

Facharbeit Informatik

Joel Mantik

9. März 2023

Inhaltsverzeichnis

1	Einleitung	2
2	Theoretische Grundlagen	3
2.1	Lineare Gleichungssysteme	3
2.2	Gaußsches Eliminationsverfahren	4
3	Implementierung des Algorithmus	6
3.1	Schritte des Verfahrens	6
3.2	Erläuterung des Quellcodes	7
3.3	Beispiel Rechnungen	7
3.4	Schwächen des Algorithmus	7
4	Anwendungsbereiche des Gaußschen-Eliminationsverfahrens in der Informatik	8
4.1	Kryptografie	8
4.2	Computergrafik	8
4.3	Datenanalyse	9
5	Abwägungen	10
5.1	Vorteile im Vergleich zu anderen Methoden	10
5.2	Nachteile im Vergleich zu anderen Methoden	10
6	Fazit	11
7	Anhang	12

Kapitel 1

Einleitung

”The simplest model in applied mathematics is a system of linear equations. It is also by far the most important.”

GILBERT STRANG

Der Gauß-Algorithmus ist eins der wichtigsten Lösungsverfahren zum Lösen lineare r Gleichungssysteme. Er spielt eine tragende Rolle in vielen Bereichen der Mathematik und ist dennoch recht unkompliziert. Aufgrund der Wichtigkeit, habe ich dazu entschieden, den Algorithmus in dieser Facharbeit zu implementieren, zu analysieren, und Anwendungsmöglichkeiten aufzuzeigen.

Kapitel 2

Theoretische Grundlagen

Der Gauß-Algorithmus ist ein Algorithmus, welcher beim Lösen von linearen Gleichungssystemen zum Einsatz kommt. Im folgenden Kapitel wird die mathematische Theorie erläutert und die Definitionen genannt, welche die Grundlage für die spätere Implementierung sind. Die folgenden Definitionen sind sinngemäß aus [\[Gra21\]](#) entnommen.

2.1 Lineare Gleichungssysteme

Ein lineares Gleichungssystem ist eine Sammlung von Gleichungen, in denen jede Unbekannte mit höchstens dem ersten Grad vorkommt. Es kann in der Form $Ax = b$ geschrieben werden, wobei A eine $m \times n$ Matrix ist, x ein n -dimensionaler Vektor von Unbekannten und b ein m -dimensionaler Vektor von Konstanten ist. Ein allgemeines lineares Gleichungssystem lässt sich wie folgt definieren. :

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n & = & b_2 \\ & \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n & = & b_m \end{array} \quad (2.1)$$

Das Ziel eines solchen linearen Gleichungssystems ist es, eine Lösung für x zu finden, die alle Gleichungen erfüllt. Hierbei gibt es drei Arten von Lösungen:

1. Das Gleichungssystem hat genau *eine* Lösung; es gibt genau eine Lösung, welche alle Gleichungen im System erfüllt. Die Lösungsmenge ist z. B. : $\mathbb{L} = \{(x, y, z) | (1, 2, 3)\}$.
2. Das Gleichungssystem hat *keine* Lösung, wenn es keine Lösung gibt, die alle Gleichungen erfüllt. Die Lösungsmenge ist eine leere Menge: $\mathbb{L} = \{\}$.
3. Das Gleichungssystem hat *unendlich viele* Lösungen, wenn es mehrere Lösungen gibt, die alle Gleichungen im System erfüllen. Hierbei sind die verschiedenen Variablen voneinander abhängig. Die Lösungsmenge sieht beispielsweise wie folgt aus:
 $\mathbb{L} = \{(x, y, z) | (x = y * z, y \in \mathbb{R}, z \in \mathbb{R})\}$.

2.2 Gaußsches Eliminationsverfahren

Für die folgenden Definitionen und Erläuterungen wurde sinngemäß auf [Fis14] zurückgegriffen.

