Algorithmische Mathematik I

Wintersemester 19/20
Prof. Dr. J. Gedicke
Johannes Rentrop und Jannik Schürg

UNIVERSITÄT BONN

Programmieraufgabenblatt 4.

Abgabedatum: **11.11.2019–15.11.2019**

Programmieraufgabe 1. (Iterative Verfahren)

Gegeben sei die Funktion $f : \mathbb{R} \to \mathbb{R}$, $f(a) = 3\sqrt[3]{a}$. Schreiben Sie ein Programm, das für ein eingegebens a einen Näherungswert $\tilde{f}(a)$ für eine vorgegebene Genauigkeit $\epsilon > 0$ auf zwei verschiedene Weisen berechnet:

a) Betrachten Sie die Umkehrfunktion f^{-1} und implementieren Sie eine binäre Suche. Wählen Sie ein geeignetes Anfangsintervall.

b) Implementieren Sie das Newton-Verfahren und ermitteln Sie damit die Nullstelle einer geeigneten Funktion F. Wählen Sie einen geeigneten Startwert.

Verwenden Sie jeweils als Abbruchkriterium, wenn sich aufeinanderfolgende Näherungswerte der Iteration um weniger als ϵ unterscheiden: $\epsilon_n := |x_n - x_{n-1}| < \epsilon$. Lassen Sie jeweils die Anzahl der benötigten Iterationen ausgeben. Ermitteln Sie jeweils außerdem die experimentelle Konvergenzordnung p_E sowie die experimentell Konstante c_E ab dem vierten Schritt und lassen Sie diese ausgeben. Dafür können Sie folgende Formel verwenden:

$$p_\mathrm{E} = \frac{\log(\epsilon_n/\epsilon_{n-1})}{\log(\epsilon_{n-1}/\epsilon_{n-2})}, \qquad c_\mathrm{E} = \frac{\epsilon_n}{\epsilon_{n-1}^{p_\mathrm{E}}}.$$

Variieren Sie a, ϵ sowie die Startintervalle/-werte. Decken sich die Ergebnisse mit Ihrer Erwartung?

(5+5 Punkte)

Programmieraufgabe 2. (Fallstricke der Fließkomma-Arithmetik)

Diese Aufgabe enthält Beispiele bei denen Fließkomma-Arithmetik bei unbedarfter Anwendung große Fehler im Ergebnis liefert.

a) Berechnen Sie die Folge a_n von Aufabenblatt 4, indem Sie $a_n = na_{n-1} - 1$ iterativ berechnen bis $n = 25$. Wählen Sie für den Startwert $c = e$. Nutzen Sie einmal ausschließlich `float` und einmal `double` und geben Sie die Ergebnisse aus. Erklären Sie das Ergebnis.

Realitätsbezug: Sagen wir, eine Bank bietet Ihnen an, dass Sie $(e-1)$€ anlegen können und im Jahr $n = 1, 2, \ldots$ nach Geldanlage wird der Kontostand ver-n-facht und danach um eins reduziert. Nehmen Sie das Angebot an?

b) Sei

$$\begin{aligned} f &: \mathbb{R}^2 \to \mathbb{R} \\ (a, b) &\mapsto (333.75 - a^2)b^6 + a^2(11a^2b^2 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}. \end{aligned}$$

Implementieren Sie f einmal ausschließlich mit `float` und einmal mit `double`, indem Sie so vorgehen wie in Algorithmus 2. Berechnen Sie $f(77617, 33096)$ in beiden Versionen und geben Sie das Ergebnis aus.

Hintergrund: Die Zahlen a und b sind so gewählt, dass $f(a,b) = \frac{a}{2b} - 2$, es treten Auslöschungseffekte auf. Wenn alles gut geht, werden Ihre Berechnungen 1.172604 ergeben, dieses Ergebniss wird mit `double` und `__float128` (s. Teil c)) immer mehr Stellen bekommen.

Häufig lässt sich Instabilität dadurch erkennen, dass das sich Ergebnis beim Rechnen mit mehr Präzision stark ändert. Dieses Beispiel zeigt, dass das nicht immer der Fall ist, das korrekte Ergebnis ist ein anderes. Das heißt, *selbst wenn bei höherer Präzision das Ergebnis gleich bleibt, kann es komplett falsch sein.*

c) *(Bonus)* Der Compiler GCC bietet einen weiteren Fließkommatyp, `__float128`. Dieser hat mehr als doppelt so viel Präzision wie `double` (entspricht `binary128` aus IEEE 754-2008). Implementieren Sie Teil a) und b) auch, indem Sie ausschließlich `__float128` verwenden. Sie müssen `-lquadmath` beim Kompillieren ergänzen.

Lesen Sie im Handbuch[1] nach, wie Sie `__float128` ausgeben können.

Eventuell müssen Sie Änderungen vornehmen für ihr Betriebssystem oder Compiler, verwenden Sie ggf. die Linux-Computer im CIP-Pool.

Hinweis: Betrachten Sie folgenden Code-Schnipsel:

```
1  float x = 2.17;
2  float y = (333.75 - x);
```

In Zeile 1 ist `2.17` vom Type `double` und wird zu `float` konvertiert. In Zeile 2 wird `x` zu `double` konvertiert, dann wird `333.75 - x` berechnet und das Ergebnis zurück zu `float` konvertiert. Um nur mit `float` zu rechnen benutzen Sie bspw.

```
float x = 2.17f;
float y = (333.75f - x);
```

Bei der Verwendung von `__float128` reicht es nicht, bspw.

```
const __float128 e = 2.71828182845904523536028747135;
```

zu schreiben. Die Zahl auf der rechten Seite ist vom Typ `double` und wird nicht mit der geschriebenen Präzision verwendet. Benutzen Sie stattdessen

```
const __float128 e = strtoflt128("2.71828182845904523536028747135", NULL);
```

oder die Konstante `M_Eq` aus dem Header `quadmath.h`.

Algorithmus 1 Berechung von f aus Aufgabe 2.

function F(a, b)
  $a_2 \leftarrow a \cdot a$
  $b_2 \leftarrow b \cdot b$
  $b_4 \leftarrow b_2 \cdot b_2$
  $b_6 \leftarrow b_4 \cdot b_2$
  $b_8 \leftarrow b_4 \cdot b_4$
  return $(333.75 - a_2)b6 + a_2(11a_2b_2 - 121b_4 - 2) + 5.5b_8 + \frac{a}{2b}$
end function

(5+5+5* Punkte)

[1] `https://gcc.gnu.org/onlinedocs/libquadmath.pdf`, Kapitel 3.2.