Dirk Praetorius
Christoph Überhuber

Sommersemester 07
06. Juni 2007

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

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 `serie11` *Ihres Home-Verzeichnisses. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit* `matlab` *auf der* `cad.zserv.tuwien.ac.at` *interpretiert werden können.*

Aufgabe 101*. Gegeben Sei ein sortierter Vektor x der Länge n. Man schreibe eine Funktion `findbisection(x,y)`, die einen Index i zurückgibt, für den $x_i = y$ gilt. Falls y nicht in x vorkommt, werde 0 zurückgegeben. Naive Realisierung führt auf Aufwand $\mathcal{O}(n)$ im worst case, d.h. man durchsucht den Vektor von vorne nach hinten und das gesuchte y steht entweder an der letzten Stelle in x bzw. kommt überhaupt nicht in x vor. Wie kann man den Bisektionsalgorithmus aus Aufgabe 81 geeignet modifizieren, um einen Algorithmus mit worst case Aufwand $\mathcal{O}(\log(n))$ zu erhalten?

Aufgabe 102*. Gegeben sei eine reguläre obere Dreiecksmatrix $U \in \mathbb{K}^{n\times n}$ in Blockform

$$U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}$$

mit $U_{11} \in \mathbb{K}^{n/2\times n/2}$ für n gerade und $U_{11} \in \mathbb{K}^{(n+1)/2\times(n+1)/2}$ für n ungerade. Überlegen Sie sich, dass dann U_{11} und U_{22} ebenfalls reguläre obere Dreiecksmatrizen sind. Insbesondere gilt dann

$$U^{-1} = \begin{pmatrix} U_{11}^{-1} & -U_{11}^{-1}U_{12}U_{22}^{-1} \\ 0 & U_{22}^{-1} \end{pmatrix},$$

da Multiplikation dieser Matrix mit U auf die Einheitsmatrix führt. Folglich lässt sich U^{-1} wie folgt rekursiv berechnen, denn U_{11} und U_{22} haben dieselben Eigenschaften wie U:

- Berechne U_{11}^{-1}.
- Berechne U_{22}^{-1}.
- Berechne $-U_{11}^{-1}U_{12}U_{22}^{-1}$.

Schreiben Sie eine rekursive Funktion `Uinv = blockinvU(U)`, die U^{-1} auf die genannte Weise berechnet. Der Backslash-operator und der Befehl `inv` dürfen nur zur Verifikation verwendet werden.

Aufgabe 103*. Die sogenannte *Power-Iteration* approximiert (unter gewissen Voraussetzungen) den betragsgrößten Eigenwert $\lambda \in \mathbb{R}$ einer symmetrischen Matrix $A \in \mathbb{R}^{n\times n}$ sowie einen dazugehörigen Eigenvektor $x \in \mathbb{R}^n$. Dazu wählt man einen Startvektor $x^{(0)} \in \mathbb{R}^n\backslash\{0\}$, z.B. $x^{(0)} = (1,\dots,1) \in \mathbb{R}^n$. Dann definiert man induktiv für $k \in \mathbb{N}$ die Folgen

$$x^{(k)} := \frac{Ax^{(k-1)}}{\|Ax^{(k-1)}\|_2} \quad \text{und} \quad \lambda_k := x^{(k)} \cdot Ax^{(k)} := \sum_{j=1}^{n} x_j^{(k)} (Ax^{(k)})_j,$$

wobei $\|y\|_2 := \left(\sum_{j=1}^n y_j^2\right)^{1/2}$ die euklidische Norm bezeichne. Dann konvergiert die Folge (λ_k) gegen λ, und $(x^{(k)})$ konvergiert gegen einen Eigenvektor zu λ. Schreiben Sie eine Funktion `poweriteration`, die eine Matrix A, eine Toleranz $\tau > 0$ und optional auch einen Startvektor $x^{(0)}$ übernimmt, dann A auf Symmetrie überprüft und ggf. mit Fehlermeldung abbricht und schließlich die Folgen λ_k und $(x^{(k)})$ berechnet, bis gilt

$$\|Ax^{(k)} - \lambda_k x^{(k)}\|_2 \le \tau \quad \text{sowie} \quad |\lambda_{k-1} - \lambda_k| \le \begin{cases} \tau & \text{für } |\lambda_k| \le \tau, \\ \tau\,|\lambda_k| & \text{anderenfalls.} \end{cases}$$

Die Funktion liefere in diesem Fall λ_k und x_k zurück. Realisieren Sie die Funktion möglichst rechenökonomisch, d.h. vermeiden Sie unnötige Berechnungen (insb. von Matrix-Vektor-Produkten), indem Sie Ergebnisse ggf. zwischenspeichern. Sie können Ihre Funktion mit Hilfe der MATLAB-Funktion `eig` verifizieren. Verwenden Sie die Funktion `norm` sowie Matlab-Arithmetik, soweit wie möglich.