Gegeben sei das Allgemeine Lineare Gleichungssystem 2.1. Gesucht ist nun die Menge der $(x_1, \dots, x_n) \in \mathbb{R}^n$, die alle Gleichungen erfüllen. Dies erreicht man indem man folgendermaßen vorgeht.:

1. Die erweiterte Koeffizientenmatrix (A, b) aufschreiben.:

$$(A, b) := \begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} & b_m \end{pmatrix} \quad (2.2)$$

2. A durch elementare Zeilentransformationen, also Vertauschen von Zeilen, Multiplikation einer Zeile mit einer Zahl $\neq 0$ oder Addition des Vielfachen von einer Zeile zu einer anderen Zeile auf Zeilenstufenform bringen. Eine $m * n$ -Matrix A heißt in *Zeilenstufenform*, wenn sie von der folgenden Form ist:

$$\left(\begin{array}{cccccccccccccccc|c} 0 & \dots & 0 & 1 & * & \dots & * & 0 & * & \dots & * & 0 & * & \dots & * & 0 & * & \dots & * \\ & & & & & & & 1 & * & \dots & * & 0 & * & \dots & * & 0 & * & \dots & * \\ & & & & & & & & & & & 1 & * & \dots & * & 0 & * & \dots & * \\ & & & & & & & & & & & & & & & 1 & * & \dots & * \\ & & & & & & \ominus & & & & & & & & & & & & \vdots \end{array} \right)$$

Dabei steht \ominus für eine beliebige Zahl, und die freien Plätze sind alle mit Nullen besetzt. Der erste von Null verschiedene Eintrag in jeder Zeile ist 1. Dieser Eintrag wird das Pivotelement der Zeile genannt. Das

Pivotelement der $(i + 1)$ -ten Zeile steht immer rechts des Pivotelementes der i -ten Zeile, und alle Einträge oberhalb eines Pivotelementes sind gleich Null.

3. Nun lassen sich durch Rücksubstitution die Werte für die Variablen ermitteln. Man teilt die letzte Spalte durch den Wert des Koeffizienten, so dass die Variable alleine steht. Der gefundene Wert wird dann in die nächst höhere Zeile eingesetzt, dann wird analog zum ersten Schritt vorgegangen.

Kapitel 3

Implementierung des Algorithmus

Für die im Folgenden dargelegte Implementation und Analyse des Algorithmus wurde die Programmiersprache "Java" verwendet.

3.1 Schritte des Verfahrens

Grundlegend lässt sich der Algorithmus durch dieselben Schritte wie in 2.2 implementieren, wobei man diese allerdings noch feiner unterteilen muss.:

1. Eingabe: Ein lineares Gleichungssystem $Ax = b$ mit einer $n * n$ Matrix A und einem n -dimensionalem Vektor b .
2. Pivotisierung: Finden des größten Elements a_{ki} in der Spalte von A unterhalb der Diagonalen und tauschen der Zeilen k und i von A und B .
3. Elimination: Bei der Elimination wird eine Zeile oder das Vielfache der Zeile, (in diesem Algorithmus von der Pivotisierung bestimmt), zur nächsten Zeile addiert, so dass ein Koeffizient den Wert 0 annimmt.
4. Rückwärtssubstitution:

Der Aufbau des Algorithmus wird in folgendem Diagramm deutlich:

Gaussalgorithmus
+ datei: Path + koeff: double[][] + tuple: double[]
- einlesen(): void - initialisieren(): void - pivotisieren(): void - substitution(): void

Abbildung 3.1: Implementationsdiagramm der Klasse Gauß-Algorithmus

3.2 Erläuterung des Quellcodes

3.3 Beispiel Rechnungen

3.4 Schwächen des Algorithmus

Der Algorithmus in seiner Form hat Schwächen aufzuweisen: Zum einen kann es durch die Verwendung vom Datentyp Double und die mehrfache Rechnung mit denselben Werten zu Rundungsfehlern kommen. Zum anderen hat der Algorithmus bei komplizierten Matritzen mit vielen Zeilen und Spalten eine große Laufzeit, die Worstcase Laufzeit würde aufgrund von doppelten for-Schleifen $O(n * n)$ betragen.

