Probeklausur Numerisches Programmieren

Wintersemester 2019/20, MUSTERLÖSUNG

Hendrik Möller

23. Februar 2020

Der Autor übernimmt keine Garantie über die Korrektheit der Aufgaben oder Lösungen.

Des Weiteren ist diese Klausur nur als ein "Vorschlag" von möglichen Aufgaben(-typen) anzusehen, wie sie in der eigentlichen Klausur drankommen könnten. Ein Bezug oder Ähnlichkeit zu vorhandenen Aufgaben sind vollkommen willkürlich.

- Schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!
- Als Hilfsmittel ist einzig und allein ein handschriftlich, beidseitig beschriebenes Blatt DIN A4 mit eigenen Notizen erlaubt (keine Ausdrucke, keine Kopien).
- Die vorgesehene Arbeitszeit beträgt 90 Minuten.
- In dieser Klausur können Sie insgesamt 42 Punkte erreichen. Zum Bestehen werden voraussichtlich 17 Punkte benötigt.
- Die einzelnen Teilaufgaben sind unabhängig voneinander lösbar. Sollten Sie zu einer Teilaufgabe keine Lösung finden, so können Sie diese Teilaufgabe also einfach überspringen und mit der nächsten Teilaufgabe fortfahren.

1 Zahlen & Kondition [3 + 2.5 + 3 = 8.5 Pkt.]

Es soll folgende Darstellung für Gleitkommazahlen untersucht werden:

- Die Basis ist 2,
- es gib kein Vorzeichenbit,
- 3 Bits werden für den Exponenten verwendet. Dieser ist als eine vorzeichenbehaftete Ganzzahl kodiert, das erste der drei Bits steht also für das Vorzeichen des Exponenten,
- für die Mantisse wird ein Bit verwendet. Von einer Normalisierung mit führender Eins wird ausgegangen (diese muss nicht abgespeichert werden).

Sonderfälle werden nicht betrachtet.

(a) Was ist die größte und kleinste Zahl, die mit obigen Format dargestellt werden kann?

Wie lautet die kleinste positive Ganzzahl > 0, die nicht exakt darstellbar ist?

Größte darstellbare: $011|1 = 1,1 \cdot 2^3 = 1100 = 12$

Kleinste darstellbare: $111|0 = 1,0 \cdot 2^{-3} = 2^{-3}$

Maschinengenauigkeit: 2^{-2}

Kleinste positive Ganzzahl: 5

(b)

$$f(x) = (2\exp(x) - 2)/3$$

Untersuchen Sie die Kondition von $f(x)$ für die Stelle $x = 0$.

$$cond(f, x) = \left| \frac{x \cdot \frac{2e^x}{3}}{\frac{2e^x - 2}{3}} \right| = \left| \frac{2x \cdot e^x}{2e^x - 2} \right|$$

$$\lim_{x \to 0} \left| \frac{x \cdot e^x}{e^x - 1} \right| = |x + 1| = 1$$

Ist also gut konditioniert.

(c) Viele Programmiersprachen verwenden denormalisierte Gleitkommazahlen, damit der Zahlenbereich von float vergrößert wird. Für den kleinsten Exponenten wird die Restriktion aufgehoben, dass vor den Mantissenbits eine 1 stehen muss.

Was die kleinste Zahl > 0, die im Standard IEEE Float Format dargestellt werden kann?

Was ist die kleinste Zahl > 0, welche im IEEE Format mit obiger Denormalisierung angegeben werden kann?

Im Standard IEEE Format: 2^{-126}

Mit Denormalisierung: 2^{-149}

2 Interpolation & Quadratur [3 + 2 + 1 + 1 + 1.5 = 8.5 Pkt.]

$$\int_0^2 f(x)dx \text{ mit } f(x) = \frac{(2x+1)}{x}$$

Die Funktion ist bei $x = 0$ nicht definiert, deswegen können manche Standard-Quadraturverfahren mit Punkten an der Intervalluntergrenze nicht verwendet werden. Wir möchten ein neues Verfahren entwickeln mit den Punkten an den Positionen $x_0 = 0.5$ und $x_1 = 2$.

