Technische Universität München
Institut für Informatik
Prof. Dr. Daniel Cremers
Dr. Rudolph Triebel
Nikola Tchipev, M.Sc.
Sebastian Rettenberger, M.Sc.

WiSe 2014/15

Numerisches Programmieren

3. Programmieraufgabe: Spline Interpolation.
Direkte Lösung linearer Gleichungssysteme

Direkte Lösung linearer Gleichungssysteme

Gauß-Elimination

Ziel dieser Programmieraufgabe ist die Implementierung von Verfahren zur Lösung von Systemen linearer Gleichungen. Man will also zu gegebenem

$$\begin{aligned} A &= (a_{i,j})_{0 \leq i,j < n} \in \mathbb{R}^{n,n} \\ b &= (b_i)_{0 \leq i < n} \in \mathbb{R}^n \end{aligned}$$

ein $x \in \mathbb{R}^n$ finden mit

$$A \cdot x = b.$$

Zunächst soll das Gauß-Eliminationsverfahren mit Spaltenpivotsuche implementiert werden. Im ersten Schritt des Gauß-Eliminationsverfahrens müssen alle Elemente der ersten Spalte unterhalb der Hauptdiagonalen zu Null werden. Dies wird dadurch erreicht, dass von jeder Zeile i die erste Zeile multipliziert mit $\frac{a_{i,0}}{a_{0,0}}$ abgezogen wird. Die Pivotsuche setzt noch vor diesem Schritt an und sorgt dafür, dass kein Element unter dem Hauptdiagonalenelement $a_{0,0}$ betragsmäsig größer ist als $a_{0,0}$. Dazu wird $a_{0,0}$ mit sämtlichen Elementen darunter verglichen. Falls es ein betragsmäsig Größeres gibt werden die beiden zugehörigen Zeilen vertauscht. Die Pivotsuche wird in jeder Spalte durchgeführt bevor die Einträge unter dem Hauptdiagonalenelement eliminiert werden. Im folgenden Beispiel ist die erste Spalte schon fertig bearbeitet. D.h., es muss nun auf der zweiten Spalte die Pivotsuche durchgeführt werden. Von den zu untersuchenden Matrixelementen ist das in der dritten Zeile das betragsmäsig Größte, die zweite Zeile wird daher mit der dritten Zeile vertauscht:

$$\left(\begin{array}{cccc|c} 1 & 4 & 8 & 3 & 7 \\ 0 & 2 & 2 & 4 & 0 \\ 0 & -3 & -7 & 2 & 1 \\ 0 & 1 & 5 & 2 & 2 \end{array}\right) \Rightarrow \left(\begin{array}{cccc|c} 1 & 4 & 8 & 3 & 7 \\ 0 & -3 & -7 & 2 & 1 \\ 0 & 2 & 2 & 4 & 0 \\ 0 & 1 & 5 & 2 & 2 \end{array}\right)$$

Bandmatrizen

Für ein Gleichungssystem mit n Gleichungen benötigt die Gauß-Elimination einen Rechenaufwand von $O(n^3)$ Operationen. Für Bandmatrizen lässt sich der Rechenaufwand aber erheblich reduzieren. Bandmatrizen sind Matrizen, bei denen nur auf der Hauptdiagonalen und einigen Nebendiagonalen Elemente ungleich Null sind:

$$\begin{pmatrix} a_{0,0} & \cdots & a_{1,p_A-1} & & & & \\ \vdots & \ddots & \ddots & \ddots & & & \\ a_{q_A-1,1} & \ddots & \ddots & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \ddots & a_{n-p_A,n-1} & \\ & & \ddots & \ddots & \ddots & \vdots & \\ & & & a_{n-1,n-q_A} & \cdots & a_{n-1,n-1} & \end{pmatrix}$$

Die untere Bandbreite q_A ist die Anzahl der Nebendiagonalen unterhalb der Hauptdiagonalen inklusive der Hauptdiagonalen. Die obere Bandbreite p_A ist analog definiert. Die Gesamtbandbreite des Bandes ist somit $p_A + q_A - 1$. Ein Gleichungssystem einer solchen Struktur lässt sich mit $O(n \cdot p_A \cdot q_A)$ Operationen lösen.

