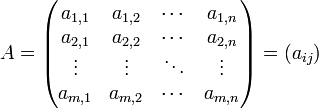
|  |
| --- |
| Fachhochschule Jena |
| Matrix |
| Informatik IIa - Projektarbeit |
|  |
| **Jürgen Döffinger, Florian Hilbrecht, Hannes Kuschick, Christoph Dick** |
| **06.05.2010** |

|  |
| --- |
| Dokumentation – Projektarbeit - Informatik IIa |



Inhaltsverzeichnis

[2 Analyse 4](#_Toc264930357)

[2.1 Aufgabenstellung 4](#_Toc264930358)

[2.2 Planung 5](#_Toc264930359)

[2.2.1 Klassenstruktur 5](#_Toc264930360)

[2.2.2 Personalplanung 7](#_Toc264930361)

[3 Klasse MatrixVector 8](#_Toc264930362)

[3.1 Attribute 8](#_Toc264930363)

[3.2 Methoden 8](#_Toc264930364)

[3.3 Ausnahmebehandlungen 9](#_Toc264930365)

[3.4 Test 10](#_Toc264930366)

[3.4.1 Testfälle 10](#_Toc264930367)

[3.4.2 Testprotokoll 14](#_Toc264930368)

[4 Klasse MatrixArithmetic 16](#_Toc264930369)

[4.1 Attribute 16](#_Toc264930370)

[4.2 Methoden 16](#_Toc264930371)

[4.3 Ausnahmebehandlungen 16](#_Toc264930372)

[4.4 Test 16](#_Toc264930373)

[5 Klasse MatrixMath 17](#_Toc264930374)

[5.1 Attribute 17](#_Toc264930375)

[5.2 Methoden 17](#_Toc264930376)

[5.3 Ausnahmebehandlung 17](#_Toc264930377)

[5.4 Test 17](#_Toc264930378)

[6 Klasse Matrix 18](#_Toc264930379)

[6.1 Attribute 18](#_Toc264930380)

[6.2 Methoden 18](#_Toc264930381)

[6.3 Ausnahmebehandlung 20](#_Toc264930382)

[6.4 Test 20](#_Toc264930383)

[6.4.1 Testfälle 20](#_Toc264930384)

[6.4.2 Testprotokoll 22](#_Toc264930385)

[8 Klasse MatrixException 24](#_Toc264930386)

[8.1 Attribute 24](#_Toc264930387)

[8.2 Methoden 24](#_Toc264930388)

[8.3 Ausnahmebehandlung 24](#_Toc264930389)

[8.4 Test 24](#_Toc264930390)

[9 Verzeichnisse 25](#_Toc264930391)

[9.1 Quellenverzeichnis 25](#_Toc264930392)

# Analyse

## Aufgabenstellung

**System zur algebraischen Manipulation von Matrizen**

Implementieren Sie ein generisches System zur algebraischen Behandlung von Matri­zen. Folgende Operationen sollen realisiert werden:

* Matrix-Addition, -Subtraktion und -Multiplikation,
* Skalarmultiplikation,
* Vektor-Multiplikation,
* Gauß-Elimination (Lösung von linearen Gleichungssystemen),
* Determinantenberechnung,
* Matrix-Inversion.

Implementieren Sie das System so, dass Matrizen verschiedener Dimension, auch nicht-quadratische behandelt werden können (für die Determinantenberechnung und die Matrixinversion sind die Matrizen als quadratisch vorausgesetzt). Implementieren Sie das System als Container-Klasse, d. h. unabhängig von den Datentypen der Matrixelemente, der Vektorelemente und der Skalare. Sie können dabei auch auf die Standard-Template-Library zurückgreifen (z. B. vector). Implementieren Sie die Funktionen der Matrix-Addition, -Subtraktion und -Multiplikation als überladene Operatoren +, − und ∗ Implementieren Sie die Ein- und Ausgabefunktionen für Matrizen und Vekroren durch ¨Überladen der Stream-Operatoren >> und <<.

