

## Problem Set 5

Johanna Sacher | 221103072

johanna.sacher@student.uni-halle.de

Christian Paffrath | 221103085

christian.paffrath@student.uni-halle.de

### I Floating Point Filter

Suppose you are dealing with two-dimensional points in integral homogenous coordinates. Evaluating the orientation of three points (see Slide 269) can be a computational problem, as

- the exact evaluation involves the manipulation of large integers that go beyond the built-in `int` and `long` data types, and which is thus computational expensive, and
- the approximate evaluation using floating point arithmetic can be very error-prone.

An approach to overcome this problem is to use floating point filters. These filters use floating point arithmetic whenever it is safe, and exact evaluation only if necessary.

To use a floating point filter for the orientation of three points  $p, q, r$  in homogenous coordinates, you need to calculate an approximation  $\tilde{E}$  of the determinant

$$E = \begin{vmatrix} p_x & p_y & p_w \\ q_x & q_y & q_w \\ r_x & r_y & r_w \end{vmatrix} \quad (\text{I.1})$$

using floating point arithmetic, and the associated bound  $B = \varepsilon \cdot \text{ind}_E \cdot \text{mes}_E$

on the maximal difference between  $\tilde{E}$  and the (unknown) exact value  $E$ . Here,

- $\varepsilon$  is a constant known as the *machine precision*,
- the index  $\text{ind}_E$  can be set to 11 in case of the orientation problem, and
- $\text{mes}_E$  is calculated recursively.

Then, the floating point filter returns

$$\text{orient}(p, q, r) = \begin{cases} 1, & \text{if } \tilde{E} > B, \\ -1, & \text{if } \tilde{E} < -B, \\ 0 & \text{if } B < 1, \\ \text{sign}(E), & \text{otherwise.} \end{cases} \quad (\text{I.2})$$

Note that an exact evaluation of  $E$  is required only in the last case.

## I.a Fehlerberechnung mit Expression Tree

Show that  $ind_E = 9$  is valid with respect to the table on Slide 274. For this purpose, draw the expression tree associated to the arithmetic expression  $E$  and discuss how the error of the root node is calculated.

