Dirk Praetorius,
Michele Ruggeri

Sommersemester 2017
25.04.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Schreiben Sie eine Funktion `double* merge(double* a, int m, double* b, int n)`, die zwei aufsteigend sortierte Vektoren $a \in \mathbb{R}^m$ und $b \in \mathbb{R}^n$ so vereinigt, dass der resultierende Vektor $c \in \mathbb{R}^{m+n}$ ebenfalls aufsteigend sortiert ist, z.B. soll $a = (1, 3, 3, 4, 7)$ und $b = (1, 2, 3, 8)$ als Ergebnis $c = (1, 1, 2, 3, 3, 3, 4, 7, 8)$ liefern. Dabei soll ausgenutzt werden, dass die Vektoren a und b bereits sortiert sind. Schreiben Sie ein aufrufendes Hauptprogramm, in dem $m, n \in \mathbb{N}$ sowie $a \in \mathbb{R}^m$ und $b \in \mathbb{R}^n$ eingelesen werden und $c \in \mathbb{R}^{m+n}$ ausgegeben wird. Testen Sie ihr Programm mit entsprechenden Beispielen! Speichern Sie den Source-Code unter `merge.m` in das Verzeichnis `serie06`.

Aufgabe 6.2. Schreiben Sie eine rekursive Funktion `void mergesort(double* x, int n)`, die einen Vektor $x \in \mathbb{R}^n$ mittels des *Mergesort*-Algorithmus aufsteigend sortiert. Gehen Sie dabei nach folgender Strategie vor:

- Ist die Länge $n \leq 2$, so wird der Vektor $x \in \mathbb{R}^n$ explizit sortiert.
- Ist die Länge $n > 2$, halbiert man x in zwei Teilvektoren y und z. Man ruft rekursiv `mergesort` für y und z auf und vereinigt die beiden sortierten Teilvektoren wieder zu einem sortierten Vektor.

Schreiben Sie darüber hinaus ein Hauptprogramm, in dem Sie den Vektor x und die Länge n einlesen, `mergesort` aufrufen und das Ergebnis ausgeben. Testen Sie das Programm entsprechend! Wie groß ist der Aufwand ihrer Funktion? Speichern Sie den Source-Code unter `mergesort.c` in das Verzeichnis `serie06`.

Aufgabe 6.3. Schreiben Sie eine rekursive Funktion `void quicksort(double* x, int n)`, die einen Vektor $x \in \mathbb{R}^n$ mittels des *Quicksort*-Algorithmus aufsteigend sortiert. Man wählt willkürlich ein Pivotelement aus der zu sortierenden Liste x, z.B. x_1. Dann zerlegt man die Liste in zwei Teillisten $x^{(<)}$ und $x^{(\geq)}$ und das Pivotelement x_1: $x^{(<)}$ enthält dabei alle Elemente $< x_1$, $x^{(\geq)}$ enthält nur Elemente $\geq x_1$. $x^{(<)}$ und $x^{(\geq)}$ werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Eine direkte Implementierung dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, gehen Sie nun folgendermaßen 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 in der Form $(x_1, x^{(<)}, x^{(\geq)})$ vor. Mit einer weiteren Vertauschung erreicht man sofort die Gestalt $(x^{(<)}, x_1, x^{(\geq)})$. Es müssen nur noch $x^{(<)}$ und $x^{(\geq)}$ rekursiv sortiert werden. Schreiben Sie darüberhinaus ein Hauptprogramm, in dem Sie den Vektor x und und die Länge n einlesen und `quicksort` aufrufen. Testen Sie Ihren Code entsprechend! Wie groß ist der Aufwand ihrer Funktion? Speichern Sie den Source-Code unter `quicksort.c` in das Verzeichnis `serie06`.

Aufgabe 6.4. Schreiben Sie eine Funktion `void unique(double* x, int* n)`, die einen Vektor $x \in \mathbb{R}^n$ aufsteigend sortiert, doppelte Einträge streicht und den Vektor in gekürzter Form zurückgibt. Die Funktion soll also beispielsweise den Vektor $x = (4, 3, 5, 1, 4, 3, 4) \in \mathbb{R}^7$ durch den Vektor $x = (1, 3, 4, 5) \in \mathbb{R}^4$ überschreiben. Der Vektor x soll mit dem gekürzten Vektor überschrieben werden d.h. Vektorlänge und dynamisches Koeffizientenfeld müssen ggf. angepasst werden! Schreiben Sie ein aufrufendes Hauptprogramm, in dem die Vektorlänge n und der Vektor $x \in \mathbb{R}^n$ eingelesen werden und das Ergebnis der Funktion `unique` ausgegeben wird. Testen Sie Ihre Implementierung mit passenden Beispielen! Speichern Sie den Source-Code unter `unique.c` in das Verzeichnis `serie06`.

