Programmieren C++: Zweidim. Arrays, statisch und dynamisch

Klaus Kusche

Gauß'sches Gleichungslösen

Wir wollen ein Programm schreiben, das ein *lineares Gleichungssystem* in mehreren Variablen *nach der Gauß-Methode* löst. Wir verzichten dabei auf Klassen und Methoden: Wir verwenden ganz normale Arrays und ganz normale Funktionen.

Wir brauchen für ein Gleichungssystem mit n Variablen *zwei Arrays*.
Beide werden in `main` lokal angelegt.

- Das erste Array ist *zweidimensional*, es enthält n Zeilen mit je n+1 Spalten und repräsentiert das *Gleichungssystem*:

  Jede Zeile entspricht einer Gleichung. Die ersten n Spalten jeder Zeile sind die Koeffizienten der n Variablen in der jeweiligen Gleichung, die letzte Spalte enthält das konstante Glied (hinter dem =) der Gleichung.

- Das zweite Array ist *eindimensional*, es enthält n Werte, und zwar für jede der n Variablen den Wert der *Lösung*.

Weiters implementieren wir *folgende Funktionen*:

- `input`: Diese Funktion wird mit einer `istream`-Referenz, dem Gleichungssystem-Array und der Variablen-Anzahl als Parameter aufgerufen.

  Sie liest mit zwei geschachtelten Schleifen der Reihe nach n * (n+1) Zahlen aus dem angegebenen File in die entsprechenden Array-Elemente.

  `input` hat keinen Returnwert.

- `solve`: Diese Funktion wird mit dem Gleichungssystem-Array, dem Lösungs-Array und der Variablen-Anzahl als Parameter aufgerufen.

  Sie versucht, aus dem Gleichungssystem die Lösung zu berechnen (siehe Anhang), und liefert `true` bei Erfolg und `false` wenn es keine oder unendlich viele Lösungen gibt.

- `writeMatrix`: Diese Funktion wird mit dem Gleichungssystem-Array und der Variablen-Anzahl als Parameter aufgerufen und hat keinen Returnwert.

  Sie gibt das Gleichungssystem schön formatiert auf `cout` aus.

- `writeResult`: Diese Funktion wird mit dem Lösung-Array und der Variablen-Anzahl als Parameter aufgerufen und hat keinen Returnwert.

  Sie gibt die Lösung schön formatiert auf `cout` aus.

- Das *Hauptprogramm* wird mit einem *Filenamen* auf der Befehlszeile aufgerufen (prüfen!). Es öffnet diesen File zum Lesen (prüfen!)
  und liest daraus zuerst einmal die Anzahl der Variablen im Gleichungssystem (prüfen, ob sie größer 0 und für Variante 1. kleinergleich dem Maximum ist).

  Als nächstes ruft es zunächst einmal `input` mit diesem File auf
  und gibt das gelesene Gleichungssystem zur Kontrolle mit `writeMatrix` aus.

  Dann ruft es `solve` auf. Bei Erfolg wird das triangulierte Gleichungssystem sowie mittels `writeResult` die Lösung ausgegeben, bei Mißerfolg das teilweise triangulierte Gleichungssystem sowie eine Fehlermeldung.

Für die Fleißigen: Versuche, mittels `iomanip` eine *schöne Formatierung* (Spalten fixer Breite) zu erreichen!

1.) Lösung mit Arrays fixer Größe:

Im ersten Anlauf geben wir eine *maximale Arraygröße* (Variablenanzahl) als *globale Konstante* vor und dimensionieren unsere Arrays mit dieser konstanten Größe. Ist die eingelesene Variablenanzahl *größer*, soll das Programm mit einem *Fehler* enden, ist sie *kleiner*, lassen wir einen Teil der Arrays *unbenutzt*.

2.) Lösung mit variabel großen Arrays:

Wenn Punkt 1. funktioniert, bauen wir unser Programm so um, dass es die Arrays *genau so groß* anlegt, wie es für die eingelesene *Variablenanzahl* notwendig ist (und am Ende des Programmes auch wieder freigibt!).

Beim eindimensionalen Array für die Lösungswerte ist das mittels `new` einfach.

Das zweidimensionale Array für das Gleichungssystem müssen wir aber umbauen, denn erstens kann `new` nur mehrdimensionale Arrays anlegen, bei denen nur die vorderste Dimension variabel ist (alle anderen müssen fix sein), und zweitens kann man in C++ auch nur solche Arrays als Parameter an Funktionen übergeben, bei denen alle außer der ersten Dimension fixe Größe haben.

Wir müssen daher statt eines zweidimensionalen Arrays
zuerst ein *eindimensionales Array mit n* `double`*-Pointern* anlegen
und dann *mittels Schleife n eindimensionale* `double`*-Arrays* mit je n+1 Elementen anlegen und im ersten Array speichern
(und das Ganze in umgekehrter Reihenfolge am Programmende wieder freigeben).

Außer der Deklaration und dem Anlegen der Arrays müssen *nur die Parameter-Deklarationen* in den Funktionsprototypen *geändert* werden, der gesamte Rechen-Code sollte *unverändert* (ident zu 1.) bleiben!

Anhang: Das Gauß-Verfahren

Es besteht aus zwei Schritten:

1.) **Triangulation:**

Ziel ist es, ein beliebiges Gleichungssystem so umzuformen,
dass das linke untere Dreieck der Matrix nur 0-Werte enthält.
Mit anderen Worten: Die erste Gleichung enthält alle Variablen,
die zweite Gleichung alle außer der ersten Variable, usw. bis zur letzten Gleichung,
die nur mehr die letzte Variable und eine Konstante enthält.

Man geht dazu die Zeilen der Koeffizientenmatrix a von oben nach unten durch
und betrachtet in der i-ten Zeile die i-te Variable (Spalte):
Diese Spalte muss in allen Zeilen j unterhalb der Zeile i auf 0 gebracht werden,
indem man von jeder Zeile j ein Vielfaches der Zeile i abzieht.

Damit das klappt, darf der Wert $a_{i,i}$ nicht 0 sein. Ist er es doch,
muss man zuerst eine Zeile unterhalb von i suchen, die in Spalte i keine 0 hat,
und diese Zeile mit Zeile i vertauschen.

Haben sowohl Zeile i als auch alle Zeilen darunter in der Spalte i den Wert 0,
so ist das Gleichungssystem nicht eindeutig lösbar.

Hat man an der Stelle $a_{i,i}$ eine Zahl ungleich 0,
so geht man alle Zeilen j unterhalb von i durch,
berechnet für jede Zeile den Faktor t als $a_{j,i} / a_{i,i}$,
und zieht von jedem Element der Zeile j
das entspechende Element der Zeile i * t ab.

Es genügt sowohl beim Vertauschen als auch beim Subtrahieren,
die Elemente ab Spalte i zu betrachten:
Alle Elemente links von Spalte i sind ohnehin schon 0.

2.) **Lösungsberechnung:**

In diesem Schritt werden die Zeilen *von unten nach oben*
und die Variablen *von rechts nach links* durchgegangen:
In jeder Zeile i wird die Lösung für die Variable x_i berechnet und gespeichert.

Für Zeile i berechnet man zuerst einmal die Summe s der Produkte $x_k * a_{k,j}$
für alle Spalten k *rechts* von Spalte i (außer der letzten Spalte).

Dann berechnet man die Lösung von x_i als $(a_{i,n} - s) / a_{i,i}$
($a_{i,n}$ ist die letzte Spalte in Zeile i, also das konstante Glied).