(a) Berechnen Sie das Interpolationspolynom $p(x)$ von f mit den Punkten an den Positionen x_0 und x_1.

$$\begin{aligned} p(0.5) &= 4 \\ p(2) &= \frac{5}{2} \\ p(x) &= \frac{9}{2} - x \end{aligned}$$

(b) Nutzen Sie das Ergebnis aus (a), um mit der dazugehörigen Quadraturregel eine Annäherung für das obige Intergal zu erhalten.

Die gesuchte Regel ist natürlich die Trapezregel, weil wir genau 2 Stützstellen haben.

$$\begin{aligned} p(0) &= \frac{9}{2} \\ p(2) &= \frac{5}{2} \end{aligned}$$

$$\begin{aligned} Q_T(p) &= 2 \cdot \frac{1}{2} \left(\frac{9+5}{2} \right) \\ &= 7 \end{aligned}$$

(c) Zeichnen Sie die berechnete Fläche der Approximation grafisch in folgende Abbildung ein.

Das rot eingezeichnete Trapez entspricht der berechneten Fläche.

Zeichnen Sie die berechnete Fläche der Approximation grafisch in folgende Abbildung ein.

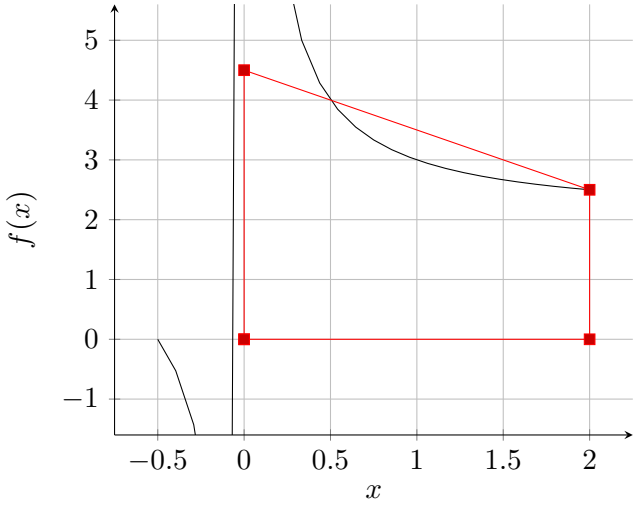

Abbildung 1: Plot der Funktion f

(d) Sie möchten nun die Gauß Quadratur mit der Trapezregel vergleichen. Bis zu welchem Grad können Polynome durch eine Gaußquadratur mit n Stützstellen exakt integriert werden? Begründen Sie Ihre Antwort kurz.

n Stützstellen plus n Gewichte ergibt $2n$ Freiheitsgrade

Also ein LGS mit $2n$ Gleichungen. Exakt lösbar bis $2n$ Unbekannte ergibt Polynom von Grad $2n - 1$.

(e) Ihre Approximation ist Ihnen nicht gut genug und sie wollen die Anzahl der Stützstellen verdoppeln. Was für ein Nachteil hat die Gauß-Qaudratur gegenüber der Trapezregel, wenn wir davon ausgehen, dass die Berechnung der Stützstellen x_i und Gewichte w_i keine Berechnungen benötigt?

Die Gaußquadratur passt dynamisch die Stützstellen an. Wenn man also die Anzahl dieser verändert, ändern sich deren Positionen. Damit muss man also doppelt soviele Funktionsauswertungen machen im Vergleich zur Trapezsumme, da wir dort die der vorherigen (mit weniger Stützstellen) wiederverwenden können.

3 Differentialgleichungen [2 + 3 + 1 + 3 = 9 Pkt.]

$$y'(t) = f(t, y) = \frac{1}{2}(y(t) \cdot t)^2$$

$$y(0) = y_0 = 2$$

(a) Nennen Sie die Berechnungsvorschrift des expliziten Eulerverfahrens mit diskreten Werten y_k und Zeitschrittweite δt für ein Anfangswertproblem.

