BOK-Projekt Matrizenrechner

Jannis Herrmann, Paul von Drachenfels

26. Juli 2023

Einleitung

Das Ziel dieses Projektes war es, elementare Ergebnisse aus LA1 anzuwenden um einen Matrizenrechner für nxn-Matrizen zu schreiben. Implementiert wurde die Klasse "Matrix"mit folgenden Methoden und zusätzlichen Funktionen:

- Matrix-Multiplikation, sowie Addition durch das überladen der Operatoren '+' und '*'
- Die Terminal-Ausgabe einer Matrix mithilfe der 'show' Funktion
- Der Gauß-Algorithmus und dessen Anwendung zum Berechnen der Determinaten und der Inversen
- Das Einlesen und Ausgeben von Files um die Arbeit mit großen Matrizen zu vereinfachen

Außerdem haben wir den Operator "[]"überladen, um auf Zeilen der Matrix zugreifen zu können, sowie * mehrfach überladen um auch die Skalarmultiplikation zu implementieren.

Aufbau der Klasse

Unsere Implementierung hat drei Atribute:

- const int M, N. Diese repräsentieren die Anzahl der Zeilen bzw Spalten.
- double** matrix stellt den Inhalt der Matrix da. matrix einmal ausgewertet liefert einen double pointer der das Äquivalent zu einer Zeile der Matrix ist. Das zweite mal ausgewertet liefert einen Eintrag in der Zeile

Gauß-Algorithmus, Determinante und Inverese

In LA1 lernt man die rekursive Berechnung der Determinaten. Diese ist für sehr große Matrizen allerdings zu aufwendig. Der Gauß-Algorithmus macht das effizienter. Wir haben diesen so implementiert, dass nur Zeilenaddition verwendet wird. Diese Operation verändert die Determinante nicht.
Der Algorithmus beginnt in der ersten Spalte. Sind zunächst alle Eintrage der Spalte Null, so ist das kein Widerspruch zur Zeilenstufenform und wir gehen in die nächste Spalte. Gibt es einen Eintrag der ungleich Null ist, so wird dieser

in die aktive Zeile (hier noch 0) gebracht. Danach werden alle Einträge (aktive Zeile + 1, aktive Spalte) auf Null gesetzt und die aktive Zeile wird um eines erhöht. Usw.

Die Determinante von quadratischen Matrizen lässt sich nun über die Multiplikation der Diagonaleinträge berechnen.

Die Inverse kann man nun mithilfe des Gauss-Jordan-Verfahrens bestimmen. Dazu wird die Matrix durch Umformungen in die Form der Einheitsmatrix gebracht. Die gleichen Operationen werden dabei auf die Einheitsmatrix angewandt. Das Resulat dieser Operationen liefert dann die Inverse.
Wir haben es leider nicht geschafft ein Default-Argument vom Typ Matrix zu übergeben, weshalb wir den Gauss-Algorithmus überladen mussten. Dadruch ist redundanter Code entstanden, aber auf eine bessere Lösung sind wir nicht gekommen.
Für dieses Verfahren wurde auch die Methode $'punktspiegel'$ implementiert, um eine Matrix mithilfe des Gauss-Algorithmus von Zeilenstufenform in Diagonalgestallt zu bringen.

Eingabe und Ausgabe mithilfe von Files

Das eintragweise Eingeben von Matrizen (mit mat[i][j] =) ist sehr umständlich insbesondere für Dimensionen größer 3x3. Deshalb haben wir die Möglichkeit implementiert, ein Input File einzulesen. In diesem werden Matrizen nach einem bestimmten Format eingelesen und dann die angegebenen Rechenoperationen ausgeführt.

Das Format um Matrizen in das Inputfile zu schreiben umfasst die Dimension in der ersten Zeile, sowie den Inhalt der Matrix in den weiteren m Zeilen. Darauf folgt eine der Rechenoperationen '+', '*', 'gauss', 'inv', 'det', 'transp' und wenn die Operation es fordert eine weitere Matrix.

Wir mussten einige Hilfsfunktionen implementieren um diese Funktion zu implementieren:

- Die Funktion 'checkFormat' überprüft, ob die angegebene Dimension mit der Anzahl und Anordung der Einträge übereinstimmt. Dafür werden die nächsten beiden Funktionen verwendet.
- Die Funktion 'twoPositiveIntCheck' Überprüft, ob die Dimension richtig angegeben wurde
- 'nDoubleInString checkt, ob ein String n Einträge enthält, die zum Typ double konvertiert werden können

Diese Funktionen werden in der Readfile Funktion verwendet, um einen Pointer auf eine Matrix zurückzugeben.

Alle diese Funktionen werden dann in der 'calculate' Funktion verwendet. Beim Einlesen des Files werden std::vectoren angelegt die die Matrizen, bzw die Ope-

rationen enthalten. Die Rechnung wird ausgeführt und mithilfe der 'matrixToFile' Funktion in eine output.txt Datei geschrieben.

Erfahrungen und Schwierigkeit

Eine große Schwierigkeit, auf die wir beim Programmieren gestoßen sind, war die Rückgabe von Objekten. Von Python, waren wir es gewohnt, dass eine 'return Objekt' Anweisung reibungslos funktioniert. In C++ hat das jedoch nicht geklappt. Unsere Vermutung dazu war, dass das innerhalb einer Funktion erstellte Objekt das zurückgegeben werden soll, nach verlassen des Scopes nicht mehr existiert. Unsere Lösung für dieses Problem war es einen Matrixpointer mithilfe des Schlüsselwortes new zu erstellen. Wir waren uns länger uneinig, ob dieser Speicherplatz nun blockiert ist, aber Speicheranalysen mithilfe des Taskmanagers führten zu dem Ergebnis das der Speicher in der Tat wieder freigegeben wird.