Probeklausur Numerisches Programmieren

Wintersemester 2018, MUSTERLÖSUNG

Hendrik Möller

24. 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 17 Punkte benötigt.
- Hinweis: Nach Auffassung des Autors ist diese Klausur schwerer als die "richtige" Klausur sein wird.
- 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 Gleitkommazahlen [3 + 2 + 3 = 8 Pkt.]

Wir betrachen die fiktive Gleitkommadarstellung G_t, welche 8 Bits zur Verfügung hat. Für den Exponent als vorzeichenbehaftete Ganzzahl haben wir die ersten 4 Bits. Der Mantisse stehen 4 Bits zur Verfügung. Von einer normalisierten Mantisse mit führender Eins wird ausgegangen. Diese muss **nicht** abgespeichert werden. Die Basis sei $B = 2$, durch einen Defekt bekommen wir die 8 Bits aber als 2 Hex-Ziffern angegeben und können auch diese auch nur als solche wieder schreiben. Sonderbelegungen seien keine vorhanden.

Aufgerundet wird auch nach dem Hex-Ziffern und hierbei sobald die nicht darstellbare ≥ 8 ist, ansonsten abgerundet.

Hinweis: Folgende Zahlen Übersetzungen gelten:

Dezimalzahl	G_t Format	G_t Zahl in Binärdarstellung
10	$3\|4$	$0011\|0100$
6.5	$2\|A$	$0010\|1010$
2	$1\|0$	$0001\|0000$
$3/16$	$B\|8$	$1011\|1000$

(a) Wandeln Sie 8.5 in das G_t Format um. Ist die Darstellung eindeutig?

$3|1$, da in Binär wäre es $0011|0001$. Die Darstellung ist eindeutig, da wir eine normalisierte Mantisse haben. Andere Zahlen mit einem Exponenten von 0 sind aber nicht eindeutig, da der Exponent wegen dem Vorzeichen $+0$ und -0 annehmen kann.

(b) Was ist die kleinste Ganzzahl ≥ 0, die mit der G_t Darstellung gerade nicht mehr exakt darstellbar ist?

Was ist mit der betragsmäßig kleinsten rationalen Zahl, die gerade noch exakt darstellbar ist?

0, da die kleinste **Ganzzahl** 1 wäre, die darstellbar ist.

Kleinste darstellbare rationale Zahl ist dann 2^{-7}.

(c) Wir möchten die Zahl 134 in dem G_t Format darstellen, müssen aber feststellen, dass wir dafür runden müssen. Runden wir auf oder ab? Wie groß ist der absolute Fehler?

134 umwandeln wäre $1,000011 \cdot 2^7$, in binärer G_t Darstellung $0111|000011$ und somit in G_t Format $7|0C$.

Es wird aufgerundet auf $7|1$, also $0111|0001$ und damit in dezimal 136. Der absolute Fehler beträgt demnach $|134 - 136| = 2$.

2 Quadratur [3 + 1 + 1.5 = 5.5 Pkt.]

Betrachten wir die Gauß-Quadratur. Sie möchten diese Methode mit $n = 3$ Stützpunkten verwenden. Außerdem soll gelten, dass $x_0 = -0.75$, $x_1 = 0.5$, dessen Gewichtung $w_1 = 1$ ist sowie die Gewichtung des letzten Stützpunktes $w_2 = \frac{2}{3}$.

Als Erinnerung im Folgenden die Formel, auf der die Gauß-Quadratur beruht:

$$\sum_{i=0}^{n-1} f(x_i) \cdot w_i \stackrel{!}{=} \int_a^b f(x)\ \mathrm{d}x$$

(a) Berechnen Sie mithilfe der Gauß-Quadraturformel die fehlenden Informationen x_2 und w_0 aus. Die Integralgrenzen seien wie üblich $[a, b] = [-1, 1]$.

$$\begin{aligned} w_0 + w_1 + w_2 &= 2 \\ w_0 &= 2 - 1 - \frac{2}{3} \\ w_0 &= \frac{1}{3} \end{aligned}$$

$$\begin{aligned} w_0 x_0 + w_1 x_1 + w_2 x_2 &= 0 \\ -\frac{1}{4} + \frac{1}{2} + \frac{2}{3} x_2 &= 0 \\ x_2 &= -\frac{3}{8} \end{aligned}$$

(b) Das Integral

$$\int_{-1}^{1} x^5 + x^2 - 1\ \mathrm{d}x$$

soll mit dem Gauß-Quadratur Verfahren angenähert werden. Was muss gelten, damit die Gauß-Quadratur von dem Fehler her ungefähr die Maschinengenauigkeit erreicht?