LR-Zerlegung (= LU-Zerlegung)

Ein Nachteil des Gauß-Eliminationsverfahrens ist, dass man das komplette Gleichungssystem erneut lösen muss, selbst wenn sich nur der Vektor b ändert, die Matrix A aber gleich bleibt. Hier bietet die LR-Zerlegung einen Vorteil. Zunächst wird die Matrix A in die beiden Matrizen L und R zerlegt. Dazu wird nur die Matrix A verwendet, b muss noch nicht festgelegt sein. Danach kann man für verschiedene b durch Anwenden der Vorwärts- und Rückwärts-Substitution die Lösung berechnen ohne die Matrix A erneut zerlegen zu müssen.

In der Vorlesung wurde die LR-Zerlegung vollbesetzter Matrizen vorgestellt. Im Rahmen dieser Programmieraufgabe soll die LR-Zerlegung von Bandmatrizen implementiert werden. Dazu soll die Klasse *BandMatrix* verwendet werden. Diese benötigt zur Speicherung der Bandmatrix nur $O(n \cdot (p + q - 1))$ Speicherplatz, auf die Matrixelemente kann aber über die übliche Matrixindizierung zugegriffen werden. Es darf aber natürlich nur auf solche Matrixelemente zugegriffen werden, die auch wirklich im Bereich des Bandes liegen.

Wird die LR-Zerlegung auf einer solchen Matrix durchgeführt, so sind die entstehenden Matrizen L und R auch Bandmatrizen. L ist hierbei eine Bandmatrix mit $q_L = q_A$ und $p_L = 1$ und R ist eine Bandmatrix mit $q_R = 1$ und $p_R = p_A$.

Führt man z.B. die LR-Zerlegung der Matrix

$$A = \begin{pmatrix} * & * & & & & \\ * & * & * & & & \\ & * & * & \ddots & & \\ & & * & \ddots & * & \\ & & & \ddots & * & * \\ & & & & * & * \end{pmatrix}, \; p_A = q_A = 2$$

durch, so haben die Matrizen L und R folgende Struktur:

$$L = \begin{pmatrix} 1 & & & & & \\ * & 1 & & & & \\ & * & 1 & & & \\ & & * & \ddots & & \\ & & & \ddots & 1 & \\ & & & & * & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} * & * & & & & \\ & * & * & & & \\ & & * & \ddots & & \\ & & & \ddots & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}.$$

Spline Interpolation. Bezier Kurven.

Wie schon in der Vorlesung und Tutorübung besprochen, kann globale Polynominterpolation zu großen Oszillationen führen. Stattdessen verwendet man oft stückweise Polynominterpolation mit der Anforderung von glatten Übergangen an den Stützpunkten.

In diesem Blatt werden wir wieder mit kubische Polynomen arbeiten. Die Splinefunktion $s(x)$ soll überall zweimal stetig differenzierbar sein ($s \in \mathcal{C}^2$) und dafür keine anderen Informationen benötigen als die $n+1$ zu interpolierenden Stützpunkte $(x_0, y_0), \ldots, (x_n, y_n)$ und die Ableitungen y'_0 und y'_n an der erste und letzte Stützpunkten. Im Übungsblatt 6, haben wir gesehen, dass dieser Ansatz zu einem $(n-1)\times(n-1)$ linearen Gleichungssystem führt für die Ableitungen an den inneren Punkten $y'_i, i = 1, \ldots, (n-1)$. In diese Aufgabe werden wir allerdings mit nicht äquidistante Stützstellen arbeiten müssen (weil die Stützstellen anhand die grafische Benutzeroberfläche vorgegeben werden).