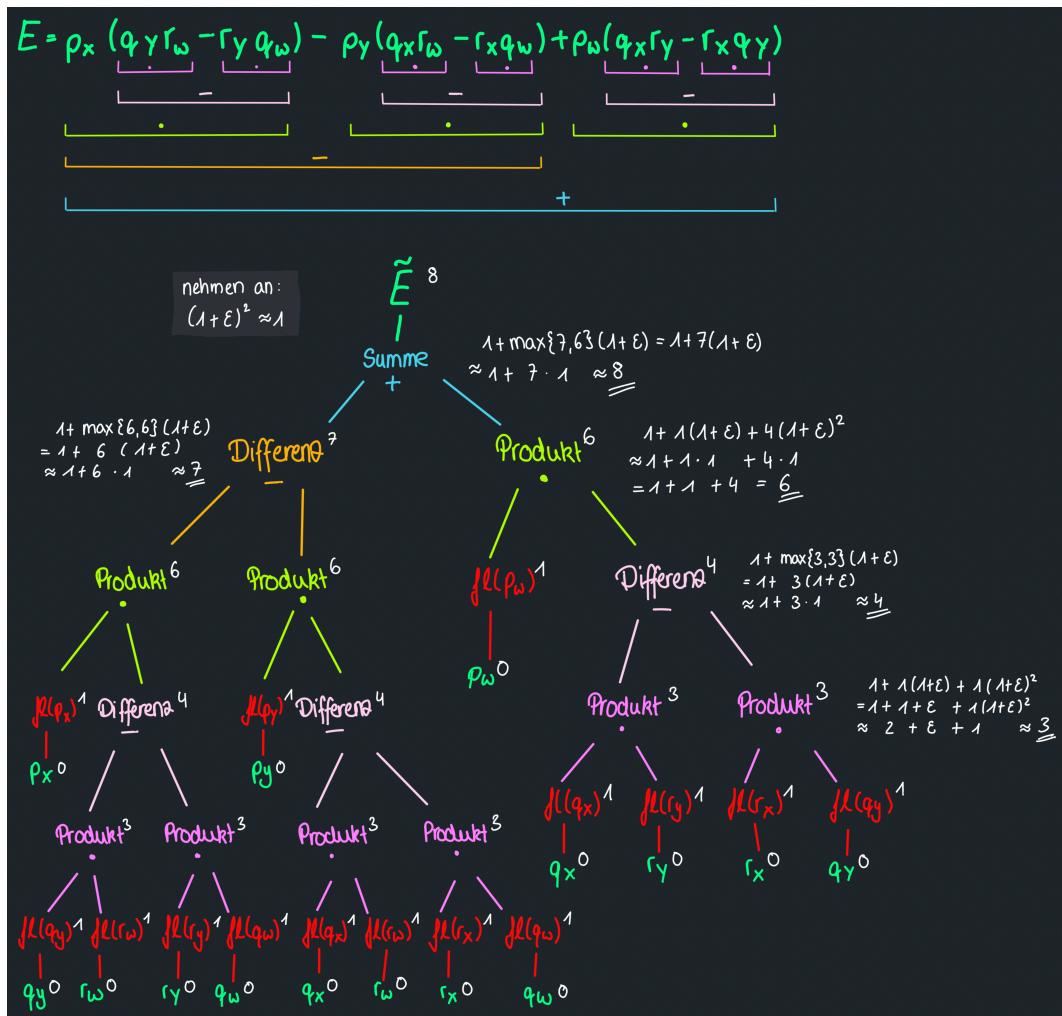


Abbildung I.1: Ausdruck  $E$  und Rekursives Bestimmen von  $ind_E$  im Ausdrucksbaum für  $E$

Der Ausdrucksbaum wird ausgehend von  $E$  konstruiert. Abbildung I.1 zeigt, wie der Fehler  $ind_E = 9$  des Wurzelknotens von unten nach oben rekursiv berechnet wird. Hier haben wir uns an der Tabelle von Folie 274 orientiert und sind zu einem Ergebnis von  $\approx 8$  für  $ind_E$  gekommen. Da der Index nicht von den Werten von  $p, q$  und  $r$  abhängt sondern nur von der Anzahl der Punkte (3 in diesem Fall), haben wir damit gezeigt, dass  $ind_E = 9$  eine valide obere Schranke für den Index ist.

## I.b Orientierung von 3 Punkten bestimmen

Implement routines to evaluate the orientation of three points using

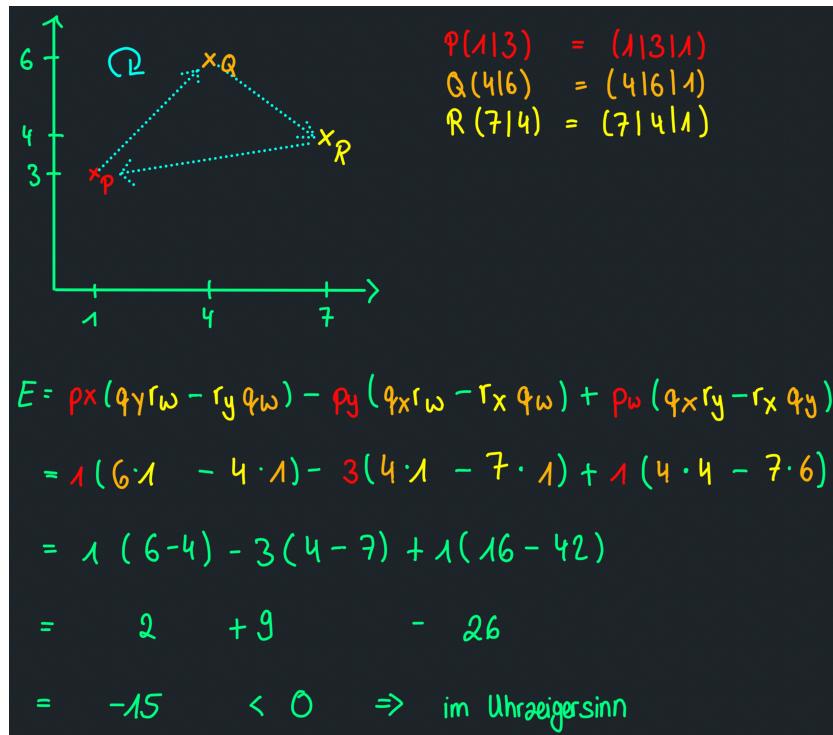
- single and double precision arithmetic (`float, double`),
- exact arithmetic (using a data type for large integers),
- a floating point filter.

