Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Mario Ohlberger
M.Sc.-Math. Julia Brunken

20. September 2016

Übung zum Kompaktkurs

Einführung in die Programmierung zur Numerik mit Python

Sommersemesterferien 2016 — Blatt 2

Aufgabe 1 (Newton-Verfahren)

Implementieren Sie das Newton-Verfahren zur Nullstellensuche einer Funktion $f : \mathbb{R} \to \mathbb{R}$. Schreiben Sie dazu eine Funktion `newton`, welche als Argumente die Funktion f, ihre Ableitung f', den Startpunkt x_0, die Toleranz $\varepsilon > 0$ und eine maximale Iterationszahl $N_{\max}$ erwartet. `newton` soll eine Näherung einer Nullstelle x_* mit $f(x_*) = 0$ zurückgeben. Implementieren Sie die beiden Funktionen

$$f_1(x) = x^2 + 2x - 15$$
$$f_2(x) = x^3 - 33x^2 + 263x - 231$$

sowie deren Ableitungen innerhalb Ihres Skriptes und testen Sie das Newton-Verfahren für f_1, f_2 mit diversen Startwerten x_0.

Hinweise: *Das Newton-Verfahren besteht darin, iterativ eine Folge von Näherungen x_n nach der Vorschrift*

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

zu bestimmen, bis entweder $|f(x_{n+1})| < \varepsilon$ oder $n = N_{\max}$. Implementieren Sie die Betragsfunktion ebenfalls in Ihrem Skript!

Aufgabe 2 (Fibonacci-Zahlen)

Setzen Sie Aufgabe 2 von Blatt 1 nun mithilfe rekursiver Funktionsaufrufe um. Schreiben Sie hierzu eine Funktion $fibonacci(n)$, die eine eingegebene, natürliche Zahl n erhält und dann rekursiv die n-te Fibonacci-Zahl nach dem Schema

$$fibonacci(n) = fibonacci(n-1) + fibonacci(n-2), \quad fibonacci(0) = fibonacci(1) = 1$$

berechnet. Speichern Sie die Ergebnisse schrittweise in einer Liste und geben Sie diese aus. Vergleichen und bewerten Sie beide Herangehensweisen. Wie lässt sich der rekursive Ansatz verbessern?

Aufgabe 3 (Sortieren)

Schreiben Sie zwei Funktionen, die jeweils eine unsortierte Liste aus Zahlen (Integer oder float) erhalten und diese aufsteigend sortieren. Es soll weder das *return* Statement verwendet noch eine neue Liste angelegt werden. Folgende Funktionen sind zu implementieren:

(a) Die Funktion *bubblesort()* zur Anwendung des Bubblesort-Algorithmus.

(b) Die Funktion *quicksort()* zur Anwendung des Quicksort-Algorithmus.

Testen Sie Ihre Implementierung an folgenden Listen:

2,-3,-1,1,0,5,9,7 und Z,B,J,K,A,D,F,E,N,H

Überlegen Sie sich hierzu wie Sie die zweite Liste sortieren können ohne die Sortierfunktionen abzuändern. Informieren Sie sich dazu über die Funktion *ord()*.

Hinweis:

- *Beim Bubblesort-Algorithmus wird die Liste von links nach rechts durchlaufen. In jedem Schritt wird das aktuelle Element mit dem rechten Nachbarn verglichen. Ist der rechte Nachbar kleiner als das aktuelle Element, so werden beide Elemente vertauscht. Am Ende des Durchlaufs steht das größte Element am Ende der Liste. Diese Durchläufe werden solange wiederholt, bis die Liste vollständig sortiert ist - d.h. keine Vertauschungen mehr auftreten. Dabei muss das jeweils letzte Element im vorherigen Durchlauf nicht mehr betrachtet werden, da es größer als die restlichen Elemente der Liste ist.*

- *Der Quicksort-Algorithmus läuft nach dem Prinzip "Divide and Conquer" ab: Zunächst wird ein Pivotelement aus der Liste ausgesucht, dann werden alle Elemente kleiner als das Pivotelement in eine linke Teilliste und alle Elemente größer als das Pivotelement in eine rechte Teilliste aufgeteilt. Elemente gleich dem Pivotelement können beliebig auf die Teillisten aufgeteilt werden. Dann wird der Quicksort Algorithmus auf die beiden Teillisten angewendet, bis die Teilliste Länge eins oder null aufweist und die Rekursion wird abgebrochen.*