Ihr System muss mit den elementaren Datentypen float und double und den in der Vorlesung und im Praktikum behandelten Klassen der Komplexen Zahlen und der Brüche funktionieren. Ein einfacher Beispiel-Test sähe z. B. folgendermaßen aus:

#include <iostream >

#include "Matrix.h" // Ihre Matrix-Klasse #include "Complex.h" // Komplexe Zahlen #include "Bruch.h" // Brüche

using namespace std;

int main() { // Test

// Fließkomma -Matrizen float f;

|  |  |  |  |
| --- | --- | --- | --- |
| // 3x4-, 4x2-, 3x3-Matrizen  Matrix <float > mf1(3,4) , mf2(4,2) , mf3; | | | |
| cin | >> f | >> mf1 >> | mf2; |
| cout | << | mf1 \* f \* | mf2 << endl; // (Skalar)Multiplikation |
| cout | << | mf3.det() | << endl; // Determinante |
| cout | << | mf3.hls() | << endl; // Lösung des homogenen LGS |

FH-Jena, FB ET/IT Übungen zu Informatik IIa SS 2010 Projekte

// Komplex-Matrizen

Complex c;

// 3x4-, 4x2-, 3x3-Matrizen

Matrix <Complex > mc1 (3,4), mc2(4,2) , mc3;

cin >> c >> mc1 >> mc2;

cout << mc1 \* c \* mc2; // (Skalar)Multiplikation cout << mc3.det() // Determinante

cout << mc3.hls() // Lösung des homogenen LGS

// Bruch-Matrizen // ...

}

Die hier gemachten Annahmen über die Konstruktoren und Member-Funktionen der Matrix-Klasse können Sie als Anregung für Ihre Implementierung auffassen.

## Planung

Während der Planung wurden folgende Phasen festgelegt:

1. Struktur der Klasse festlegen

2. Personalplanung

3. Codierung (nach welchen Verfahren wird was programmiert)

4. Qualitätssicherung / Test

### Klassenstruktur

Wie im Klassendiagramm (Bild 1) zu sehen ist, besteht das Projekt Matrix aus einer Basisklasse „MatrixVector“, zwei davon abgeleiteten Klassen „Matrixarithmetic“ und „MatrixMath“ und einer von „MatrixArithmetic“ und „MatrixMath“ abgeleiteten Klasse „Matrix“. Des Weiteren beinhaltet das Projekt „Matrix“ die Klasse „MatrixException“.

Dabei dient MatrixVector der Speicherung der Matrixelemente und deren Verarbeitung in Hinsicht auf Eingabe und Ausgabe. In dieser Klasse sind auch Abfragen zu verschiedenen Eigenschaften einer Matrix implementiert.

Die Klasse MatrixArithmetic enthält sämtliche arithmetischen Funktionen, wie Addition, Subtraktion, Multiplikation zweier Matrizen, mit einem Skalar und das Vektorprodukt.

In der Klasse MatrixMath sind Funktionen zur Bildung der transponierten Matrix, der Determinante, der adjunkten Matrix und der inversen Matrix umgesetzt.

Die Klasse Matrix führt alle zuvor genannten Klassen in dieser zusammen und beinhaltet den Gleichungslöser für lineare Gleichungssysteme.



Bild 1 Klassendiagramm[[1]](#footnote-1)

### Personalplanung

Folgende Klassen sind von folgenden Personen umgesetzt wurden.

|  |  |  |
| --- | --- | --- |
| Person | Klasse | Methode |
| Florian Hilbrecht  (M.-Nr.: 631558) | MatrixVector | resize  getRows  getColumns  size  setRows  setColumns  Operator <<  Operator >>  Operator []  Operator ()  Operator ==  Operator !=  Operator =  getRowElements  getColumnElements  isSquare  isZero  isUnit  isDiagonal  isScalar  isUpper  isLower  isTriangular  isColumnVector  isRowVector  swapRow  swapColumn |
| Hannes Kuschick (M-Nr.:) | MatrixArithmetic | isSameType  VectorProduct  Operator +  Operator –  Operator \*  Operator +=  Operator -=  Operator \*= |
| Christoph Dieck (M.-Nr.:) | MatrixMath | dreiecksmatrix  transponiert  determinante  adjunkte  inverse |
| Jürgen Döffinger  (M.-Nr.: 631551) | Matrix | toUpper  isContradiction  isInfinite  solve |
|  | MatrixException | textcolor  getErrorNumber  getErrorMsg  showMsg |

# Klasse MatrixVector

Die Klasse MatrixVector ist die Basisklasse aller weiteren Matrix – Klassen. Sie stellt im wesentlichen Methoden zur Ein- und Ausgabe, sowie zum Abfragen von verschiedenen Eigenschaften von Matrizen zur Verfügung.