Wir haben [drei Routinen](#) zur Lösung des Orientierungsproblems für drei Punkte implementiert:

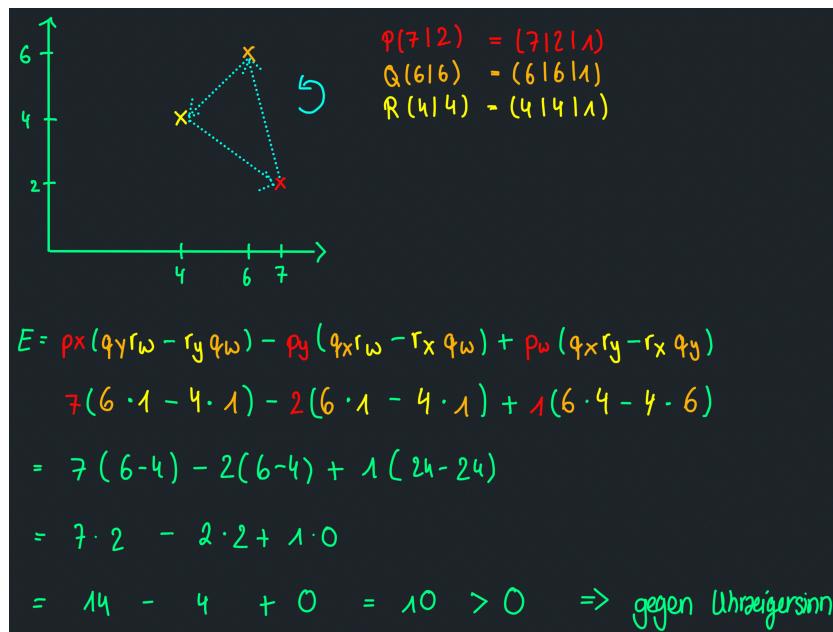
- [Floating-Point Arithmetik](#), bei der die homogenen Koordinaten zuerst in kartesischen Koordinaten umgerechnet werden. Anschließend wird das Vorzeichen der approximierten Determinante  $\tilde{E}$  mit double-precision floating-point Arithmetik bestimmt.
- [Exakte Integer-Arithmetik](#), bei der die Determinante  $E$  aus den homogenen Koordinaten mit Hilfe des Python-Moduls [Fractions](#) für unendliche Präzision berechnet wird.
- Einen [Floating-Point Filter](#), welcher zur Approximierung der Determinante  $\tilde{E}$  double-precision floating-point Arithmetik nutzt,  $mes_E$ ,  $ind_E$  und die obere Grenze  $B$  berechnet, und nur [wenn wirklich notwendig](#) exakte Ganzzahl-Arithmetik anwendet.

Unsere Implementierung haben wir zuerst »händisch« grob auf die drei Fälle *Punkte im Uhrzeigersinn* (s. [Abbildung I.2](#)), *Punkte gegen den Uhrzeigersinn* (s. [Abbildung I.3](#)) und *Punkte kollinear* (s. [Abbildung I.4](#)) getestet. Alle drei Tests waren erfolgreich.

Anschließend haben wir unsere Implementierungen genutzt, um die in [Teilaufgabe I.c](#) generierten Test-Instanzen auszuwerten.



