Dirk Praetorius,
Michele Ruggeri

Sommersemester 2017
04.04.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 5

Aufgabe 5.1. Schreiben Sie eine Funktion `maxcompare`, die für zwei gegebene Vektoren $a, b \in \mathbb{R}^n$ zählt, wie oft das Maximum von a und b mit der Bezeichnung $M := \max\{a_i, b_i \,|\, i = 1, \dots, n\}$ im Vektor a und b an der gleichen Stelle vorkommt. Zum Beispiel soll die Funktion für die Vektoren $a = (1.1, 4, 2e-4, 4, 3, 4, -1.5)$ und $b = (2.2, 4, 4, 2e-5, 4, -1, 2.7, 4)$ den Wert 2 zurückgeben, da das Maximum $M = 4$ in beiden Vektoren an zwei Stellen nämlich $a_2 = b_2 = a_5 = b_5 = M$, gleichermaßen vorkommt. Wenn etwa M nur entweder in a oder b vorkommt, dann soll die Funktion klarerweise 0 zurückgeben. Die Länge $n \in \mathbb{N}$ soll eine Konstante im Hauptprogramm sein, die Funktion `maxcompare` ist aber für beliebige Längen zu programmieren. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem $a, b \in \mathbb{R}^n$ eingelesen werden und `maxcompare` aufgerufen wird. Speichern Sie den Source-Code unter `maxcompare.c` in das Verzeichnis `serie05`.

Aufgabe 5.2. *Bubblesort* ist ein Sortier-Algorithmus: Man vergleicht aufsteigend jedes Element eines Arrays x_j mit seinem Nachfolger x_{j+1} und – falls notwendig – vertauscht die beiden. Nach dem ersten Durchlauf muss zumindest das letzte Element bereits am richtigen Platz sein. Der nächste Durchlauf muss also nur noch bis zur vorletzten Stelle gehen, usw. Wie viele geschachtelte Schleifen braucht dieses Vorgehen? Schreiben Sie eine Funktion `bubblesort`, die ein gegebenes Array $x \in \mathbb{R}^n$ mittels Bubble-Sort aufsteigend sortiert, d.h. $x_1 \le x_2 \le \cdots \le x_n$. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor x einliest und in sortierter Reihenfolge ausgibt. Die Länge des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `bubblesort` ist für beliebige Länge n zu programmieren. Bestimmen Sie den Aufwand Ihrer Funktion. Speichern Sie den Source-Code unter `bubblesort.c` in das Verzeichnis `serie05`.

Aufgabe 5.3. Schreiben Sie eine Funktion `wurzelschranke`, die zu einer gegebenen Zahl $x \ge 0$ die natürliche Zahl $k \in \mathbb{N}_0$ mit $k \le \sqrt{x} < k + 1$ zurückgibt. Dabei dürfen weder die Wurzel-Funktion `sqrt`, noch Rundungsoperationen (z.B. `floor` oder `ceil` etc.) verwendet werden. Schreiben Sie ferner ein aufrufendes Hauptprogramm das $x \in \mathbb{R}$ einliest und $k \in \mathbb{N}$ ausgibt. Speichern Sie den Source-Code unter `wurzelschranke.c` in das Verzeichnis `serie05`.

Aufgabe 5.4. Für eine Matrix $A \in \mathbb{R}^{m \times n}$ ist die Zeilensummennorm durch

$$\|A\| = \max_{j=1,\dots,m} \sum_{k=1}^{n} |A_{jk}|$$

gegeben. Schreiben Sie eine Funktion `zeilensummennorm`, die die Zeilensummennorm einer spaltenweise gespeicherten Matrix A berechnet und zurückgibt. Schreiben Sie ein aufrufendes Hauptprogramm, in dem A eingelesen und $\|A\|$ ausgegeben wird. Welchen Aufwand hat Ihre Funktion? Falls die Funktion für $n = m = 10^4$ eine Laufzeit von 0.1 Sekunden hat, welche Laufzeit erwarten Sie für $n = m = 3 \cdot 10^5$? Speichern Sie den Source-Code unter `zeilensummennorm.c` in das Verzeichnis `serie05`.

