Facharbeit Informatik

Joel Mantik

5. 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 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 $A_x = b$ geschrieben werden, wobei A eine $m * n$ 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. :

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

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

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 $(x_1, \ldots, x_n) \in \mathbb{R}^n$, die alle Gleichungen erfüllen. Dies erreicht man indem man folgendermaßen vorgeht.:

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

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

2. A auf Zeilenstufenform brinen, und dabei die Spalte b mit umformen, wobei man das λ-fache der i-ten Zeile zur k-ten Zeile addiert, hierbei muss $\lambda \neq 0$ und $i \neq k$ sein. Für eine 3x3 Matrix sähe die Zeilenstufenform am Ende so aus.:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a_{22} & a_{23} & b_2 \\ 0 & 0 & a_{33} & b_3 \end{pmatrix} \tag{2.3}$$

   Analog wäre die Zeilenstufenform für $m * n$ Matrizen, wobei mehr Nullen pro Zeile exisitieren müssten, um die Zeilenstufenform zu erreichen.

3. Nun lassen sich durch Rücksubstitution die Werte für die Variablen ermiteln. 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 die selben Schritte wie in **??** implementieren, wobei man diese allerdings noch feiner unterteilen muss.:

1. Eingabe: Ein lineares Gleichungssystem $A_x = b$ mit einer $n * n$ Matrix A und einem n-dimensionalem Vektor b.
2. Initialisierung:
3. 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.
4. Elimination:
5. Rückwärtssubstitution:

Der Aufbau des Algorithmus wird in folgendem Diagramm deutlich:

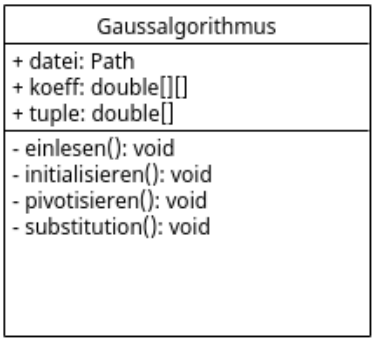

Abbildung 3.1: Implementationsdiagramm der Klasse Gauß-Algorithmus

3.2 Erläuterung des Quellcodes

3.3 Beispiel Rechungen

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 Martizzen mit vielen Zeilen und Spalten eine große Laufzeit, die Worstcase Laufzeit würde aufgrund von doppelten for-Schleifen $O(n * n)$ betragen.

Kapitel 4

Anwendungen des Gaußschen-Eliminationsverfahrens

4.1 Lösung von linearen Gleichungssystemen

4.2 Inversion von Matrizen

4.3 Beispielanwendungen

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]; // Initialierung
       der erweiterten Koeffizienten Matrix
12     static double[][] solMatrix = new double[3][4]; //
       Initialierung 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 < koeff[line].length; i++) {
92              solMatrix[line][i] = koeff[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.