Sei $h_i = x_i - x_{i-1}$, $i = 1, \ldots, n$. Dann sieht das System für $i = 2, \ldots (n-2)$ so aus:

$$\frac{y'_{i-1}}{h_i} + \left(\frac{2}{h_i} + \frac{2}{h_{i+1}}\right) y'_i + \frac{y'_{i+1}}{h_{i+1}} = 3\left(\frac{y_i - y_{i-1}}{h_i^2} + \frac{y_{i+1} - y_i}{h_{i+1}^2}\right). \tag{1}$$

Für $i = 1$ und $i = (n-1)$ muss man im Anspruch nehmen, dass y'_0 und y'_n schon bekannt sind:

$$\left(\frac{2}{h_1} + \frac{2}{h_2}\right) y'_1 + \frac{y'_2}{h_2} = 3\left(\frac{y_1 - y_0}{h_1^2} + \frac{y_2 - y_1}{h_2^2}\right) - \frac{y'_0}{h_1} \tag{2}$$

$$\frac{y'_{n-2}}{h_{n-1}} + \left(\frac{2}{h_{n-1}} + \frac{2}{h_n}\right) y'_{n-1} = 3\left(\frac{y_{n-1} - y_{n-2}}{h_{n-1}^2} + \frac{y_n - y_{n-1}}{h_n^2}\right) - \frac{y'_n}{h_n}. \tag{3}$$

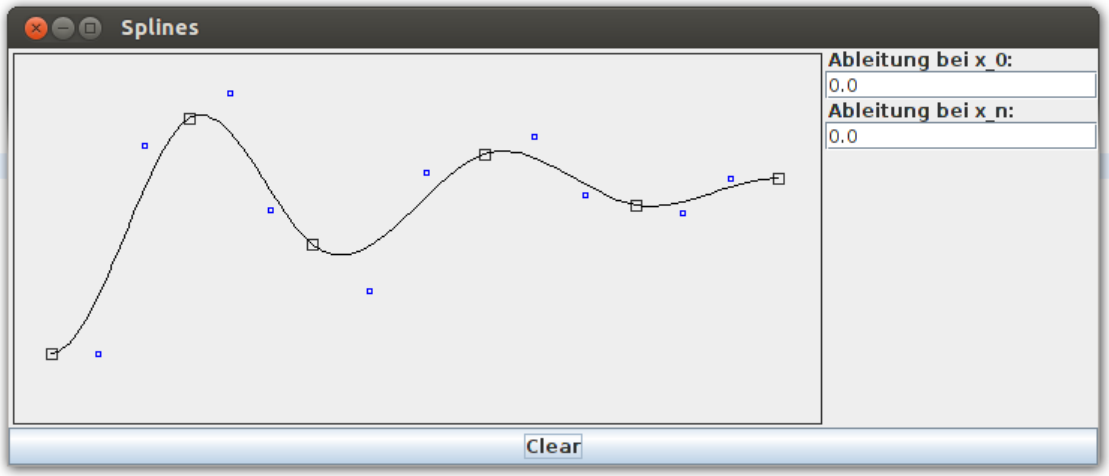

Abbildung 1: Spline Interpolation mit Bezier-Kurven auf den Teilintervale. Schwarz: Stützpunkte und Splinefunktion $s(x)$, blau: Kontrolpunkten für die Bezier-Kurven in jedem Teilinterval.

Im form von $Ay' = b$ sieht es so aus:

$$Ay' = \begin{pmatrix} \left(\frac{2}{h_1} + \frac{2}{h_2}\right) & \frac{1}{h_2} & & \\ \frac{1}{h_2} & \left(\frac{2}{h_2} + \frac{2}{h_3}\right) & \ddots & \\ & \ddots & \ddots & \frac{2}{h_{n-1}} \\ & & \frac{2}{h_{n-1}} & \left(\frac{2}{h_{n-1}} + \frac{2}{h_n}\right) \end{pmatrix} \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_{n-2} \\ y'_{n-1} \end{pmatrix}$$

$$b = 3 \begin{pmatrix} \frac{y_1 - y_0}{h_1^2} + \frac{y_2 - y_1}{h_2^2} \\ \frac{y_2 - y_1}{h_2^2} + \frac{y_3 - y_2}{h_3^2} \\ \vdots \\ \frac{y_{n-2} - y_{n-3}}{h_{n-2}^2} + \frac{y_{n-1} - y_{n-2}}{h_{n-1}^2} \\ \frac{y_{n-1} - y_{n-2}}{h_{n-1}^2} + \frac{y_n - y_{n-1}}{h_n^2} \end{pmatrix} - \begin{pmatrix} \frac{y'_0}{h_1} \\ 0 \\ \vdots \\ 0 \\ \frac{y'_n}{h_n} \end{pmatrix}$$

Durch Vorgabe vom $x_0, \ldots, x_n$, $y_0, \ldots, y_n$ und den Ableitungen an den Intervallgrenzen y'_0 und y'_n wird das LGS eindeutig lösbar. In der Klasse *Spline* sollen Methoden implementiert werden, die zu den eben genannten Daten die (Band-) Matrix A und die rechte seite b aufstellen. Die tatsächliche Lösung diesem System wird anhand direkter Löser linearer Gleichungssysteme berechnet.