Berechnen Sie zwei Schritte jenes Verfahrens mit obigen Anfangswertbedingungen und Schrittweite $\delta t = t_{k+1} - t_k = 1$, das Zeitintervall ist also in $[0; 2]$.

$$t_k = t_0 + k \cdot \delta t$$
$$y_{k+1} = y_k + \delta t \cdot f(t_k, y_k)$$

$$y_1 = 2 + 0 = 2$$
$$y_2 = 2 + \frac{1}{2}(2 \cdot 1)^2 = 4$$

(b) Ein Mehrschrittverfahren zweiter Ordnung hat die Vorschrift:

$$y_{k+1} := y_k + \frac{\delta t}{2} \cdot (4 \cdot f(t_k, y_k) - 2 \cdot f(t_{k-1}, y_{k-1}))$$

Führen Sie einen Schritt dieses Verfahrens mit Schrittweite $\delta t = 1$ und Anfangswert $y_0 = 2$ aus. Sie rechnen also y_2 aus, benötigen hierfür noch y_1. Nehmen Sie hierfür einfach ihre Approximation aus Teilaufgabe (a). Falls Sie Teilaufgabe (a) nicht lösen konnten, verwenden Sie $y_1 = 2$.

$$\begin{aligned} y_2 &= 2 + \frac{1}{2} \cdot \left(4 \cdot \frac{1}{2}(2 \cdot 1)^2 - \frac{1}{2}(1 \cdot 0)^2\right) \\ &= 2 + \frac{1}{2} \cdot 2 \cdot 4 \\ &= 2 + 4 = 6 \end{aligned}$$

(c) Wie verändert sich der globale Diskretisierungsfehler für das Mehrschrittverfahren aus Teilaufgabe (b) bei Halbierung der Schrittweite?

Der Fehler viertelt sich! (nicht: er verringet sich um ein Viertel!)

(d) Im Folgenden ist eine Abbildung, bei der eine Funktion $f(x)$ zu sehen ist sowie die verschiedenen Iterationschritte drei verschiedener Verfahren.

Die drei genutzten Verfahren sind der explizite Euler, der implizite Euler sowie das Runge-Kutta Verfahren. Welche Farbe gehört zu welchem Verfahren? Geben Sie eine Begründung ein (Ausschlusskriterium ist hierbei nicht erlaubt)!

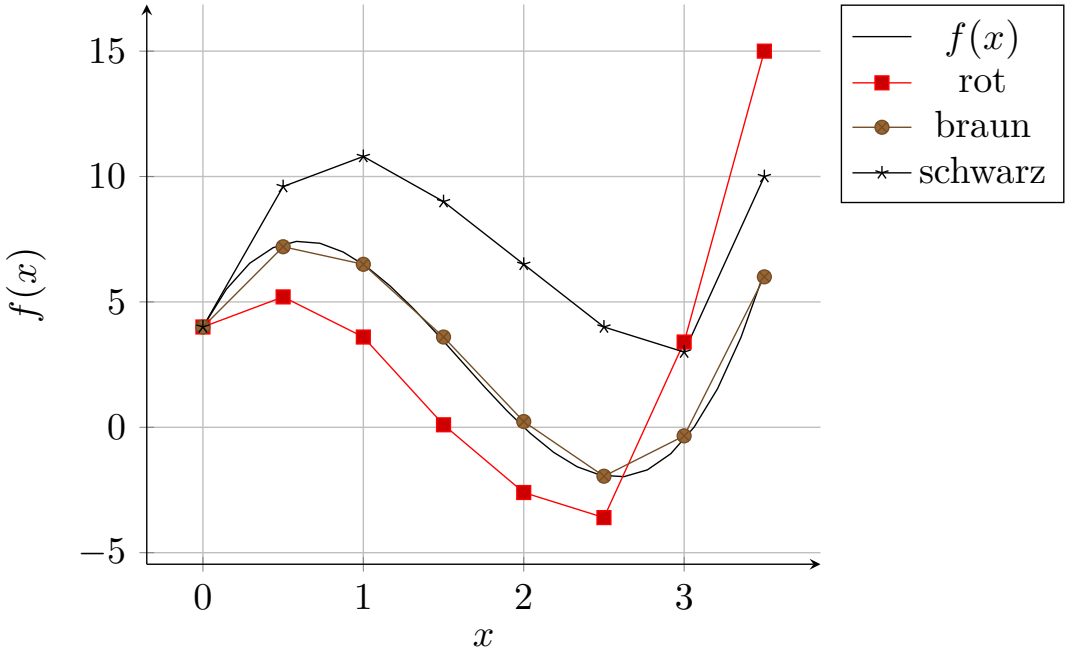

Abbildung 2: Die Lösungen der Iterationen drei verschiedener numerischer Verfahren.