Es muss eine Stützstellenanzahl von $n \geq 3$ gelten, damit der erreichbare Grad von $(2 \cdot n) - 1 = 5$ erreicht wird.

(c) Von einer uns unbekannte Funktion $f(x)$ haben wir nur die Stützpunkte an den Stellen $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, $x_4 = 6$ gegeben. Wir möchten die Fläche im Intervall $I : [0, 6]$ auswerten. Diskutieren Sie, ob sich das Verfahren der Romberg-Quadratur hierfür eignet.

Es eignet sich nicht wirklich dafür, da wir mit Trapezsummen exponentieller Stückanzahl arbeiten. Damit können wir aber dann nur genau eine Trapezsumme aufstellen, mit der sich alleine noch nicht die Kombinationsregel anwenden lässt. (Theoretisch würde es sich aber noch relativ gut mit zwei Simpsonregeln ausrechnen lassen.)

3 Interpolation [3 + 3 + 1 + 1 = 8 Pkt.]

Zur Approximation einer Funktion $f(x)$ soll die Spline-Interpolation genutzt werden.

$$\begin{bmatrix} 4 & 1 & & \\ 1 & 4 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{bmatrix} \cdot \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_{n-2} \\ y'_{n-1} \end{pmatrix} = \frac{3}{h} \cdot \begin{bmatrix} y_2 - y_0 \\ y_3 - y_1 \\ \vdots \\ y_{n-1} - y_{n-3} \\ y_n - y_{n-2} \end{bmatrix} - \begin{pmatrix} y'_0 \\ 0 \\ \vdots \\ 0 \\ y'_n \end{pmatrix} \tag{1}$$

(a) Es seien folgende Informationen über die Funktion $f(x)$ gegeben:

i	0	1	2	3	4
x_i	-1	0	1	2	3
y_i	2	4	1	-2	2
y'_i	?	-1	?	-1	11

Berechnen Sie die fehlenden Ableitungen.

$$\begin{pmatrix} -4 + y'_2 \\ -2 + 4y'_2 \\ y'_2 - 4 \end{pmatrix} = \begin{pmatrix} -3 - y'_0 \\ -18 \\ -8 \end{pmatrix}$$

Dann als einzelnen Gleichungen schreiben erhalten wir $y'_0 = 5$ und $y'_2 = -4$.

(b) Stellen Sie für die Intervalle I_0 und I_1 die Transformationsfunktionen auf, damit jeweils auf das Intervall $[1, 3]$ abgebildet wird. Erklären Sie **kurz**, wofür man diese benötigt bzw. warum man diese nutzt.

$I_0 : [0; 4]$

$I_1 : [-2; 4]$

$$t_0 = \frac{x + 2}{2}$$

$$t_1 = \frac{x + 5}{3}$$

Man benötigt sie um auf beliebige Intervalle abbilden zu können. Praktisch ist dies bei der Hermite-Interpolation, da man dann die ausgerechneten kubischen Basispolynome $H_0, \ldots, H_n$ immer wieder verwenden kann.

(c) Wir möchten eine oszillierende Funktion $g(x)$ interpolieren. Wenn es uns nur um Genauigkeit geht, nutzen wir Spline-Interpolation? Begründen Sie ihre Antwort.

Nein nutzen wir nicht, da bei der Spline-Interpolation die Schwingungen (aka Oszillationen) minimiert werden. Die Approximation kann eigentlich nicht gut werden.

(d) Die Formel für die Simpson-Regel ist:

$$Q_{\mathrm{SS}}(f;h) = \frac{h}{3} \cdot (f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{N-2} + 4f_{N-1} + f_N)$$

Formulieren Sie die Simpson-Regel für die vier equidistanten Stützwerte y_0, y_1, y_2 und y_3 und den zwei Randstützstellen x_0 und x_3.

Das ist nicht möglich, da wir mit Simpson-Regel immer eine ungerade Anzahl von Stützstellen benötigen. (Theoretisch könnten wir aber dann eine Simpson-Regel und dann noch eine Trapezregel anwenden).

4 Iterative Verfahren [1 + 2 = 3 Pkt.]

Wir betrachten die Matrix A, welche die Eigenwerte $\lambda_0 = 1$, $\lambda_1 = 4$ und $\lambda_2 = 5$ besitzt.

(a) Bestimmen Sie den Shift μ der Direct Power Iteration, bei welcher diese für die Matrix A nicht konvergiert.