Sobald wir die Ableitungen berechnet haben, können wir die Kurve $s(x)$ eindeutig zeichnen. Um den kubischen Polynom auf einem Teilinterval zu zeichnen werden wir Bezier Kurven verwenden, siehe Abbildung 1.

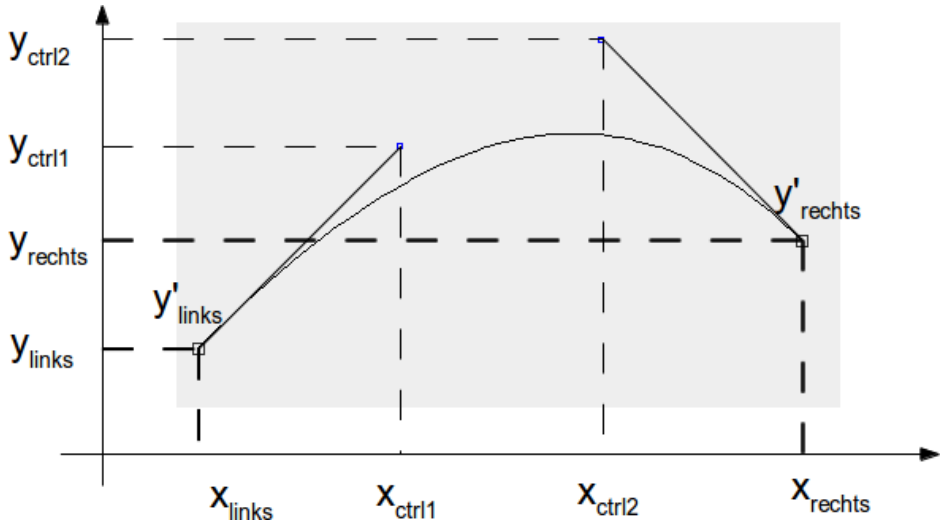

Abbildung 2: Teilinterval.

Wir zeichnen das kubischen Polynom $p(x)$ das

$$\begin{aligned} p(x_{links}) &= y_{links} \\ p(x_{rechts}) &= y_{rechts} \\ p'(x_{links}) &= y'_{links} \\ p'(x_{rechts}) &= y'_{rechts} \end{aligned}$$

erfüllt anhand einer kubischen Bezier-Kurve. Wir müssen lediglich die Start- und Endpunkten und die zwei Kontrolpunkten wählen:

	X-Wert	Y-Wert
Startpunkt	x_{links}	y_{links}
Kontrolpunkt 1	$x_{links} + \frac{h}{3}$	$y_{links} + \frac{h}{3} y'_{links}$
Kontrolpunkt 2	$x_{rechts} - \frac{h}{3}$	$y_{rechts} - \frac{h}{3} y'_{rechts}$
Endpunkt	x_{rechts}	y_{rechts}

wobei $h = x_{rechts} - x_{links}$.

Licht Aus (20 Bonuspunkte!)

Die Ausgangssituation des Spiels “Licht Aus!”[1] ist ein 5x5 Lichtboard an, auf dem einige Lichter eingeschaltet, die restlichen ausgeschaltet sind. Durch Klicken auf ein Licht ändert sich dessen Zustand. War das Licht an, wird es durch den Klick ausgeschaltet. War das Licht aus, wird es umgekehrt durch den Klick eingeschaltet. Simultan passiert das Gleiche auch mit den vier Nachbarlichtern: oben, unten, links und rechts.