Kapitel 4

Anwendungsbereiche des Gaußschen- Eliminationsverfahrens in der Informatik

Der Algorithmus spielt in der Informatik aufgrund seiner Simplität eine tragende Rolle in vielen Teilbereichen, wie der Kryptografie, der Computergrafik oder der Datenanalyse. Die Anwendungsmöglichkeiten, in diesen werden im Folgenden erläutert.

4.1 Kryptografie

Das Gauss'sche Eliminationsverfahren kann verwendet werden, um modulare Gleichungen zu lösen, die in der Kryptografie häufig auftreten. Zum Beispiel wird das Verfahren bei der Berechnung von Schlüsseln in asymmetrischen Kryptosystemen wie RSA eingesetzt. Hierbei wird mit dem Gaußalgorithmus eine Primfaktorzerlegung durchgeführt, die dann verwendet um Schlüssel in asymmetrischen Kryptosystemen zu verwenden.

4.2 Computergrafik

In der 3D-Computergrafik werden 4×4 -Matrizen verwendet, um Objekte im Raum zu transformieren. Diese Matrizen enthalten Informationen über Translationen, Rotationen und Skalierungen von Objekten. Das Gauss'sche Eliminationsverfahren wird verwendet, um die inverse Matrix zu berechnen, die dann verwendet wird, um die Transformationsoperationen umzukehren. Die inverse Matrix wird auch dazu verwendet, um Normale von 3D-Objekten zu transformieren, um sie in eine konsistente Richtung zu bringen, was wichtig ist für Beleuchtungs- und Schattierungsberechnungen.

4.3 Datenanalyse

Das Gauss'sche Eliminationsverfahren wird in der linearen Regression eingesetzt, um die beste Anpassung einer linearen Funktion an gegebene Daten zu finden. Es wird verwendet, um die Parameter der Funktion zu berechnen, die am besten zu den Daten passen, indem es die Summe der Quadrate der Fehler minimiert. Die lineare Regression ist ein wichtiges Werkzeug in der Datenanalyse, um Beziehungen zwischen Variablen zu identifizieren und Vorhersagen über zukünftige Daten zu treffen. Das Verfahren kann auch dazu verwendet werden, um die Koeffizienten eines linearen Gleichungssystems zu lösen, das in der Datenanalyse oder der Signalverarbeitung verwendet wird.

