Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
23./24.11.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 5

Aufgabe 5.1. Gegeben sei eine endliche Zahlenfolge x (dynamisches Array vom Typ `int`) und eine Schranke $n \in \mathbb{Z}$. Schreiben Sie eine Funktion `y = cut(x,n)`, die aus x alle Glieder $x(j)$ mit $x(j) \geq n$ streicht, d.h. y ist ein gekürztes (ggf. reallokiertes) x. Ferner schreibe man ein aufrufendes Hauptprogramm, in dem der Vektor x sowie die Schranke n eingelesen werden. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `cut.c` in das Verzeichnis `serie05`.

Aufgabe 5.2. Schreiben Sie eine Funktion `double* dec2bin(int N, int* n)`, die zu einer natürlichen Zahl $0 \leq N < 65535$ die Binärdarstellung berechnet und zurückgibt. Es sollen die Koeffizienten $a_i \in \{0, 1\}$ für $i = 0, \dots, n$ ermittelt werden, sodass $N = \sum_{i=0}^{n-1} a_i 2^i$ gilt ($n \leq 16$). Die Funktion liefere die Binärdarstellung von N ohne führende Nullen. Beachten Sie, dass auch die Länge des Vektors „zurückgegeben“ werden muss. Beispielsweise gebe die Funktion für $N = 77$ den Vektor `1 0 0 1 1 0 1` zurück. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem N eingelesen und die entsprechende Binärdarstellung ausgegeben wird. Speichern Sie den Source-Code unter `dec2bin.c` in das Verzeichnis `serie05`.

Aufgabe 5.3. 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 (*fct)(double), double (*fctprime)(double), 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 $f : [a, b] \to \mathbb{R}$ und die dazugehörige Ableitung $f' : [a, b] \to \mathbb{R}$ sollen mittels Funktionspointer an die Funktion `newton` übergeben werden. 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? Was ist der Zusammenhang zwischen dem Newton-Verfahren und der Aufgabe 4.8 der letzten Serie? Speichern Sie den Source-Code unter `newton.c` in das Verzeichnis `serie05`.

Aufgabe 5.4. Für eine differenzierbare Funktion $f : [a, b] \to \mathbb{R}$ kann man die Ableitung $f'(x)$ in einem festen Punkt $x \in \mathbb{R}$ durch den einseitigen Differenzenquotienten

$$\Phi(h) := \frac{f(x + h) - f(x)}{h} \quad \text{für } h > 0$$

approximieren. Schreiben Sie eine Funktion `double* diff(double (*fct)(double), double x, double h0, double tau, int* n)`, die für $h_n := 2^{-n} h_0$ die Folge der $\Phi(h_n)$ berechnet, bis gilt

$$|\Phi(h_n) - \Phi(h_{n+1})| \leq \begin{cases} \tau & \text{falls } |\Phi(h_n)| \leq \tau, \text{oder} \\ \tau \, |\Phi(h_n)| & \text{anderenfalls.} \end{cases}$$

Die Funktion liefere in diesem Fall die vollständige Folge $(\Phi(h_0), \ldots, \Phi(h_n))$ der Iterierten zurück. Beachten Sie, dass auch die Länge des Vektors „zurückgegeben“ werden muss. Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` arbeiten. Schreiben Sie ein Main-Programm, in dem Sie Ihre Funktion eingehend testen. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `diff.c` in das Verzeichnis `serie05`.

Aufgabe 5.5. 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 `serie05`.

Aufgabe 5.6. Man kombiniere das Aitken-Verfahren aus Aufgabe 5.5 mit dem einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 5.4. Mit $h_n := 2^{-n}h_0$ betrachten wir die Folge der $x_n := \Phi(h_n)$ und erhalten daraus die Folge (y_n). Man schreibe eine Funktion `double diffaitken(double (*fct)(double), double x, double h0, double tau)`, die die Funktion f, den Auswertungspunkt x, die Schrittweite $h_0 > 0$ sowie die Toleranz $\tau > 0$ übernimmt und $y_{n+1} \approx f'(x)$ zurückliefert, sobald gilt

$$|y_n - y_{n+1}| \leq \begin{cases} \tau, & \text{falls } |y_{n+1}| \leq \tau, \\ \tau\,|y_{n+1}|, & \text{anderenfalls.} \end{cases}$$

Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` arbeiten. In jedem Schritt gebe man h_{n+1}, $|y_{n+1} - y_n|$ sowie y_{n+1} aus. Vergleichen Sie die Anzahl der Iterationen mit und ohne (d.h. $y_n = x_n$) Aitken-Verfahren. Wie und mit welchen Funktionen haben Sie Ihren Code getestet? Speichern Sie den Source-Code unter `diffaitken.c` in das Verzeichnis `serie05`.

Aufgabe 5.7. 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. Dabei soll explizit ausgenutzt werden, dass die Teilvektoren sortiert sind.

Schreiben Sie darüber hinaus ein Hauptprogramm, in dem Sie den Vektor x und die Länge n einlesen, `mergesort` aufrufen und das Ergebnis ausgeben. Wie haben Sie Ihren Code auf Korrektheit getestet? Wie groß ist der Aufwand ihrer Funktion? Speichern Sie den Source-Code unter `mergesort.c` in das Verzeichnis `serie05`.

Aufgabe 5.8. Man realisiere Mergesort aus Aufgabe 5.7, ohne im Rekursionsschritt Vektoren der halben Länge anzulegen, sondern verwende Pointer-Arithmetik: Ist `x` der Basepointer auf das Array x (d.h. der Pointer auf x_0), so ist `x+k` der Basepointer auf x_k. Man muss im Rekursionsschritt also lediglich den Basepointer auf x_0, den Startindex k und den Endindex ℓ eines Teilfeldes von x übergeben. Für das sortierte Ergebnisfeld ist *einmalig* am Anfang ein dynamisches Array anzulegen. Weiterer Hilfsspeicher ist *nicht* nötig. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `mergesort2.c` in das Verzeichnis `serie05`.