Dirk Praetorius,
Gregor Gantner

Wintersemester 2018/19
06./07.12.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Erklären Sie den Zusammenhang zwischen Arrays und Pointern. Das folgende Programm soll jeweils Adresse und Wert der Einträge eines Arrays ausgeben. Welche der folgenden Programmzeilen macht das richtige und warum (warum nicht)?

```
#include <stdio.h>

int main(){
  int vec[5] = {1, 3, 5, 7, 9};
  int i = 0;

  for(i=0; i<5; i++){
    printf("0) %p: %d\n",    vec[i],  (vec+i));
    printf("1) %p: %d\n",   (vec+i),   vec[i]);
    printf("2) %p: %d\n",   &vec[i], *(vec+i));
    printf("3) %p: %d\n\n", &vec[i],  *vec+i);
  }
}
```

Aufgabe 6.2. 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 (*f)(double), double x, double h0, double tau)`, die für $h_n := 2^{-n} h_0$ ($n \in \mathbb{N}$) die Folge der $\Phi(h_n)$ berechnet, bis gilt

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

Die Funktion liefere in diesem Fall $\Phi(h_n)$ als Approximation von $f'(x)$ zurück. Stellen Sie mittels `assert` sicher, dass $\tau, h_0 > 0$ gilt. Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` arbeiten. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem x, h_0 und τ eingelesen werden und $\Phi(h_n)$ ausgegeben wird. Wie und mit welchen Beispielen können Sie Ihren Code auf Korrektheit testen? Speichern Sie den Source-Code unter `diff.c` in das Verzeichnis `serie06`.

Aufgabe 6.3. 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 6.4. Man kombiniere das Aitken-Verfahren aus Aufgabe 6.3 mit dem einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 6.2. 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 `serie06`.

Aufgabe 6.5. 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.c` in das Verzeichnis `serie06`.

Aufgabe 6.6. 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? Speichern Sie den Source-Code unter `mergesort.c` in das Verzeichnis `serie06`.
Bonus: Wie groß ist der Aufwand ihrer Funktion?

Aufgabe 6.7. Zu gegebenen reellen Stützstellen $x_1 < \cdots < x_n$ und Funktionswerten $y_j \in \mathbb{R}$ garantiert die Lineare Algebra ein eindeutiges Polynom $p(t) = \sum_{j=1}^{n} a_j t^{j-1}$ vom Grad $n-1$ mit $p(x_j) = y_j$ für alle $j = 1, \ldots, n$. Nun sei $t \in \mathbb{R}$ fixiert und $p(t)$ gesucht. Man kann $p(t)$ mit dem *Neville-Verfahren* berechnen, ohne zunächst den Koeffizientenvektor $a \in \mathbb{R}^n$ berechnen zu müssen: Dazu definiere man für $j, m \in \mathbb{N}$ mit $m \geq 2$ und $j + m \leq n + 1$ die Werte

$$\begin{aligned} p_{j,1} &:= y_j, \\ p_{j,m} &:= \frac{(t - x_j)p_{j+1,m-1} - (t - x_{j+m-1})p_{j,m-1}}{x_{j+m-1} - x_j}. \end{aligned}$$

Es gilt dann $p(t) = p_{1,n}$. Schreiben Sie eine Funktion `neville`, die den Auswertungspunkt $t \in \mathbb{R}$ sowie die Vektoren $x, y \in \mathbb{R}^n$ übernimmt und $p(t)$ mittels Neville-Verfahren berechnet. Dazu berücksichtige

man das folgende schematische Vorgehen

$$
\begin{array}{lcccccccccccc}
y_1 & = & p_{1,1} & \longrightarrow & p_{1,2} & \longrightarrow & p_{1,3} & \longrightarrow & \dots & \longrightarrow & p_{1,n} & = & p(t) \\
 & & & \nearrow & & \nearrow & & & & \nearrow & & & \\
y_2 & = & p_{2,1} & \longrightarrow & p_{2,2} & & & & & & & & \\
 & & & \nearrow & & & & \nearrow & & & & & \\
y_3 & = & p_{3,1} & \longrightarrow & \vdots & & & & & & & & \\
\vdots & & \vdots & & \vdots & \nearrow & & & & & & & \\
y_{n-1} & = & p_{n-1,1} & \longrightarrow & p_{n-1,2} & & & & & & & & \\
 & & & \nearrow & & & & & & & & & \\
y_n & = & p_{n,1} & & & & & & & & & &
\end{array}
$$

Der mathematische Beweis für diesen Algorithmus folgt in der Vorlesung zur Numerischen Mathematik. Zunächst schreibe man die Funktion so, dass die Matrix $(p_{j,m})_{j,m=1}^{n}$ vollständig aufgebaut wird. Speichern Sie den Source-Code unter `neville.c` in das Verzeichnis `serie06`. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 6.8. Man kann das Neville-Verfahren aus Aufgabe 6.7 so programmieren, dass zur Speicherung der Werte *keine* Matrix $(p_{j,m})_{j,m=1}^{n}$ aufgebaut wird, sondern die gegebenen y_j-Werte geeignet überschrieben werden. Dadurch wird kein weiterer Speicher benötigt. Man realisiere dieses Vorgehen in einer Funktion `neville2`. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `neville2.c` in das Verzeichnis `serie06`.