Technische Universität München
Institut für Informatik
Prof. Dr. Hans-Joachim Bungartz
Michael Lieb, M. Sc.
Dipl.-Inf. Gerrit Buse

WS 2009/2010

Numerisches Programmieren

2. Programmieraufgabe: Polynominterpolation

Lagrange-Polynome

Eine einfache Methode zur Bestimmung eines Interpolationspolynoms zu gegebenen Stützpunkten $(x_i, y_i), i \in 0, \ldots, n$ verwendet die sogenannten Lagrange-Polynome. Das Lagrange-Polynom des k-ten Stützpunkts lautet:

$$L_k(x) := \prod_{i:i \neq k} \frac{x - x_i}{x_k - x_i}. \tag{1}$$

Um das Interpolationspolynom aufzustellen, werden die einzelnen Lagrange-Polynome mit den zugehörigen Stützwerten gewichtet und aufsummiert:

$$p(x) := \sum_{k=0}^{n} y_k \cdot L_k(x). \tag{2}$$

In dieser Praktikumsaufgabe sollen Methoden zum Erstellen eines Interpolationspolynoms aus gegebenen Stützpunkten implementiert werden. Im zur Verfügung gestellten Programmrahmen finden Sie dazu die Dateien *Polynom.java* und *Lagrange.java*. *Polynom.java* realisiert die Klasse *Polynom*, mit der Polynome dargestellt und ausgewertet werden können. Ein Polynom vom Grad n hat $n + 1$ Koeffizienten a_n:

$$p(x) := a_0 \cdot x^0 + a_1 \cdot x^1 + \ldots + a_n \cdot x^n. \tag{3}$$

Zur Speicherung des Polynoms genügt es daher, die $n+1$ Koeffizienten $a_0, \ldots, a_n$ zu speichern. Dazu dient die Membervariable *coefficients* der Klasse *Polynom*.

Die Klasse *Lagrange* enthält Methodenrümpfe zum Erstellen der Lagrange-Polynome (siehe Glg. (1)) und des Interpolationspolynoms (siehe Glg. (2)).

Numerische Ableitung

Bei Polynomen ist das Differenzieren analytisch sehr einfach. Bei komplexeren Funktionen ist das analytische Bilden der Ableitung allerdings nicht immer möglich. Sofern die Funktion an einer Stelle x_0 differenzierbar ist, existiert die Ableitung an dieser Stelle und entspricht

genau der Steigung der Tangente an der Funktion in diesem Punkt. Diese Tangente lässt sich durch eine Sekante approximieren:

$$D_f(x_0, h) = \frac{f(x_0 + h) - f(x_0)}{h}. \tag{4}$$

$D_f(x_0, h)$ wird rechtsseitiger Differenzenquotient genannt. In Abb. 1 ist der Zusammenhang graphisch dargestellt.

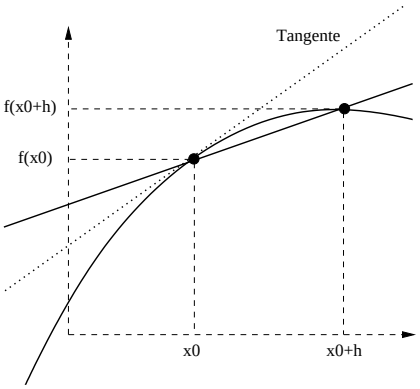

Abbildung 1: rechtsseitiger Differenzenquotient

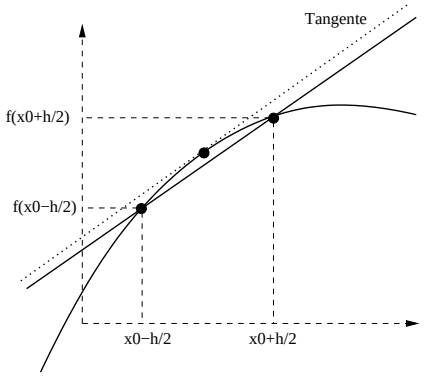

Abbildung 2: zentraler Differenzenquotient

Eine andere Möglichkeit ist die Verwendung des zentralen Differenzenquotienten (Abb. 2):

$$D_f(x_0, h) = \frac{f(x_0 + \frac{h}{2}) - f(x_0 - \frac{h}{2})}{h} \tag{5}$$

Für beide Möglichkeiten ist die Wahl des richtigen h entscheidend (vgl. Übungsblatt 2, Aufgabe 2). Wird es zu groß gewählt, so erreicht man nur eine sehr schlechte Näherung für die Ableitung. Bei zu kleinem h kann allerdings Auslöschung auftreten.