Das ist der Shift, sodass es zwei betragsmäßig größte Eigenwerte gibt. In diesem Fall wäre das mit $\mu = 3$ erfüllt, da dann $\lambda_0 - \mu = -2$ und $\lambda_2 - \mu = 2$ gilt, welche beide betragsmäßig gleich groß und am größten sind.

(b) Den Eigenvektor zum Eigenwert λ_2 können wir mit $\mu = 0$ erreichen. Wie können wir garantieren, den Eigenvektor des (ohne Shift) kleinsten Eigenwertes λ_0 zu erreichen? Wie sieht es mit dem mittleren Eigenwert λ_1 aus?

Mit $\mu = \lambda_2 = 5$. Den mittleren Eigenwert können wir mit der direkten Power Iteration nie garantiert erreichen. Mit der Inversen Power Iteration ginge das aber, indem wir $\mu = \frac{\lambda_2 + \lambda_0}{2}$ wählen.

5 Differentialgleichungen [3 + 3 + 4 = 10 Pkt.]

Geben sei folgendes AWP:

$$\begin{aligned} y'(t) &= y(t) + 1 \\ y(0) &= y_0 \geq 0 \\ t &\geq 0 \end{aligned}$$

(a) Bestimmen Sie die analytisch korrekte Lösung des AWPs mithilfe von Separation der Variablen. Gehen Sie davon aus, dass unsere Eingabe verfälscht sei mit $y_\epsilon = y_0 + \epsilon$. Ist das AWP für $t \to \infty$ gut oder schlecht konditioniert?

$$\begin{aligned} \int_{y_0}^{y} \frac{1}{\eta + 1} \, \mathrm{d}\eta &= \int_{t_0}^{t} \mathrm{d}\tau \\ \ln|y+1| - \ln|y_0+1| &= t - t_0 = t \\ \frac{|y+1|}{|y_0+1|} &= e^t \\ |y+1| &= (y_0+1) \cdot e^t \\ y(t) &= (y_0+1) \cdot e^t - 1 \\ y_\epsilon &= y(t) + \epsilon \cdot e^t \end{aligned}$$

Es ist für $t \to \infty$ offensichtlich schlecht konditioniert.

(b) Stellen Sie mithilfe des expliziten Eulers eine Formel auf, welche direkt vom Startwert aus die Lösung nach der zweiten Iteration ausgibt. Stellen Sie etwas fest? Formulieren Sie eine Vermutung.

$$\begin{aligned} y_1 &= y_0 + \delta t \cdot (y_0 + 1) = y_0 \cdot (1 + \delta t) + \delta t \\ y_2 &= y_1 + \delta t \cdot (y_1 + 1) \\ y_2 &= y_0 \cdot (1 + \delta t) + \delta t + \delta t \cdot (y_0 \cdot (1 + \delta t) + \delta t + 1) \\ y_2 &= y_0 + y_0 \delta t + \delta t + \delta t \cdot (y_0 + y_0 \delta t + \delta t + 1) \\ y_2 &= y_0 + y_0 \delta t + \delta t + y_0 \delta t + y_0 \delta t^2 + \delta t^2 + \delta t \\ y_2 &= y_0 \cdot (1 + 2\delta t + \delta t^2) + \delta t \cdot (2 + \delta t) \\ y_2 &= y_0 \cdot (1 + \delta t)^2 + (1 + 2\delta t + \delta t^2) - 1 \\ y_2 &= y_0 \cdot (1 + \delta t)^2 + (1 + \delta t)^2 - 1 \end{aligned}$$

Vermutung:

$$y_n = y_0 \cdot (1 + \delta t)^n + (1 + \delta t)^n - 1$$

(c) Zeichnen Sie in folgende Grafik zwei Schritte des Heun-Verfahrens mit obigen $t_0 = 0$, $y_0 = 0$ sowie Schrittweite $\delta t = 1$ ein. Die eingezeichnete Funktion ist eine Lösung des ODEs aus Teilaufgabe (a).

Wenn man bei Heun die Schrittweite verdoppelt, um wieviel ändert sich dann der Diskretisierungsfehler?