Aufgabe 104*. Implementieren Sie den Quicksort-Algorithmus, um einen Vektor $x \in \mathbb{R}^n$ zu sortieren: Quicksort wählt willkürlich ein Pivotelement aus der zu sortierenden Liste x, z.B. x_1. Dann zerlegt man die Liste in zwei Teillisten: $x^{(<)}$ enthält alle Elemente $< x_1$, $x^{(\geq)}$ enthält alle Elemente $\geq x_1$. Diese Teillisten werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Es besteht (in der Reihenfolge) aus der unteren Liste $x^{(<)}$, dem Pivotelement und der oberen Liste $x^{(\geq)}$. — Eine direkte Implementation dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, geht man wie folgt vor: Beginnend mit $j = 2$ sucht man ein Element $x_j \geq x_1$, d.h. x_j gehört zu $x^{(\geq)}$. Ferner sucht man beginnend bei $k = n$ ein Element $x_k < x_1$, d.h. x_k gehört zu $x^{(<)}$. In diesem Fall vertauscht man x_j und x_k. Wenn sich die Zähler j und k treffen, liegt die Liste x bereits in der Form $(x^{(<)}, x_1, x^{(\geq)})$ vor. Es müssen nun nur noch die Teillisten sortiert werden.

Aufgabe 105. Man schreibe eine rekursive Funktion `detlaplace`, die die Determinante $\det(A)$ einer Matrix $A \in \mathbb{R}^{n\times n}$ mit Hilfe des Laplaceschen Entwicklungssatzes berechnet. Sie können Ihre Funktion mit der Matlab-Funktion `det` verifizieren.

Aufgabe 106. Alternativ kann man die Determinante einer Matrix $A \in \mathbb{R}^{n\times n}$ über die normalisierte LU-Zerlegung aus Aufgabe 59 berechnen. Es gilt nämlich $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Schreiben Sie eine Funktion `detLU`, die die Determinante einer Matrix A mittels der normalisierten LU-Zerlegung berechnet. Sie können Ihre Funktion mit der Matlab-Funktion `det` verifizieren. Die Berechnung der LU-Zerlegung können sie in Matlab mittels `lu` überprüfen.

Aufgabe 107. Man schreibe eine Funktion `merge`, die zwei aufsteigend sortierte Felder a und b so vereinigt, dass das resultierende Feld c ebenfalls aufsteigend sortiert ist, z.B. soll also Aufruf mit $a = (1, 3, 3, 4, 7)$ und $b = (1, 2, 3, 8)$ als Ergebnis $c = (1, 1, 2, 3, 3, 3, 4, 7, 8)$ liefern.

Aufgabe 108. Man schreibe eine rekursive Funktion `mergesort`, die ein Feld a aufsteigend sortiert und das sortierte Feld zurückgibt:

- Hat a Länge ≤ 2, so wird das Feld a explizit sortiert.
- Hat a Länge > 2, halbiere man a in zwei Teilfelder a_1 und a_2, rufe `mergesort` für a_1 und a_2 auf und vereinige die sortierten Teilfelder mittels `merge` aus Aufgabe 107.

Was ist der Vorteil von *Mergesort* gegenüber *Bubblesort*?

Aufgabe 109. Das Integral $\int_a^b f\,dx$ einer stetigen Funktion f wird üblicherweise durch sogenannte Quadraturformeln berechnet. Die *Trapezregel* $I_1(f, a, b)$ sowie die *Simpson-Regel* $I_2(f, a, b)$ sind beispielsweise durch

$$I_1(f,a,b) := \frac{b-a}{2}\big(f(a)+f(b)\big) \quad \text{und} \quad I_2(f,a,b) := \frac{b-a}{6}\Big(f(a)+4\,f\big(\frac{a+b}{2}\big)+f(b)\Big)$$

gegeben. Schreiben Sie eine rekursive Funktion `adquad`, die das Integral $\int_a^b f\,dx$ mittels Simpson-Regel $I_2(f, a, b)$ und iterierter Intervallhalbierung

$$\int_a^b f\,dx = \int_a^{(a+b)/2} f\,dx + \int_{(a+b)/2}^b f\,dx \approx I_2(f,a,(a+b)/2) + I_2(f,(a+b)/2,b)$$

berechnet. Dabei soll der Integrationsbereich jeweils halbiert werden, falls

$$|I_1(f,a,b) - I_2(f,a,b)| > \varepsilon\,\max\{|I_1(f,a,b)|, |I_2(f,a,b)|\} \quad \text{und} \quad |b-a| > h_{\min}$$

ist. Der Funktion sollen neben f, a und b auch die Parameter ε und $h_{\min}$ übergeben werden. Sie können Ihren Code mit Hilfe der Matlab-Funktion `quad` verifizieren.

Aufgabe 110. Man schreibe eine (rekursive) Funktion `papierschnitt`, die alle Möglichkeiten visualisiert, wie ein Papierbogen der ganzzahligen Länge `laenge` in Papierbahnen der Länge 1 und 2 geschnitten werden kann. — D.h. man stelle eine natürliche Zahl $n =$ `laenge` auf alle möglichen Weisen als Summe $n = \sum_{j=1}^{k} \sigma_j$ mit Summanden $\sigma_j \in \{1, 2\}$ dar. Dabei soll die Reihenfolge beachtet werden. Für `laenge=4` gibt es beispielsweise 5 Möglichkeiten:

- $4 = 2 + 2$
- $4 = 2 + 1 + 1$
- $4 = 1 + 2 + 1$
- $4 = 1 + 1 + 2$
- $4 = 1 + 1 + 1 + 1$