## Attribute

Die Klasse MatrixVector enthält die folgenden Attribute (private):

|  |  |
| --- | --- |
| **Attribut** | **Bedeutung** |
| elements | Ist vom Typ vector<vector<T>>  Enthält alle Elemente der Matrix |
| rows | Entspricht der Zeilenanzahl der Matrix |
| columns | Entspricht der Spaltenanzahl der Matrix |

## Methoden

Die Klasse MatrixVector enthält die folgenden Methoden:

Es sind vier Konstruktoren:

* Zur Erstellung einer Matrix ohne Übergabe von Parametern

(es wird automatisch eine 3x3 Matrix erstellt)

* Zur Erstellung einer Matrix mit Übergabe eines Parameters

(es wird eine quadratische Matrix entsprechend des übergebenen Parameters erstellt)

* Zur Erstellung einer Matrix mit Übergabe von zwei Parametern

(es wird eine Matrix entsprechend der übergebenen Parameter (Zeilenanzahl, Spaltenanzahl) erstellt)

(jeweils Initialisierung aller Elemente mit Null!)

* Copy Konstruktor

sowie ein „virtueller“ Destruktor definiert worden.

|  |  |
| --- | --- |
| **Methode** | **Bedeutung** |
| init() | Initialisierungsroutine für Copy-Konstruktor |
| resize() | Initialisierung der Matrix |
| getRows() | Zeilenanzahl zurückgeben |
| getColumns() | Spaltenanzahl zurückgeben |
| size() | Größe der Matrix in Form rows x columns zurückgeben |
| setRows() | Zeilenanzahl neu definieren (Achtung!!! Elemente der Matrix werden auf 0 initialisiert) |
| setColumns() | Spaltenanzahl neu definieren (Achtung!!! Elemente der Matrix werden auf 0 initialisiert) |
| operator <<() | shift-operator << überladen für die Ausgabe der Matrix |
| operator >>() | shift-Operator >> überladen zur Eingabe einer Matrix |
| operator []() | Indexoperator überladen zum Zugriff auf einzelne Zeilen der Matrix  Gekapselt, da es sonst möglich wäre einzelne Elemente, Zeilen oder Spalten mithilfe der Memberfunktion, der Klasse vector zu zerstören |
| operator ()() | Indexoperator überladen zum Zugriff auf ein einzelnes Element der Matrix, dabei ist der row = 1 auch Zeile 1 der Matrix, sowie column = 1 auch Spalte 1 der Matrix |
| operator ==() | Überladen des Vergleichsoperators == |
| operator !=() | Überladen des (Vergleichsoperators) != |
| getRowElements() | Gibt alle Elemente einer bestimmten Zeile als Zeilenvektor zurück |
| getColumnElements() | Gibt alle Elemente einer bestimmten Spalte als Spaltenvektor zurück |
| isSquare() | Prüft ob es sich um eine quadratische Matrix handelt |
| isZero() | Prüft ob es sich um eine Nullmatrixhandelt und gibt entsprechend true oder false zurück |
| isUnit() | Prüft ob es sich bei der Matrix um eine Einheitsmatrix handelt. |
| isDiagonal() | Prüft ob es sich um eine Diagonalmatrix handelt |
| isScalar() | Prüft ob es sich um eine Skalarmatrix handelt |
| isUpper() | Prüft ob es sich um eine rechte/obere Dreiecksmatrix handelt. |
| isLower() | Prüft ob es sich um einer linke/untere Dreiecksmatrix |
| isTriangular() | Prüft auf Dreiecksmatrix und gibt entsprechenden Zahlenwert zurück  bei oberer und unterer Dreiecksmatrix (Diagonalmatrix) wird eine 3 zurückgegeben  bei oberer Dreiecksmatrix wird 1 zurückgegeben  bei unterer Dreiecksmatrix wird 2 zurückgegeben  handelt es sich nicht um eine Dreiecksmatrix wird eine 0 zurückgegeben |
| isColumnVector() | Prüft ob es sich um einen Spaltenvektor handelt |
| isRowVector() | Prüft ob es sich um einen Zeilenvektor handelt |
| swapRow() | Tauscht Zeilen aus |
| swapColumn() | Tauscht Spalten aus |
| operator =() | Zuweisungsoperator |