**Abbildung I.2:** Negative Determinante → Orientierung im Uhrzeigersinn



**Abbildung I.3:** Positive Determinante → Orientierung gegen den Uhrzeigersinn

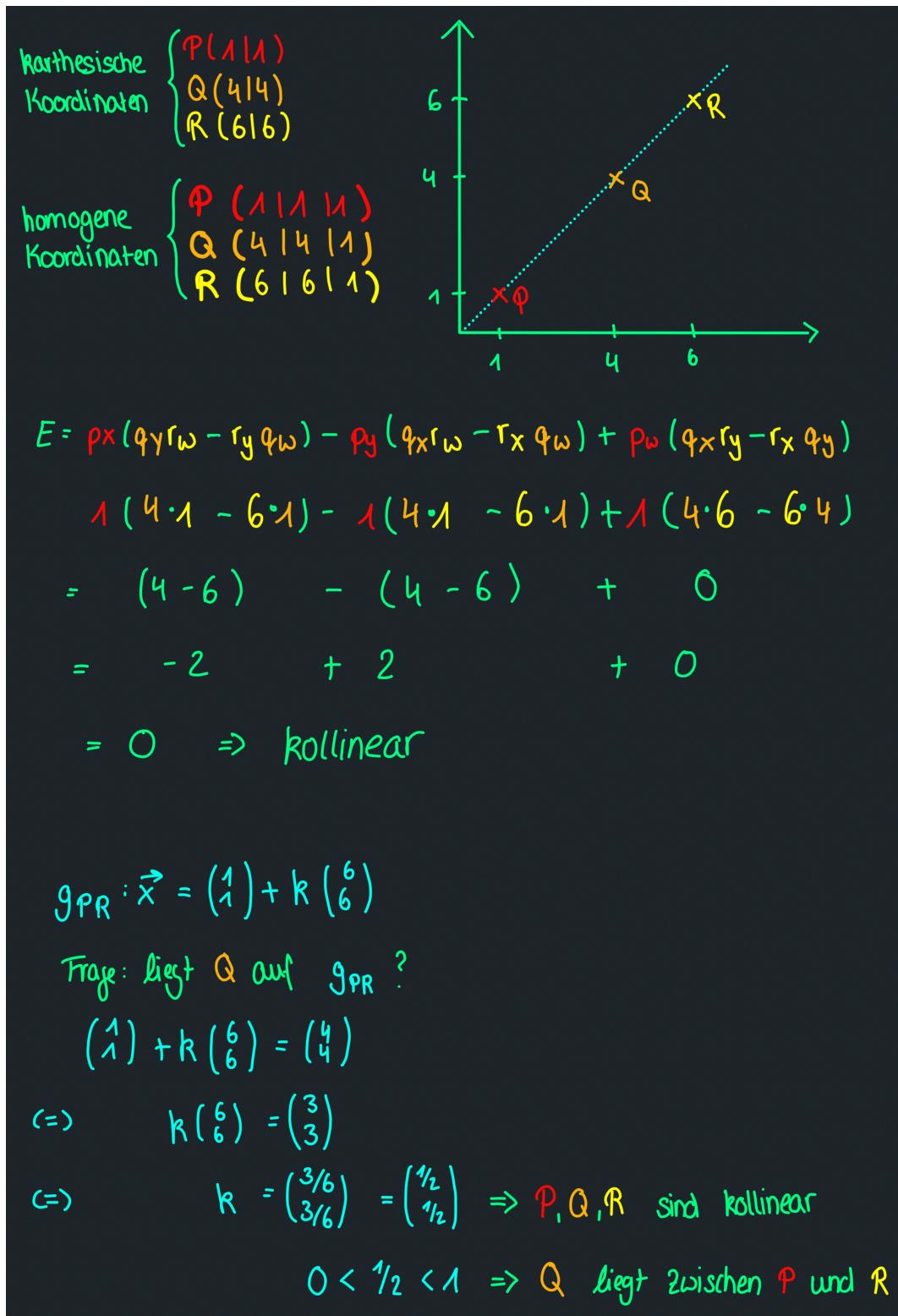


Abbildung I.4: Determinante = 0 → Kollineare Punkte

### I.c Test-Instanzen

Compile a set of  $n \approx 1000000$  test instances for which exact arithmetic exceeds built-in data types (`int` and `long`) and for which floating point approximation is not sufficient. Beneath others, also consider collinear points. Describe your approach.

Mit unserem [Instanz-Generator](#) haben wir 1 000 000 Punkte in homogenen Koordinaten [generiert](#). Dazu haben wir das `random` Python-Modul genutzt und um die Grenzen der exakten Arithmetik auszureißen, haben wir die Grenzen für `randint()` auf die Maximalgröße für `int` in Python gesetzt (In Python gibt es keinen `long`-type mehr, `int` können so lang sein, wie der Speicher hergibt).

Um sicherzustellen, dass auch kollinare Punkte korrekt behandelt werden, wollten wir spezifisch erzeugte [Triple von kollinearen Punkten](#) zu den Daten hinzufügen, haben dies aber zeitlich nicht mehr geschafft. Unser Test in [Teilaufgabe I.b](#) zeigt aber, das Kollinearität generell kein Problem für unsere Implementierung sein sollte, da sie bereits korrekt festgestellt wurde.

### I.d Evaluierung der Implementierung

Design and run an experiment to evaluate the efficiency of your implementation. Quantify the speedup between the exact evaluation and the floating point filter.

[Abbildung I.5](#) zeigt leider deutlich, dass unser Floating-Point Filter mit  $\approx 34$  Sekunden Laufzeit weit hinter den anderen beiden Routinen zurück liegt.

```
Timer for Floating Point Arithmetics - CPU-Time: 11.575040000004208, Wall-Time: 11.497345685958862
Timer for Exact Arithmetics - CPU-Time: 16.242589000002255, Wall-Time: 16.15915012359619
Timer for Floating Point Filter - CPU-Time: 33.74437799999898, Wall-Time: 33.671862840652466
```

**Abbildung I.5:** Vergleich der Laufzeiten aller drei Routinen

Wir sind uns nicht ganz sicher, ob das an unserer Wahl für die Grenzen der zufälligen Koordinaten liegt oder an unserer Implementierung - dann müsste der Fehler ja irgendwo bei der Berechnung von  $mes_E$  oder  $ind_E$  liegen, weil der Filter ja auch länger braucht als die beiden anderen Routinen zusammen und er ja immer entweder die eine oder die andere von beiden nutzt. Irgendwas läuft da schief.