Schwarz gehört zum expliziten Euler Verfahren, da man sieht, dass jeweils die aktuelle Steigung genutzt wurde für eine Iteration.

Rot gehört zum impliziten Euler Verfahren, da man sieht, dass jeweils die Steigung im nächsten Punkt genutzt wurde für eine Iteration. Eine alternative Begründung (die eigentlich dasselbe ist) wäre zu sagen, dass man sieht, dass es dieselben Steigungen wie bei der schwarzen Kurve ist nur jeweils eine Iteration verschoben.

Braun gehört dem Runge-Kutta Verfahren an, da es die höchste Ordnung von den drei Verfahren besitzt und damit die größte Genauigkeit, was sich unschwer erkennen lässt.

Häufige Fehler hier:
Implizites Euler Verfahren ist NICHT das genaueste, es besitzt nur Ordnung 1! Als Begründung zählt außerdem nicht zu sagen, dass Expliziter Euler häufig über dem exakten Wert ist und impliziter Euler darunter, das stimmt nicht einmal in dieser Aufgabe für das gesamte angezeigte Intervall und macht keinen Sinn!

4 Fixpunktiteration [2 + 2 + 3 = 7 Pkt.]

Formel des Newton-Verfahrens zur Nullstellenberechung ist definiert als:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Wir untersuchen:

$$f(x) = \frac{1}{2}x^2 - 2x + 2$$

(a) Sie möchten das Newton-Verfahren für die Nullstellenbestimmung auf die obige Funktion anwenden, Ihnen fällt jedoch auf, dass das Newton Verfahren deutlich langsamer ist als Sie es gewohnt sind. Warum? Zeigen sie es anhand einer Rechnung.

Es ist eine doppelte Nullstelle bei $x = 2$. Dadurch ist es nur noch lineare anstelle von quadratischer Konvergenz.

$$\begin{aligned} f(2) &= 2 - 4 + 2 = 0 \\ f'(x) &= x - 2 \\ f'(2) &= 2 - 2 = 0 \end{aligned}$$

(b) Modifizieren Sie das Newton-Verfahren derart, dass die normale Konvergenzgeschwindigkeit erreicht wird.

Führen Sie dann einen Zeitschritt mit Startwert $x_0 = 1$ durch.

$$\begin{aligned} \Phi(x) &= x - 2\frac{\frac{1}{2}x^2 - 2x + 2}{x - 2} \\ &= x - \frac{x^2 - 4x + 4}{x - 2} \\ &= x - \frac{(x-2)^2}{x - 2} \\ &= x - x + 2 = 2 \\ x_1 &= 2 \end{aligned}$$

(c) Bestimmen Sie in Abbildung 2 graphisch alle Fixpunkte von g und markieren Sie