## Ausnahmebehandlungen

Die Klasse MatrixVector greift bei den von dieser Klasse ausgelösten Ausnahmebehandlungen auf die Klasse MatrixException zurück. Dabei sind die Fehlernummern 100 bis 199 für die Klasse MatrixVector vorgesehen. Derzeit sind folgende Ausnahmen deklariert:

|  |  |  |
| --- | --- | --- |
| Fehlernummer | Fehlermeldung | auslösende Methode |
| 100 | Die Angabe der Zeilenanzahl und/oder Spaltenanzahl ist kleiner als 1. |  |
| 101 | Sie haben eine Zeilenanzahl kleiner 1 eingegeben. |
| 102 | Sie haben eine Spaltenanzahl kleiner 1 eingegeben. |
| 103 | Zeilen- und/oder Spaltenangabe lag ausserhalb des Definitionsbereiches der Matrix. |
| 104 | Die Zeilen- und Spaltenanzahl der Matrizen stimmen nicht ueberein. |
| 105 | Die Zeilenangabe lag ausserhalb des Definitionsbereiches der Matrix. |
| 106 | Die Spaltenangabe lag ausserhalb des Definitionsbereiches der Matrix. |

## Test

### Testfälle

Die folgenden Testfälle wurden mit der Programmversion „Version 1.8.6“ für die Datentypen *float*, *double*, *complex* und *Bruch* durchgeführt.