Um den Fehler bei der Näherung bestimmen zu können, muss die exakte Ableitung bekannt sein. Es soll daher im Rahmen dieser Programmieraufgabe untersucht werden, wie sich der Fehler bei der numerischen Ableitung von Polynomen für unterschiedliche h verhält. Die im Programmrahmen bereitgestellte Datei *Derivative.java* enthält bereits Methoden zur Berechnung der Differenzenquotienten bereit (siehe Glg. (4) und Glg. (5)). Ihre Aufgabe ist es, die Methoden zur Bestimmung des optimalen h und eine weitere Methode zur Bestimmung der analytischen Ableitung eines Polynoms zu implementieren.

k aus n Secret Sharing

Neben dem Verschlüsseln von Informationen ist es in der Kryptologie auch oftmals wichtig sicherzustellen, dass eine Information von einer bestimmten Person stammt (dazu dient beispielsweise die digitale Unterschrift). Durch Kenntnis eines Schlüssels kann eine Person authentifiziert werden. Beim *k aus n secret sharing* will man einen Schlüssel in n Teilschlüssel aufteilen und an n Personen verteilen. Nun sollen die Teilschlüssel von beliebigen k Personen

(mit $k < n$) ausreichen, um den Schlüssel zu rekonstruieren, weniger als k Teilschlüssel sollen jedoch keine Rückschlüsse auf den Gesamtschlüssel zulassen.

Stellen Sie sich dazu folgende Anwendung vor: $n = 7$ Gesellschaftern gehört eine Bank. Zur Durchführung großer Transaktionen muss eine Mehrheit der Gesellschafter zustimmen. Es müssen also mindestens $k = 4$ Gesellschafter z.B. über eine Chipkarte ihren Teilschlüssel einbringen. Dabei soll es keine Rolle spielen, welche 4 Gesellschafter dies tun.

Der Mathematiker Adi Shamir (unter anderem bekannt durch das RSA-Verfahren) hat 1979 eine Lösung dieses Problems mittels Polynominterpolation vorgeschlagen. Interessierte können sich das Paper *shamir.pdf* unter der Rubrik Praktikum von der Vorlesungswebseite herunterladen.

Das Shamir-Verfahren

Ein Polynom p vom Grad $k-1$ ist eindeutig durch seine Werte $p(x_i)$ an k paarweise verschiedenen Stützstellen x_i bestimmt. Bei obigem Beispiel mit den sieben Gesellschaftern könnte man - z.B. an den vier Stützstellen $x_i, 1 \leq i \leq 4$ - zufällige Zahlen als Stützwerte wählen. Durch diese vier Stützpunkte ist ein Polynom mit einem Grad von drei eindeutig gegeben. In Ausnahmefällen kann der Grad auch geringer sein, z.B. wenn alle Punkte auf einer Geraden liegen. Im Rahmen dieser Programmieraufgabe werden solche Ausnahmefälle ignoriert.

Als Gesamtschlüssel können wir nun z.B. den Funktionswert des Polynoms an der Stelle 0 verwenden. Nur wenn die Werte aller vier Stützpunkte bekannt sind, lässt sich das Polynom aufstellen und der Wert an der Stelle 0 berechnen. Bisher haben wir also vier Teilschlüssel, die nur gemeinsam den Zugriff auf den Gesamtschlüssel erlauben. Die drei Teilschlüssel für die verbleibenden Gesellschafter erhält man durch das Auswerten des Polynoms an den Stellen $x_j, 5 \leq j \leq 7$. Jeder der Gesellschafter hat nun als Teilschlüssel den Stützwert an einer ihm zugeordneten Stützstelle (vgl. Abb. 3). Durch das Konstruktionsprinzip ist klar, dass alle diese Punkte auf dem aufgestellten Polynom mit Grad drei liegen. Wie bereits erläutert, kann man ein Polynom dritten Grades aber eindeutig aus vier Stützwerten an paarweise verschiedenen Stützstellen konstruieren. Es können also beliebige vier Gesellschafter gemeinsam das Polynom aufstellen und den Gesamtschlüssel rekonstruieren. Mit weniger als vier Stützpunkten sind jedoch keinerlei Rückschlüsse auf den Gesamtschlüssel möglich.

Für die Implementierung des Shamir-Verfahrens wird zunächst eine Methode zum Auswerten eines Polynoms benötigt. Wir wollen aber nicht für jede Transaktion das Polynom komplett aufstellen, daher geschieht das Auswerten mit dem Aitken-Neville-Schema. Dabei wird direkt aus den Stützstellen der Wert des zugehörigen Interpolationspolynoms an einer gegebenen Stelle berechnet. Dazu muss die Methode *evaluate* in der Datei *Aitken_real.java* implementiert werden. Die Klasse *Shamir_real* dient der Erstellung und Verwaltung der Teilschlüssel.

Problem

Versuchen Sie, mit unterschiedlichen Kombinationen einer größeren Anzahl an Gesellschaftern den Gesamtschlüssel zu erzeugen (Methode *create_key*) und vergleichen Sie ihn mit dem „echten“ Gesamtschlüssel. Dabei stellt man fest, dass es bei höheren Polynomgraden nicht mehr egal ist, welche Gesellschafter versuchen, den Gesamtschlüssel zu erzeugen. Die Ursache dafür sind Rundungsfehler.