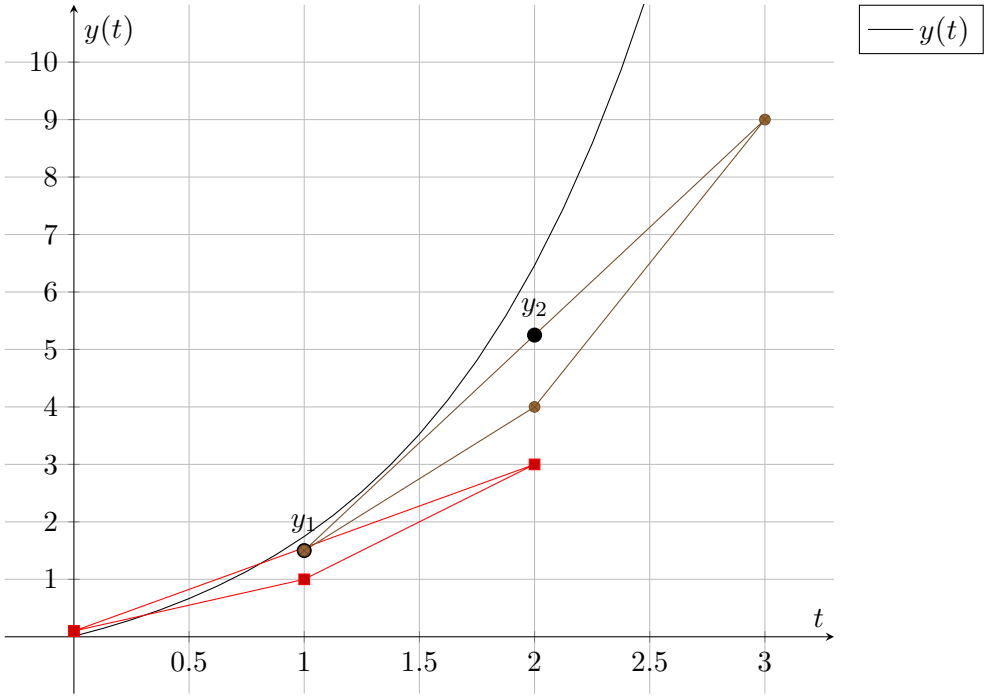

Abbildung 1: Plot der Funktion $y(t)$.

Wenn man bei Heun die Schrittweite verdoppelt, vervierfacht sich der Diskretisierungsfehler.

6 Programmieraufgabe [4 + 3.5 = 7.5 Pkt.]

Wir möchten stückweise lineare Interpolation implementieren. Hierfür werden immer zwei benachbarte Stützpunkte durch eine Gerade verbunden.

Wir haben das Array "pointsX" gegeben, welches die vorhandenen Stützstellen in aufsteigender Reihenfolge speichert. Mit der Funktion "fEval()" kann man sich dann die entsprechenden Stützwerte holen. n ist die Anzahl an genutzten Stützstellen, also die Länge des Arrays "pointsX".

Folgende Implementation sei gegeben:

```
float[] pointsX; // Is correctly set elsewhere
int n; // Is correctly set elsewhere

float fEval(float point) {
   // returns the evaluation at the position point
}

// the algorithm call
float fApprox(float x) {
   float result;
   float[] nearestGridPointsX = getNearestGridPoints(pointsX, x, n);
   result = linearInterpolate(nearestGridPointsX[0], nearestGridPointsX[1], x);
}
```

(a) Implementieren Sie "getNearestGridPoints()", welche für einen x-Wert die benachbarten (also am nächsten an x liegenden) Stützstellen zurückgibt. Der erste Index soll hierbei die Stützstelle $< x$ entsprechen.

Überprüfen Sie mögliche Fehlereingaben.

```
//returns the nearest two grid points as array. The first entry is the nearest
    point lower than x, the second the one higher than x.
float[] getNearestGridPoints(float[] pointsX, float x, int n) {

   float[] nearestGridPoints = new float[2];
   if (x < pointsX[0] || x > pointsX[n-1]) { throw new Exception(); }

   for (int i = 0; i < n; i++) {
      if (pointsX[i] >= x) {
         nearestGridPoints[0] = pointsX[i-1];
         nearestGridPoints[1] = pointsX[i];
         return nearestGridPoints;
      }
   }
   return nearestGridPoints;
}
```

(b) Implementieren sie nun die Funktion, die allgemein für zwei übergebene Stützpunkte eine dazwischen liegenden linear interpolierten Wert zurückgibt.

```
// leftX < x, rightX > x does not have to be checked
// x is the position on where f is evaluated

float linearInterpolate(float leftX, float rightX, float x) {

   float result;

   float leftY = fEval(leftX);
   float rightY = fEval(rightX);

   float gradient = (rightY - leftY) / (rightX - leftX);

   result = leftY + (x - leftX) * gradient;

   return result;
}
```