Facharbeit Informatik

Joel Mantik

14. März 2023

Inhaltsverzeichnis

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 linearer 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.

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 $A_x = b$ geschrieben werden, wobei A eine mxn 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.:[Gra21]

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned} \tag{2.1}$$

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: $\mathbb{L} = \emptyset$.

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:
   $\mathbb{L} = \{(x, y, z)|(x = ay + z, y \in \mathbb{R}, z \in \mathbb{R})\}$.

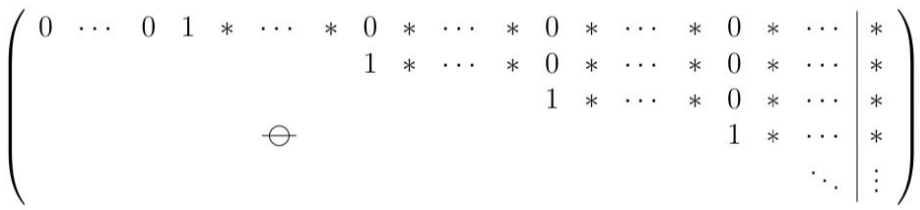
2.2 Gaußsches Eliminationsverfahren

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

1. Die erweiterte Koeffizientenmatrix (A, b) aufschreiben. [Fis14]:

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

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 mxn-Matrix A heißt in *Zeilenstufenform*, wenn sie von der folgenden Form ist:



Dabei steht der Stern 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 Pivot-Element der Zeile genannt. Das Pivot-Element der $(i + 1)$-ten Zeile steht immer rechts des Pivot-Elements der i-ten Zeile, und alle Einträge oberhalb eines Pivot-Elements sind gleich Null. [Zwe22]

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 Erläuterung des Quellcodes

Der grundlegende Aufbau des Programms lässt sich an folgendem Implementationsdiagramm erkennen.:

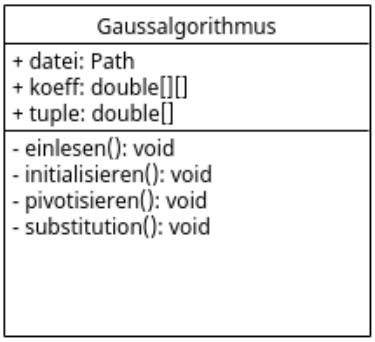


Abbildung 3.1: Implementationsdiagramm der Klasse Gauß-Algorithmus

Die Methode einlesen() liest die Koeffizienten einer Matrix aus einer txt-Datei ein, dabei filtert sie jegliche Kommentare sowie Leerzeichen aus dieser heraus, und speichert die Koeffizienten in dem Array koeff[][]. Die ausgabe() gibt sowohl die Eingangsmatrix, als auch die Lösungsmatrix auf der Konsole aus, indem über die beiden Arrays koeff[][] und mx[][] iteriert wird und ihre Indizes ausgegeben werden. Die Methode multiplyandadd() manipuliert die Zeilen der Eingangsmatrix, indem sie die Zeilen dieser mit dem Faktor multipliziert, der benötigt wird, um bei der Addition mit der nächsten Zeile eine Null herzustellen. Durch die Forschleife wird jede Spalte der angegeben Zeile durchlaufen und der Parameter linetwo aktualisiert. Die Methode copyline() kopiert die Eingangsmatrix in eine Hilfsmatrix, damit die Eingangsmatrix erhalten bleibt und am Ende ausgeben werden kann. Die Methode gaussAlgo() ist das Herzstück des Programms.i

3.2 Beispiel Rechnungen

Kapitel 4

Abwägungen

4.1 Vorteile des Algorithmus

Der Gauß-Algorithmus hat klar den Vorteil das er bei Gleichungssystemen mit wenig Koeffizienten sehr effektiv ist, d.h er kann die Lösung schnell und einfach berechnen. Auch ist der Algorithmus in der Programmierung relativ leicht umzusetzen, da er nach einfachen Schemata funktioniert, es werden wenig Schritte benötigt. Ein weiterer Vorteil ist, dass der Algorithmus auch erweitert werden kann um beispielsweise invertierte Matrizen und/oder Eigenwerte und Eigenvektoren von Matrizen zu berechnen.