[1] http://en.wikipedia.org/wiki/Lights_Out_(game)

Ziel des Spiels ist es, alle Lichter mit möglichst wenigen Klicks auszuschalten. Abbildung 3 zeigt die mitgelieferte Lichterboard Java Anwendung, die als Ausgangspunkt für die Programmieraufgabe dient.

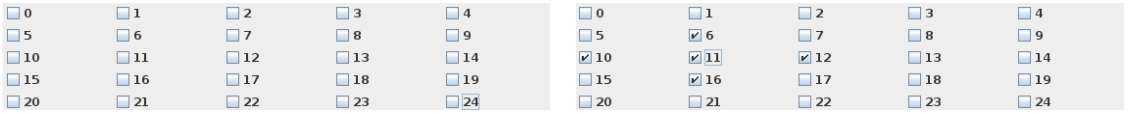

Abbildung 3: Links: Ausgeschaltetes Lichtboard. Rechts: Licht 11 wurde geklickt.

Man kann selbstverständlich - ausgehend von einem Lichtermuster - die Lösung (falls überhaupt vorhanden) durch manuelles Klicken finden. Es gibt aber auch eine Möglichkeit, die Lösung zu berechen. Dies soll Inhalt dieser Aufgabe sein.

Eine vollständige Beschreibung der Lösung kann man hier[2] finden. Kurz zusammengefasst wird folgendermaßen vorgegangen:

- Das Lichterboard kann als eine Kette mit $5 \times 5 = 25$ binären Einträgen dargestellt werden. Eine 1 bedeutet, dass das Licht an der ensprechende Stelle an ist, eine 0 dass es aus ist:

  $$00000\ 00000\ 00000\ 00000\ 00000$$

- Ein Klick auf das Lichterboard kann ebenso von 25 binären Einträgen dargestellt werden. Sind z.B. alle Lichter auf dem Board aus, würde durch das Klicken auf das Licht im linken oberen Eck (Licht 0) das folgende Board entstehen:

  $$a_0\text{: } 11000\ 10000\ 00000\ 00000\ 00000$$

  also Licht 0, 1 und 5 wären danach eingeschaltet. Alle anderen 24 möglichen Klickaktionen können analog dargestellt werden.

- An dieser stelle ahnt man vielleicht schon, dass das Resultat einer Klickaktion nichts anderes als eine XOR-Operation zwischen den binären Ketten des Boards und der Klickaktion ist. Beispiel:

  $$\begin{gathered} board\text{: } 00000\ 00011\ 00001\ 00000\ 00000 \\ XOR \\ a_0\text{: } 11000\ 10000\ 00000\ 00000\ 00000 \\ = \\ 11000\ 10011\ 00001\ 00000\ 00000 \end{gathered}$$

- Die Klickaktionen a_0 bis a_{24} können in einer beliebige Reihenfolge angewendet werden, denn $(board\ XOR\ a_1)\ XOR\ a_2 = (board\ XOR\ a_2)\ XOR\ a_1$ (Kommutativität und Assoziativität)

- Das Problem kann man folgendermaßen formuliert werden (Addition ist hier ein XOR):

  $$c_0 \cdot a_0 + c_1 \cdot a_1 + \ldots + c_{24} \cdot a_{24} + y = 0$$

  Die Interpretation davon lautet: Welche von den 25 Klickaktionen a_0 bis a_{24} muss auf das Ausgangboard y angewandt werden, um am Ende zum Zustand 0 (alles aus)

[2] http://aix1.uottawa.ca/~jkhoury/game.html

zu gelangen? $c_i = 1$ bedeutet hier, dass die Klickaktion a_i angewandt wird, $c_i = 0$ bedeutet, dass die Klickaktion a_i nicht angewandt wird. Nach einem XOR mit y (Äquivalenzumformung) kommen wir auf:

$$c_0 \cdot a_0 + c_1 \cdot a_1 + \ldots + c_{24} \cdot a_{24} = y$$

- Nun kann man das Ganze als ein Lineares Gleichungssystem schreiben:

$$Mx = y$$

wobei

$$M = \begin{pmatrix} A & I & 0 & 0 & 0 \\ I & A & I & 0 & 0 \\ 0 & I & A & I & 0 \\ 0 & 0 & I & A & I \\ 0 & 0 & 0 & I & A \end{pmatrix} A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} I = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Die Lösungskette x gibt dann an, welche Klicks benötigt werden, um das ganze Board auszuschalten.