Aufgabe 6.5. Schreiben Sie eine Funktion `void cut(double* x, int* n, double cmin, double cmax)`, die aus einem Vektor $x \in \mathbb{R}^n$ alle Einträge x_j mit $x_j < c_{\min}$ oder $x_j > c_{\max}$ ($c_{\min}, c_{\max} \in \mathbb{R}$) streicht. Für $c_{\min} = 0$ und $c_{\max} = 10$ soll die Funktion also beispielsweise den Vektor $x = (-4, 3, -5, 1, 7, 3, 11, -1) \in \mathbb{R}^8$ durch den Vektor $x = (3, 1, 7, 3) \in \mathbb{R}^4$ überschreiben. Der Vektor x soll mit dem gekürzten Vektor

überschrieben werden d.h. Vektorlänge und dynamisches Koeffizientenfeld müssen ggf. angepasst werden! Schreiben Sie ein aufrufendes Hauptprogramm, in dem die Vektorlänge n und der Vektor $x \in \mathbb{R}^n$ sowie die Schranken $c_{\min}, c_{\max}$ eingelesen werden und das Ergebnis der Funktion `cut` ausgegeben wird. Testen Sie Ihre Implementierung mit passenden Beispielen! Speichern Sie den Source-Code unter `cut.c` in das Verzeichnis `serie06`.

Aufgabe 6.6. 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 $(x_n)_{n\in\mathbb{N}}$ durch

$$x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1}) \quad \text{für } n \geq 1.$$

Man realisiere das Newton-Verfahren in einer Funktion `double newton(double x0, double tau)`, die die Approximation x_n der Nullstelle zurückgibt, wobei die Iteration abgebrochen wird, falls entweder

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

oder

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

gilt. Im ersten Fall gebe man zusätzlich eine Warnung aus, dass das numerische Ergebnis vermutlich falsch ist. Stellen Sie mittels `assert` sicher, dass $\tau > 0$ gilt. Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` und Ableitung `double fstrich(double x)` arbeiten. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x_0 und τ eingelesen werden und x_n ausgegeben wird. Wie und mit welchen Beispielen können Sie Ihren Code auf Korrektheit testen? Speichern Sie den Source-Code unter `newton.c` in das Verzeichnis `serie06`.

Aufgabe 6.7. Das Δ^2-Verfahren von Aitken ist ein Verfahren zur Konvergenzbeschleunigung von Folgen. Für eine injektive Folge $(x_n)_{n\in\mathbb{N}}$ mit $\lim_{n\to\infty} x_n = x$ definiert man

$$y_n := x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}.$$

Unter gewissen zusätzlichen Voraussetzungen an die Folge $(x_n)_{n\in\mathbb{N}}$ gilt dann

$$\lim_{n\to\infty} \frac{y_n - x}{x_n - x} = 0,$$

d.h. die Folge $(y_n)_{n\in\mathbb{N}}$ konvergiert schneller gegen x als $(x_n)_{n\in\mathbb{N}}$. Schreiben Sie eine Funktion `double* aitken(double* x, int n)`, die für einen Vektor $x \in \mathbb{R}^n$ mit Länge $n \geq 3$ den Vektor $y \in \mathbb{R}^{n-2}$ berechnet. Stellen Sie mittels `assert` sicher, dass $n \geq 3$ gilt. Testen Sie Ihre Implementierung mit passenden Beispielen. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem der Vektor x und die Länge n eingelesen werden und der Vektor y ausgegeben wird. Was passiert für die Folge $x_n = q^n$ mit $0 < q < 1$? Speichern Sie den Source-Code unter `aitken.c` in das Verzeichnis `serie06`.

Aufgabe 6.8. Für eine konvergente Folge $\{x_n\}_{n\geq 1}$ mit Grenzwert $z \in \mathbb{R}$ spricht man von Konvergenzordnung $p \geq 1$, falls es eine Konstante $c > 0$ gibt mit $|x_{n+1} - z| \leq c\,|x_n - z|^p$ für alle $n \geq 1$. Die Konvergenzordnung lässt sich als Grenzwert der Folge $\{p_n\}_{n\geq 1}$ mit

$$p_n = \frac{\log|x_{n+2} - z| - \log|x_{n+1} - z|}{\log|x_{n+1} - z| - \log|x_n - z|} \quad \text{für } n \geq 1$$

approximieren. Woher kommt diese Formel? Leiten Sie diese Gleichung her! (*Hinweis:* Verwenden Sie den Ansatz $|x_{n+1} - z| = c|x_n - z|^p$). Schreiben Sie eine Funktion `double* convorder1(double* x, int n, double z)`, die für einen gegebenen Vektor $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ und einen Grenzwert $z \in \mathbb{R}$ den Vektor $p \in \mathbb{R}^{n-2}$ berechnet und zurückgibt. Stellen Sie mittels `assert` sicher, dass $n \geq 3$ gilt. Üblicherweise ist der Grenzwert z unbekannt und man hat nur einen Vektor $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ von Approximationen. In diesem Fall kann man die Funktion `convorder` mit der Teilfolge $(x_1, \ldots, x_{n-1})$ und dem Grenzwert $z = x_n$ verwenden. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem der

Vektor x und die Länge n (sowie der Grenzwert z, wenn bekannt) eingelesen werden und der Vektor p ausgegeben wird. Testen Sie Ihre Implementierung mit passenden Beispielen! Testen Sie Ihre Implementierung indem Sie die experimentelle Konvergenzrate des Newton-Verfahrens aus Aufgabe 6.6 berechnen. Hierzu modifizieren Sie die Funtion `newton` so, dass nicht nur die approximative Nullstelle, sondern die gesamte Folge $(x_0, \ldots, x_n)$ von Approximationen zurückgegeben wird. Speichern Sie den Source-Code unter `convorder.c` in das Verzeichnis `serie06`.