4.2 Nachteile des Algorithmus

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 Martizen mit vielen Zeilen und Spalten eine große Laufzeit, die Worstcase Laufzeit würde aufgrund von doppelten for-Schleifen $O(nxn)$ betragen. Nachteile des Algorithmus sind zum einen, dass bei schlecht konditionierten Gleichungssystemen Rundungsfehler auftreten können. Dies sieht man auch in der vorliegenden Implementation, wobei durch die Verwendung vom Datentyp double und dem rechnen mit denselben gerundeten Zahlen

noch größere Rundungsfehler entstehen können. Außerdem kann der Algorithmus bei vielen Koeffizienten eine sehr hohe Rechenleistung in Anspruch nehmen.

Kapitel 5

Anwendungsbereiche des Gaußschen-Eliminationsverfahrens in der Informatik

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

5.1 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. Mehr Informationen in [KK10] Kapitel 11.4.3.

5.2 Computergrafik

In der 3D-Computergrafik werden 4x4-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.

5.3 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 6

Fazit

Kapitel 7

Anhang

```java
/**
 * @author Joel Mantik
 */
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.IOException;
import java.util.List;

public class Gauss {
    static String datei; // der Parameter der Datei
    static double[][] koeff = new double[3][4]; // Initialierung der erweiterten Koeffizienten Matrix
    static double[][] solMatrix = new double[3][4]; // Initialierung der Loesungsmatrix
    static String[] zeilenElemente;
    static int countSpalten;
    static int countZeilen;

    /**
     * main Methode in welcher alle Methoden ausgefuehrt werden
     * @param args liest die Datei mit der Matrix ein
     */
```

```
23      public static void main(String[] args) throws IOException {
    // Main Methode welche alle Methoden ausfuehrt
24          /*
25           * Hier wird der Dateiname als Parameter eingelesen
26           */
27          if (args.length < 1) {
28              System.out.println("Bitte Dateiname als Parameter
    uebergeben!");
29              return;
30          } else {
31              datei = args[0];
32          }
33
34          einlesen();
35          ausgabe(koeff, "Eingangsmatrix");
36          gaussAlgo1();
37          ausgabe(solMatrix, "Loesungsmatrix");
38      }
39
40      /**
41       * Diese Methode liest die Koeffizienten aus der Datei ein
42       */
43
44      private static void einlesen() throws IOException {
45
46          List<String> f = Files.readAllLines(Paths.get(datei));
47          int zeilenIndex = 0;
48          for (int i = 0; i < f.size(); i++) {
49              String zeile = f.get(i);
50               // Kommentare und Whitespaces der input Datei werden
     ignoriert
51              if (zeile.isEmpty() || zeile.charAt(0) == '#') {
52                  continue;
53              }
54
```

```
55              zeilenElemente = zeile.replace(',', '.').split("\\s+"
    ); // Kommata werden zu Punkten
56              for (int spaltenIndex = 0; spaltenIndex <
    zeilenElemente.length; spaltenIndex++) {
57                  String element = zeilenElemente[spaltenIndex];
58                  double wert = Double.parseDouble(element);
59                  // Der Koeffizientenmatrix werden die double
    Werte zugewiesen, welche in wert gespeichert wurden
60                  koeff[zeilenIndex][spaltenIndex] = wert;
61
62              }
63
64              zeilenIndex++;
65          }
66          // Speichert Zeilen und Spalten Anzahl
67          countSpalten = zeilenElemente.length;
68          countZeilen = koeff.length;
69          System.out.println(countSpalten);
70          System.out.println(countZeilen);
71      }
72
73      /**
74       * Methode ausgabe gibt die Eingangsmatrix und die
    Loesungsmatrix aus
75       * @param mx nimmt die Eingangsmatrix oder die Loesungsmatrix
     an
76       * @param matrixName nimmt den Namen der Matrix an
77       */
78
79      private static void ausgabe(double[][] mx, String matrixName)
    { // Ausgabe der Matrizen
80
81          System.out.println("Die " + matrixName + ":");