## II Competitive Analysis - Ski Rental Problem

On your winter vacation, you want to go skiing for some unknown number  $D$  of days. Assume that renting a pair of skis costs 1 unit per day, while buying skis costs  $b$  units. Every day, you have to decide whether to continue renting skis for one more day or to buy skis.

### II.a Optimale Offline-Strategie bei bekanntem $D$

Knowing the value of  $D$ , what would be an optimal offline strategy? On which cost?

Eine optimale Offline-Strategie wäre, am ersten Urlaubstag zu überprüfen, ob die akkumulierte Miete für alle Tage  $D$  kleiner wäre, als der Kaufpreis  $b$ . Ich miete jeden Tag Skier, falls gilt:

$$D \cdot 1 < b \quad (\text{II.1})$$

Ansonsten kaufe ich bereits am ersten Tag Skier für  $b$  Einheiten und habe so immer die optimalen (minimalen) Kosten von maximal  $b$ .

### II.b Deterministische Online-Strategie

Design a deterministic online strategy that is at most twice as costly as an optimal one.

Wenn ich  $D$  nicht kenne, dann stelle ich jeden Tag folgende Überlegung an: Wenn ich heute die Miete bezahlen würde, hätte ich dann insgesamt mehr bezahlt als  $b$ ? An dem Tag  $d_e$ , an dem dies der Fall ist, kaufe ich Skier, statt sie zu mieten. Das bedeutet, ich habe  $e - 1$  mal Miete bezahlt plus nun den Kaufpreis  $b$ . Damit entstehen Kosten von höchstens

$$b + (e \cdot 1 - 1) < 2b \quad (\text{II.2})$$

Die Gesamtkosten sind also immer höchstens zweimal so hoch wie die Kosten der optimalen Lösung, bei der ich höchstens  $b$  bezahle.

Diese Strategie wird auch »break-even« Algorithmus genannt und deshalb im Folgenden mit »BE« abgekürzt.

## II.c Competitive-Ratio

Show that your strategy has a competitive ratio of 2 or less.

Bei der kompetitiven Analyse wird ein Online-Algorithmus mit einer optimalen Offline-Lösung verglichen. Das Ziel ist zu zeigen, dass

$$C_{BE}(\sigma) \leq 2 \cdot C_{OPT}(\sigma) \quad (\text{II.3})$$

Wir definieren  $c_i$  als die Kosten für die  $i$ -te Operation und  $D_i$  als das Ergebnis dieser Operation. Eine Operation ist die Abfrage, ob wir mit der heutigen Miete  $b$  überschritten würden. Dafür müssen wir eine Summe berechnen und diese anschließend mit einem Wert vergleichen, das geht jeden Tag in konstanter Zeit. Die Potentialfunktion  $\Phi(D_i)$  wählen wir als die Kosten an Tag  $i$ . Dann ist  $\Phi(D_0) = 0$  und  $\Phi(D_i) > 0$ , da wir ab Tag 1 mindestens 1 Unit als Miete zahlen.

Die gesamten amortisierten Kosten für  $n$  Tage sind:

$$\sum_{i=1}^n a_i = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0) \quad (\text{II.4})$$

Die Potential-Differenz ist immer = den Mietkosten, also 1:

$$\Phi(D_i) - \Phi(D_{i-1}) = 1 \quad (\text{II.5})$$

Damit ergibt sich für die obere Schranke:

$$\begin{aligned} a_{BE}(\sigma) &= \sum_{i=1}^n a_{BE}(i) \\ &= \sum_{i=1}^n c_{BE}(i) + \Phi(D_i) - \Phi(D_{i-1}) \\ &= \sum_{i=1}^n c_{BE}(i) + 1 &< 2 \sum_{i=1}^n C_{OPT}(i) \end{aligned} \quad (\text{II.6})$$

□

Damit ist die break-even Strategie höchstens zweimal so schlecht wie die optimale Strategie (was man intuitiv schon bei [Teilaufgabe II.b](#) gesehen hat).