Konkrete Aufgaben

Im Folgenden werden die zu implementierenden Methoden aufgelistet. Details zur Implementierung finden Sie jeweils in den Kommentaren zu den einzelnen Methoden. Testen Sie unbedingt alle Ihre Methoden. Hierzu wird empfohlen die Datei *Aufgabe_3.java* entsprechend zu erweitern.

- Klasse *Spline*, Methode *baueMatrix*: Baut aus den Stützwerten die Matrix A auf.
- Klasse *Spline*, Methode *baueRechteSeite*: Baut aus den Stützwerten und Randbedingungen den Vektor b auf.
- Klasse *Spline*, Methode *baueBezierKurve*: Baut aus den Stützwerten und Ableitungen auf einem Teilinterval die kubische Bezierkurve.
- Klasse *Gauss*, Methode *loese*: Gauß-Elimination mit Spaltenpivotsuche.
- Klasse *LRZerlegung*, Methode *zerlege*: LR-Zerlegung der Matrix A.
- Klasse *LRZerlegung*, Methode *substitution*: Vorwärts- und Rückwärts-Substitution.
- Klasse *Gauss*, Methode *loeseBinaer*: Für die Lösung des Spiels "Light Aus!"muss eine Gauss Elimination für binäre Matrizen implementiert werden. Tipp: Dies ist sehr ähnlich zur klassischen Gauss Elimination. Es wird lediglich überall dort der XOR-Operator verwendet, wo sonst klassich der Additions-Operator zum Einsatz kommt.

  Wenn Sie diese Bonusaufgabe lösen, kriegen Sie 20 Bonuspunkte. D.h. Sie können bis zur 120 Punkte in dieser Programmieraufgabe bekommen.

Formalien und Hinweise

- Das Programmgerüst erhalten Sie auf den Webseiten zur Vorlesung.
- Ergänzen Sie das Programmgerüst **auf alle Fälle an den dafür vorgegebenen Stellen**! Darüber hinaus können Sie eigene Erweiterungen hinzufügen, falls Sie diese benötigen. Wichtig dabei ist, dass die Signaturen der Klassen mit ihren Methoden und Attributen erhalten bleiben, da sie den für die von uns durchgeführten Testfälle erforderlichen Code enthalten. Fügen Sie **keine** weiteren Dateien hinzu, da diese von unserem automatischen Korrekturtool nicht verarbeitet werden.
- Beseitigen Sie vor Abgabe Ihres Programms alle Ausgaben an die Konsole, die Sie eventuell zu Debugging- oder Testzwecken eingefügt haben und reichen sie Ihre Lösung bis zum **05. Januar 2015, 12:00 Uhr** über Moodle ein.
- Bitte laden Sie Ihre java-Dateien als flaches tgz-Archiv hoch. Der Dateiname ist beliebig wählbar, bei der Erweiterung muss es sich jedoch um **.tgz** oder **.tar.gz** handeln.

  Ein solches Archiv können Sie beispielsweise mit dem Linux-Tool tar erstellen, indem Sie die laut Aufgabenstellung zu bearbeitenden java-Dateien in ein sonst leeres Verzeichnis legen und dort anschließend den Befehl

  ```
  > tar cvvzf numpro_aufg3.tgz *.java
  ```

ausführen.
tgz-Archive sind keine Zip- oder Rar-Archive! Es funktioniert also nicht ein Zip- oder Rar-Archiv zu erstellen und die Endung des Dateinamens in .tgz zu ändern.

- Für die Abgabe der Programmieraufgaben ist das Eintragen in eine Gruppe notwendig.

Bitte beachten Sie:

- *Alle Abgaben, die nicht den formalen Kriterien genügen, werden grundsätzlich mit 0 Punkten bewertet!*
- *Wir testen die Abgaben auch auf Plagiate! Sollte sich herausstellen, dass Sie bei Ihrer Abgabe abgeschrieben haben oder abschreiben haben lassen, bewerten wir die komplette Abgabe mit 0 Punkten!*