55. Schreiben Sie eine Funktion `cvectorvector`, die das Skalarprodukt $x \cdot y := \sum_{j=1}^{n} x_j \overline{y_j}$ zweier komplexwertiger Vektoren $x, y \in \mathbb{C}^n$ berechnet. Schreiben Sie ferner ein aufrufendes Hauptprogramm, indem die Vektoren x, y eingelesen und der Wert $x \cdot y \in \mathbb{C}$ ausgegeben werden. Speichern Sie den Source-Code unter `cvectorvector.c` in das Verzeichnis `serie06`.

Aufgabe 56. Man schreibe eine Funktion `loadmatrix`, die eine Matrix $A \in \mathbb{R}^{n\times n}$ aus einer Datei einliest. Zur Speicherung verwende man die Struktur aus Aufgabe **??**. Die Funktion soll während des Lesens selbständig erkennen, ob die Matrix obere oder untere Dreiecksstruktur hat. Nach dem Einlesen soll die Matrix dann gegebenenfalls komprimiert als Dreiecksmatrix (um-) gespeichert werden.

Aufgabe 57. 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 58. Man schreibe eine Funktion `solve`, die die Lösung x des linearen Gleichungssystem $Ax = b$ berechnet. Dabei soll gegebenenfalls die Struktur des Gleichungssystems (Dreiecksmatrix!) ausgenutzt werden. Ist A keine Dreiecksmatrix, berechne man die LU-Zerlegung $A = LU$ gemäß Aufgabe 57 und löse die Faktorisierung $LUx = b$ in zwei Schritten: (1) $Ly = b$, (2) $Ux = y$.

Aufgabe 59. Eine Variante zur Berechnung einer Nullstelle einer Funktion $f : [a, b] \to \mathbb{R}$ ist das *Newton-Verfahren.* Ausgehend von einem Startwert x_0 definiert man induktiv eine Folge (x_n) durch

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Man realisiere das Newton-Verfahren in einer Funktion `newton`, wobei die Iteration abgebrochen wird, falls entweder

$$|f'(x_n)| \le \tau$$

oder

$$|f(x_n)| \le \tau \quad \text{und} \quad |x_n - x_{n-1}| \le \begin{cases} \tau & \text{für } |x_n| \le \tau, \\ \tau |x_n| & \text{sonst} \end{cases}$$

gilt. Im ersten Fall gebe man zusätzlich eine Warnung aus, dass das numerische Ergebnis vermutlich falsch ist. Die Funktion soll mit einer beliebigen reellwertigen funktion `double f(double x)` arbeiten, die als Parameter übergeben wird. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x_0 eingelesen und x_n für verschiedene vorprogrammierte Funktionen f ausgegeben wird.

Aufgabe 60. Schreiben Sie eine Funktion `sqrt2_`, die für gegebene $x > 0$ und $\tau > 0$ die Wurzel $\sqrt{x}$ mit Hilfe des Bisektionsverfahrens aus Aufgabe 38 approximiert, sodass der approximative Wert a die Fehlerschranke $|a - \sqrt{x}| \le \tau$ erfüllt: Wählen Sie hierzu die Funktion $f(x) = x^2 - 2$. Wie wählt man das Startintervall $[a, b]$? Vergleichen Sie die Anzahl der Iterationen mit der Anzahl der Iterationen, die das Newton-verfahren aus Aufgabe 59 für dieselbe Aufgabe benötigt und geben Sie jeweils tabellarisch die Fehler $|\sqrt{2} - x_n|$ für beide Verfahren aus.