Kapitel 5

Abwägungen

5.1 Vorteile im Vergleich zu anderen Methoden

5.2 Nachteile im Vergleich zu anderen Methoden

Kapitel 6

Fazit

Kapitel 7

Anhang

```
1
2 import java.nio.charset.StandardCharsets;
3 import java.nio.file.Files;
4 import java.nio.file.Path;
5 import java.nio.file.Paths;
6 import java.io.IOException;
7 import java.util.List;
8
9 public class Gauss {
10     static Path datei = Paths.get("./input1.txt"); // Einlesen
    der Datei
11     static double[][] koeff = new double[3][4]; // Initialisierung
    der erweiterten Koeffizienten Matrix
12     static double[][] solMatrix = new double[3][4]; //
    Initialisierung der Loesungsmatrix
13
14     public static void main(String[] args) throws IOException {
    // Main Methode welche alle Methoden ausfuehrt
15         einlesen();
16         ausgabe(koeff, "Eingangsmatrix");
17         gaussAlgo1();
18         ausgabe(solMatrix, "Loesungsmatrix");
19     }
20
21     private static void einlesen() throws IOException { //
    Einlesen der Koeffizienten aus Datei
22         List<String> f = Files.readAllLines(datei,
    StandardCharsets.UTF_8);
23         int zeilenIndex = 0;
24         for (int i = 0; i < f.size(); i++) {
25             String zeile = f.get(i);
26             if (zeile.isEmpty() || zeile.charAt(0) == '#') { //
    Kommentare und Whitespaces der input Datei werden ignoriert
27                 continue;
28             }
29
30             String[] zeilenElemente = zeile.replace(',', ' ').
    split("\\s+"); // Kommata werden zu Punkten
```

```

31         for (int spaltenIndex = 0; spaltenIndex <
zeilenElemente.length; spaltenIndex++) {
32             String element = zeilenElemente[spaltenIndex];
33             double wert = Double.parseDouble(element);
34             koeff[zeilenIndex][spaltenIndex] = wert; // Der
Koeffizientenmatrix werden die double Werte zugewiesen, welche
in wert gespeichert wurden
35         }
36
37         zeilenIndex++;
38     }
39 }
40 private static void ausgabe(double[][] mx, String matrixName)
{ // Ausgabe der Matrizen
41
42     System.out.println("Die " + matrixName + ":");
43     for (int j = 0; j < mx.length; j++) {
44         for (int k = 0; k < mx[j].length; k++) {
45             System.out.print(mx[j][k] + " ");
46         }
47         System.out.println();
48     }
49 }
50
51 private static void gaussAlgo1() {
52     copyLine(0); // Erste Zeile der Originalmatrix wird in
Loesungs Matrix kopiert
53     copyLine(1);
54     copyLine(2);
55     for (int zeile = 0; zeile < solMatrix.length - 1; zeile
++) {
56         int maxZeile = zeile;
57         for (int i = zeile + 1; i < solMatrix.length; i++) { //
Diese Schleife sucht in der aktuellen Spalte (durch die
Variable zeile indiziert) nach dem Element mit dem groessten
absoluten Wert und merkt sich die Zeilennummer dieses Elements
in der Variable maxZeile.
58             if (Math.abs(solMatrix[i][zeile]) > Math.abs(
solMatrix[maxZeile][zeile])) {
59                 maxZeile = i;
60             }
61         }
62         if (maxZeile != zeile) { // Wenn das Element mit dem
groessten absoluten Wert in einer anderen Zeile als der
aktuellen Zeile gefunden wurde, werden die beiden Zeilen
vertauscht. Dies ist die Pivotisierung.
63             double[] temp = solMatrix[zeile];
64             solMatrix[zeile] = solMatrix[maxZeile];
65             solMatrix[maxZeile] = temp;
66         }
67
68         for (int i = zeile + 1; i < solMatrix.length; i++) { //
Die Koeffizienten werden durch  $x = -b/a$  0
69             double factor = -solMatrix[i][zeile] / solMatrix[
zeile][zeile];

```

```

70         multiplyAndAdd(zeile, i, factor);
71     }
72 }
73
74     for(int zeilen = solMatrix.length-1; zeilen >= 0; zeilen
--){
75         double diagonale = solMatrix[zeilen][zeilen];
76
77         for(int spalte = 0; spalte < solMatrix[zeilen].length
; spalte++){
78             solMatrix[zeilen][spalte] /= diagonale;
79
80         }
81         double result = solMatrix[zeilen][solMatrix[zeilen].
length - 1]; // Durch ruecksubstitution werden die gefundenen
Werte RUECKWAERTS in die uebere Zeile eingesetzt und so x und
y errechnet
82         for(int rueckZeilen = zeilen - 1; rueckZeilen >= 0;
rueckZeilen--){
83             solMatrix[rueckZeilen][solMatrix[zeilen].length -
1] -= result * solMatrix[rueckZeilen][zeilen];
84             solMatrix[rueckZeilen][zeilen] = 0.0;
85         }
86     }
87 }
88
89
90     private static void copyLine(int line){ // Diese Methode
setzt die Endgleichung = erw. Koeffizientenmatrix
91         for(int i = 0; i < koef[f[line].length; i++) {
92             solMatrix[line][i] = koef[f[line][i];
93         }
94     }
95
96     private static void multiplyAndAdd(int lineOne, int lineTwo,
double factor) { // x = -b/a wird durchgefuehrt
97         for(int spalten = 0; spalten < solMatrix[lineOne].length;
spalten++){
98             solMatrix[lineTwo][spalten] = (solMatrix[lineOne][
spalten] * factor) + solMatrix[lineTwo][spalten];
99         }
100
101     }
102 }

```

Listing 7.1: Quellcode des Gauß-Algorithmus

Literatur

- [Fis14] Gerd Fischer. *Lineare Algebra: Eine Einführung für Studien Anfänger*. Springer Spektrum, 2014.
- [Gra21] Günther Gramlich. *Lineare Algebra: Eine Einführung*. Carl Hanser Verlag GmbH Co KG, 2021.