Aufgabe 5.5. Schreiben Sie eine Funktion `eratosthenes`, die das *Sieb des Eratosthenes* realisiert. Dies ist ein Algorithmus, mit dem alle Primzahlen bis zu einer bestimmten Zahl `nmax` berechnet werden können (benannt nach dem griechischen Mathematiker Eratosthenes). Der Algorithmus sieht folgendermaßen aus:

- Man legt eine Liste (Vektor) $\texttt{prim} = (2, \dots, \texttt{nmax}) \in \mathbb{N}^{\texttt{nmax}-1}$ an.
- Man streicht aus der Liste alle Vielfachen der ersten Zahl (also der Zahl 2).
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfachen.

Am Ende sollen alle Primzahlen von $2, \ldots,$`nmax` ausgegeben werden. Geben Sie außerdem die Anzahl der gefundenen Primzahlen mit aus. Realisieren Sie das Streichen, indem Sie die entsprechenden Einträge auf 0 setzen. Die Zahl `nmax` soll eine Konstante im Hauptprogramm sein. Speichern Sie den Source-Code unter `eratosthenes.c` in das Verzeichnis `serie05`.

Aufgabe 5.6. Die Quotientenfolge $(a_{n+1}/a_n)_{n\in\mathbb{N}}$ zur Fibonacci-Folge $(a_n)_{n\in\mathbb{N}}$,

$$a_0 := 1, \quad a_1 := 1, \quad a_n := a_{n-1} + a_{n-2} \quad \text{für } n \geq 2,$$

konvergiert gegen den goldenen Schnitt $(1+\sqrt{5})/2$. Insbesondere konvergiert die Differenz

$$b_n := \frac{a_{n+1}}{a_n} - \frac{a_n}{a_{n-1}}$$

gegen Null. Schreiben Sie eine *nicht-rekursive* Funktion `cauchy`, die zu gegebenem $k \in \mathbb{N}$ die kleinste Zahl $n \in \mathbb{N}$ mit $|b_n| \leq 1/k$ zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Zahl $k \in \mathbb{N}$ einliest und den zugehörigen Index $n \in \mathbb{N}$ ausgibt. Speichern Sie den Source-Code unter `goldenerSchnitt.c` in das Verzeichnis `serie05`.

Aufgabe 5.7. Schreiben Sie eine *nicht-rekursive* Funktion `power`, die für gegebene reelle Zahlen $x > 1$ und $C > 0$ die kleinste Zahl $n \in \mathbb{N}$ berechnet mit $x^n > C$. Dabei soll die Funktion `log` nicht verwendet werden. Stellen weiters Sie mittels `assert` sicher, dass $x > 1$ und $C > 0$ gilt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem x und C eingelesen werden und n ausgegeben wird. Speichern Sie den Source-Code unter `power.c` in das Verzeichnis `serie05`.

Aufgabe 5.8. Alternativ zum Bisektionsverfahren aus der Vorlesung kann eine Nullstelle von $f : [a,b] \to \mathbb{R}$ auch mit dem *Sekantenverfahren* berechnet werden. Dabei sind x_0 und x_1 gegebene Startwerte und man definiert induktiv x_{n+1} als Nullstelle der Geraden durch $(x_{n-1}, f(x_{n-1}))$ und $(x_n, f(x_n))$, d.h.

$$x_{n+1} := x_n - f(x_n)\,\frac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)}$$

Schreiben Sie eine Funktion `sekante(x0,x1,tau)` die die Folge der Iterierten berechnet, bis entweder

$$|f(x_n) - f(x_{n-1})| \leq \tau$$

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. Es werde dann x_n als Approximation einer Nullstelle z_0 von f zurückgegeben. 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)` arbeiten. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x_0 und x_1 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 `sekante.c` in das Verzeichnis `serie05`.