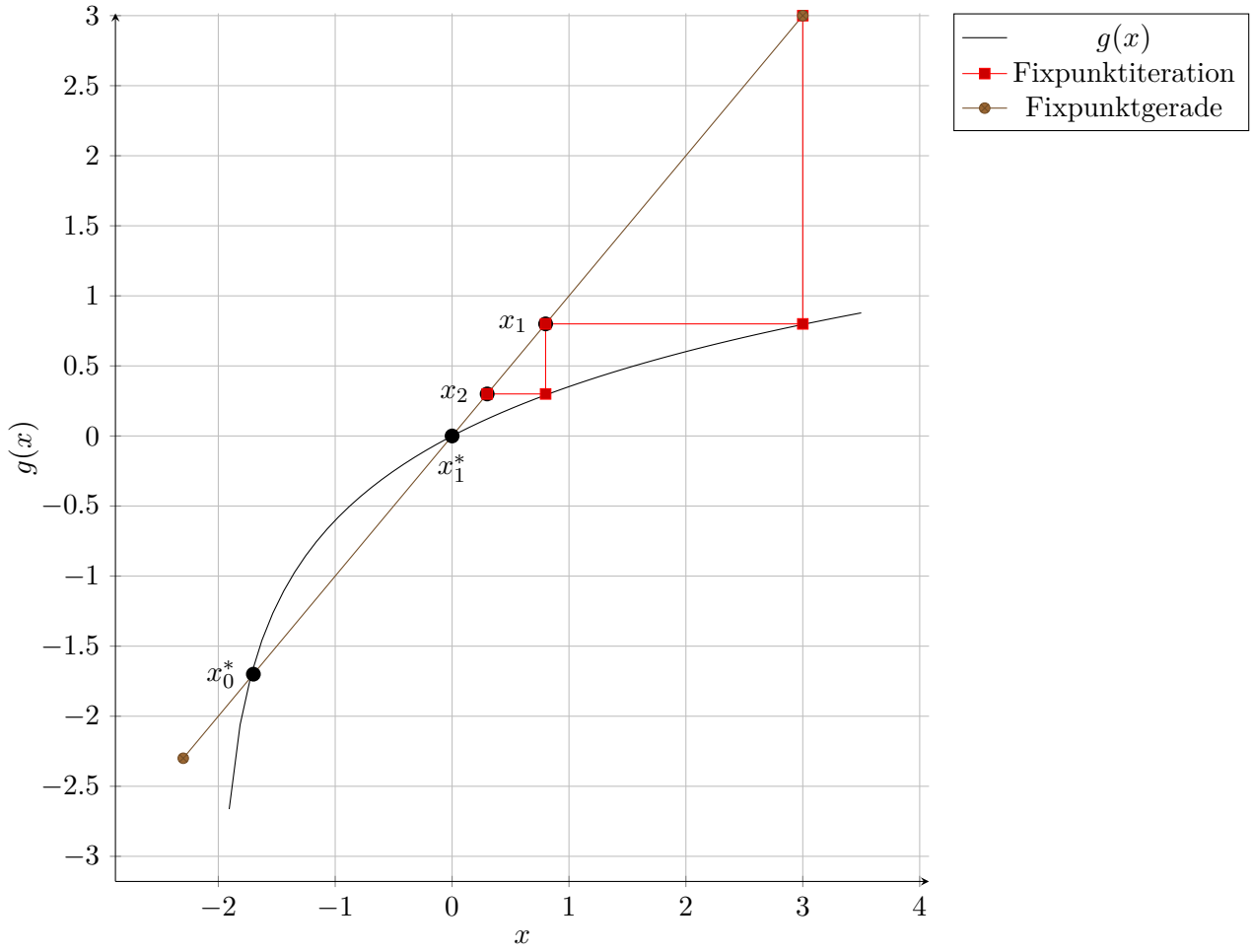

Abbildung 3: Plot der Funktion $g(x)$.

diese.

Führen Sie (graphisch) zwei Schritte der Fixpunkt-Iteration für den Startwert $x_0 = 3$ in die Abbildung ein.

Bei welchen der Fixpunkte handelt es sich um anziehende und bei welchen um abstoßende Fixpunkte? Begründen Sie ihre Antwort anhand der Graphik.

x_1^* ist ein anziehender Fixpunkt, da wir uns mit den zwei Iterationen auf ebenjenen zubewegen. Wenn man die Iteration sich gedanklich mit dem zweiten Punkt überlegt merkt man, dass es sich um einen abstoßenden Fixpunkt handeln muss.

5 Programmieraufgabe [2 + 7 = 9 Pkt.]

(a) Was für Vorteile besitzt das Gauß-Seidel Verfahren gegenüber dem Jacobi Verfahren?

Weniger Speicheraufwand und bessere Konvergenzgeschwindigkeit.

(b) Folgendes Programmgerüst sei gegeben. Vervollständigen Sie dieses ohne unnötige Berechnungen oder Speicherallokationen. Bibliotheksfunkionen dürfen **nicht** verwendet werden.

```
import java.util.Arrays;
public class CoolProgram {

   public static void main(String[] args)
   {
      // parse input (separately implemented)
      double[][] A = parseA(args);
      double[] b = parseB(args);
      double[] x = parseX(args);
      int numIterations = parseN(args);

      // calculate solution with gaussSeidel()


      System.out.println("Solution:\n" + Arrays.toString(x));
   }

   private static double[] gaussSeidel(double[][] A, double[] b, double[] x)
   {

      //TODO: implement one step of the gauss-seidel method
      for (int i = 0; i < x.length; i++) {

         double r = b[i];

         // Mat-Vec Mult.
         for (int j = 0; j < x.length; j++) {
            r -= (A[i][j] * x[j]);
         }

         x[i] += (1 / A[i][i]) * r;
      }
      return x;
   }
}
```