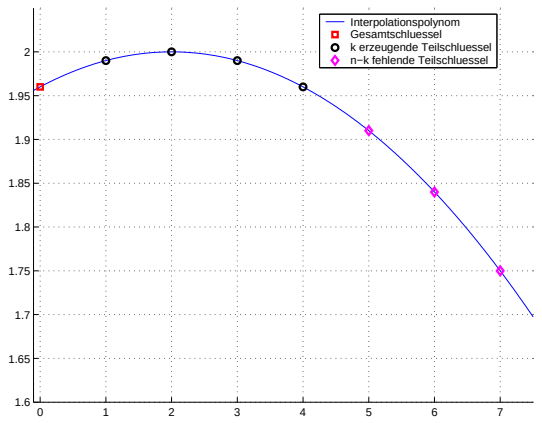

Abbildung 3: Shamir mit reellen Zahlen.

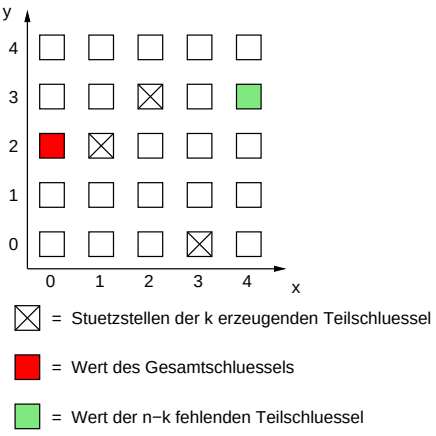

Abbildung 4: Shamir mit ganzen Zahlen.

Shamir mit ganzen Zahlen

Zur Vermeidung von Rundungsfehlern kann das Shamir-Verfahren auch mit natürlichen Zahlen (bzw. einer endlichen Menge natürlicher Zahlen) anstatt mit reellen Zahlen umgesetzt werden. Die mathematischen Grundlagen dazu werden im Folgenden erläutert. Es sei $\mathbb{Z}_p$ ein endlicher Körper bestehend aus der Menge $\{z_i | i \in \{0, \ldots, p\}\}$ mit den Verknüpfungen Addition $\oplus$ und Multiplikation $\odot$ und p eine Primzahl[1]. Das Produkt zweier Zahlen aus einem solchen Körper berechnet man, indem man zunächst beide Zahlen multipliziert (wie man dies mit natürlichen Zahlen tun würde), und danach *Produkt mod p* rechnet, d.h. den ganzzahligen Rest der Division des Zwischenergebnisses durch die Primzahl p berechnet:

$$\begin{aligned} \text{Für } x, y \in \mathbb{Z}_p: \quad x \oplus y &= (x + y) \ mod \ p, \\ x \odot y &= (x \cdot y) \ mod \ p. \end{aligned}$$

Bei der Addition und Substraktion wird auch zunächst das Zwischenergebnis berechnet. Ist es nicht aus der Menge der Zahlen des Körpers, so muss p addiert bzw. subtrahiert werden. Die Division gestaltet sich etwas schwieriger, da ganze Zahlen normalerweise nicht durcheinander geteilt werden können. Laut dem kleinen Satz von Fermat gilt für alle Primzahlen p und alle a, die kein Vielfaches von p sind:

$$a^{p-1} \ mod \ p = (a^{p-2} \ mod \ p) \odot a = 1.$$

Teilt man nun durch a, so folgt:

$$a^{p-2} = \frac{1}{a}.$$

Das **Teilen durch** a entspricht daher dem **Multiplizieren mit** a^{p-2}.

Polynome auf diesen Körpern sind zwar nicht sehr anschaulich, aber für das Shamir-Verfahren funktionieren sie problemlos (vgl. Abb. 4). Zur Realisierung müssen lediglich die Methoden aus *Shamir_real*, basierend auf den hier vorgestellten Rechenregeln, adaptiert werden. Die Methodenrümpfe dazu finden Sie in der Datei *Aitken_int.java*.

[1] siehe z.B. www.primzahlen.de

Konkrete Aufgaben

Im Folgenden werden die zu implementierenden Methoden aufgelistet. Details 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 *Polynom*: Methode *evaluate*
- Klasse *Lagrange*: Methoden *create_lagrange_base* und *assemble_polynoms*
- Klasse *Derivative*: Methoden *analytic_derivative*,*compare_deriv_forward* und *compare_deriv_central*
- Klasse *Aitken_real*: Methode *evaluate*
- Klasse *Shamir_real*: Methoden *create_key* und den Konstruktor
- Klasse *Aitken_int*: Methoden *evaluate*, *mod* und *pow_mod*
- Klasse *Shamir_int*: Methoden *create_key* und den Konstruktor

Formalien

- Das Programmgerüst erhalten Sie auf den Webseiten zur Vorlesung.
- Ergänzen Sie das Programmgerüst bitte **nur an den dafür vorgegebenen Stellen!** Falls Sie die Struktur der Programme eigenmächtig verändern, können wir sie evtl. nicht mehr testen.
- Beseitigen Sie vor Abgabe Ihres Programms alle Ausgaben an die Konsole!
- Bitte reichen Sie Ihre Abgabe bis zum **14. Dezember 2009, 12:00 Uhr** über das Web-Portal ein. Die Auswertung steht dann ab spätestens 16. Dezember 12:00 Uhr zum Abruf bereit.