82          for (int j = 0; j < mx.length; j++) {
83              for (int k = 0; k < mx[j].length; k++) {
84                  System.out.print(mx[j][k] + " ");
```

```
85              }
86              System.out.println();
87          }
88      }
89
90
91      /**
92       * Die Methode gaussAlgo fuehrt den eigentlichen Algorithmus
    aus
93       */
94      private static void gaussAlgo1() {
95          copyLine(0);// Erste Zeile der Originalmatrix wird in
    Loesungs Matrix kopiert
96          copyLine(1);
97          copyLine(2);
98          for (int zeile = 0; zeile < solMatrix.length - 1; zeile
    ++) {
99          int maxZeile = zeile;
100
101         /*
102          *  Diese Schleife sucht in der aktuellen Spalte (durch
    die Variable zeile indiziert)
103          *  nach dem Element mit dem groessten absoluten Wert und
     merkt sich die Zeilennummer dieses Elements in der Variable
    maxZeile.
104          */
105         for (int i = zeile + 1; i < solMatrix.length; i++) {
106             if (Math.abs(solMatrix[i][zeile]) > Math.abs(
    solMatrix[maxZeile][zeile])) {
107                 maxZeile = i;
108             }
109         }
110         /*
111          * Wenn das Element mit dem groessten absoluten Wert in
    einer anderen Zeile als in der aktuellen Zeile gefunden wurde,
```

```
112         *werden die beiden Zeilen vertauscht. Dies ist die
    Pivotisierung.
113          */
114         if (maxZeile != zeile) {
115             double[] temp = solMatrix[zeile];
116             solMatrix[zeile] = solMatrix[maxZeile];
117             solMatrix[maxZeile] = temp;
118         }
119 
120         // Die Koeffizienten werden durch x = -b/a 0
121         for (int i = zeile + 1; i < solMatrix.length; i++) {
122             double factor = -solMatrix[i][zeile] / solMatrix[
    zeile][zeile];
123             multiplyAndAdd(zeile, i, factor);
124         }
125         }
126 
127         for(int zeilen = solMatrix.length-1; zeilen >= 0; zeilen
    --){
128             double diagonale = solMatrix[zeilen][zeilen];
129 
130             for(int spalte = 0; spalte < solMatrix[zeilen].length
    ; spalte++){
131                 solMatrix[zeilen][spalte] /= diagonale;
132             }
133             double result = solMatrix[zeilen][solMatrix[zeilen].
    length - 1];
134             /*
135              *  Durch ruecksubstitution werden die gefundenen
    Werte RUECKWAERTS in die uebere Zeile eingesetzt
136              *  und so x und y errechnet
137             */
138             for(int rueckZeilen = zeilen - 1; rueckZeilen >= 0;
    rueckZeilen--){
139                 solMatrix[rueckZeilen][solMatrix[zeilen].length -
     1] -= result * solMatrix[rueckZeilen][zeilen];
```

```
140                 solMatrix[rueckZeilen][zeilen] = 0.0;
141             }
142         }
143     }
144 
145     /**
146      * copyline setzt die Endlgeichung = erw. Koeffizientenmatrix
147      */
148     private static void copyLine(int line){
149         for(int i = 0; i < koeff[line].length; i++) {
150             solMatrix[line][i] = koeff[line][i];
151         }
152     }
153 
154     /**
155      * multiplyAndAdd bringt die Koeffizienten auf 0
156      * @param lineOne  nimmt die 1. Zeile welche benutzt werden
    soll an
157      * @param lineTwo  nimmt die 2. Zeile welche benutzt werden
    soll an
158      * @param factor   der Wert mit dem multiplizert wird, so
    dass der Koeffzient 0 wird
159      */
160     private static void multiplyAndAdd(int lineOne, int lineTwo,
    double factor) { // x = -b/a wird durchgefuehrt
161         for(int spalten = 0; spalten < solMatrix[lineOne].length;
     spalten++){
162             solMatrix[lineTwo][spalten] = (solMatrix[lineOne][
    spalten] * factor) + solMatrix[lineTwo][spalten];
163         }
164 
165     }
166 }
```

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.

[KK10] Christian Karpfinger und Hubert Kiechle. *Kryptologie, Algebraische Methoden und Algorithmen*. Vieweg+Teubner, 2010.

[Zwe22] Prof. Dr. Sander Zwegers. *Lineare Algebra, Notizen zur Vorlesung*. Universitaet zu Koeln, Juli 2022.