|  |  |  |  |
| --- | --- | --- | --- |
| **Testfälle** | | | |
| **Nr.** | **Methode** | **Testszenario** | **Beschreibung** |
| 1 | MatrixVector() |  | Erzeugen einer Matrix durch den Standardkonstruktor.  Ohne Übergabe von Parametern, wird automatisch eine 3x3 Matrix erzeugt und mit Null initialisiert.  mit resize(); init() |
| 2 | MatrixVector  (dim) | für dim=2  Für dim=x=10,…  Alle Elemente mit 0 initialisiert | Erzeugen einer Matrix mit Übergabe eines Parameters .  Ausgabe und Initialisierung mit dem Wert Null.  mit resize(); init() |
| 3 | MatrixVector  (rows,columns) | für (3,2)  ebenfalls für (3,4);(6,4);(10;10);… | Erzeugen einer Matrix mit Übergabe von zwei Parametern.  Ausgabe und Initialisierung mit dem Wert Null.  mit resize(); init() |
| 4 | getRows() |  | Rückgabe der Anzahl der Zeilen: 3 |
| 5 |  | Rückgabe der Anzahl der Zeilen: 3 |
| 6 | getColumns() |  | Rückgabe der Anzahl der Spalten: 3 |
| 7 |  | Rückgabe der Anzahl der Spalten: 2 |
| 8 | size() | Matrix 2x2  Matrix 10x10 | Rückgabe der Anzahl der Elemente: 9  Rückgabe der Anzahl der Elemente: 4  Rückgabe der Anzahl der Elemente: 100 |
| 9 | setRows() | -> | Aus einer Matrix (6x4) wurde eine Matrix (5x4) erzeugt.  Initialisierung der Elemente mit Null. |
| 10 | setColumns() | -> | Aus einer Matrix (5x4) wurde eine Matrix (5x6) erzeugt.  Initialisierung der Elemente mit Null. |
| 11 | operator >>()  und  operator <<() | Eingabe für Matrixwerte:  1 2 3 4 5 6 7 8 9 | Einfache Eingabe von Werten für eine Matrix |
| Ausgabe: | |
| 12 |  | Eingabe für Matrixwerte  1 22 333 4444 55555 666666 7777777 88888888 999999999 | Einfache Eingabe von Werten für eine Matrix |
| Ausgabe: | |
| 13 |  | Eingabe für Matrixwerte  1 2 3 4 d r t e 4 | Einfache Eingabe von Werten für eine Matrix  (Zahlen und Buchstaben) |
| Ausgabe: | |
| 14 | operator []() |  | Zugriff auf einzelne Zeilen der Matrix. |
| 15 | operator ()()  operator =() | -> | Zuweisung eines Wertes zu einem Bestimmten Element.  hier: (2,2)=99 |
| 16 | getColumnElements() | -> | Gibt eine Spalte als Spaltenvektor zurück |
| 17 | getRowElements() | -> | Gibt eine Zeile als Zeilenvektor zurück |
| 18 | operator ==() | == | Prüft zwei Matrizen auf Gleichheit  Hier true. |
| 19 | operator !=() | != | Prüft zwei Matrizen auf Ungleichheit  Hier false. |
| 20 | isSquare() |  | Prüft, ob Matrix quadratisch ist.  Hier true. |
| 21 |  | Prüft, ob Matrix quadratisch ist.  Hier false. |
| 22 | isZero() |  | Prüft, ob Matrix eine Nullmatrix ist.  Hier true. |
| 23 |  | Prüft, ob Matrix eine Nullmatrix ist.  Hier false. |
| 24 | isUnit() |  | Prüft, ob Matrix eine Einheitsmatrix ist.  Hier true. |
| 25 |  | Prüft, ob Matrix eine Einheitsmatrix ist.  Hier false. |
| 26 | isDiagonal() |  | Prüft, ob Matrix eine Diagonalmatrix ist.  Hier true. |
| 27 |  | Prüft, ob Matrix eine Diagonalmatrix ist.  Hier false. |
| 28 | isScalar() |  | Prüft, ob (Matrix) ein Skalar ist.  Hier true. |
| 29 |  | Prüft, ob (Matrix) ein Skalar ist.  Hier false. |
| 30 | isUpper() |  | Prüft, ob eine rechte/obere Dreiecksmatrix vorliegt.  Hier true. |
| 31 |  | Prüft, ob eine rechte/obere Dreiecksmatrix vorliegt.  Hier false. |
| 32 | isLower() |  | Prüft, ob eine linke/ untere Dreiecksmatrix vorliegt.  Hier true. |
| 33 |  | Prüft, ob eine linke/ untere Dreiecksmatrix vorliegt.  Hier false. |
| 34 | isTriangular() |  | Gibt 1 zurück.  Diese Matrix ist eine obere Dreiecksmatrix |
| 35 |  | Gibt 2 zurück.  Diese Matrix ist eine untere Dreiecksmatrix |
| 36 |  | Gibt 0 zurück.  Diese Matrix ist eine keine Dreiecksmatrix |
| 37 |  | Gibt 3 zurück.  Diese Matrix ist eine obere und untere Dreiecksmatrix - Diagonalmatrix |
| 38 | isColumnVector() |  | Prüft, ob Matrix ein Spaltenvektor ist.  Hier true. |
| 39 |  | Prüft, ob Matrix ein Spaltenvektor ist.  Hier false. |
| 40 | isRowVector() |  | Prüft, ob Matrix ein Zeilenvektor ist.  Hier true. |
| 41 |  | Prüft, ob Matrix ein Zeilenvektor ist.  Hier false. |
| 42 | swapRow() | -> | Tauscht zwei Zeilen der Matrix aus. |
| 43 | swapColumn() | -> | Tauscht zwei Spalten der Matrix aus. |
|  | Tests für Fehler | | |
| 44 | Matrix mit der Dimension (0,1) anlegen  Matrix<float> q(0,1); | | Fehler 100 wird ausgelöst. |
| 45 | Zeilenanzahl kleiner Eins.  p.setRows(0) | | Fehler 101 wird ausgelöst. |
| 46 | Spaltenanzahl kleiner Eins.  p.setColumns(0) | | Fehler 102 wird ausgelöst. |
| 47 | Spalte ausserhalb der definierten Dimension  p.swapColumn(1,4) | | Fehler 103 wird ausgelöst. |
| 48 | Zuweisung Matrizen unterschiedlicher Dimensionen  r=n | | Fehler 104 wird ausgelöst. |
| 49 | Zeilenangabe ausserhalb des Definitionsbereiches der Matrix.  p.getRowElements(100) | | Fehler 105 wird ausgelöst. |
| 50 | Spaltenangabe ausserhalb des Definitionsbereiches der Matrix.  p.getColumnElements(100) | | Fehler 106 wird ausgelöst. |

(Testfälle wurden für mehrere Testszenarien ausgeführt – siehe Testprogramm.)

### Testprotokoll

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Testfall** | **Bestanden** | **Bemerkung** | **Datum** | **Version** |
| 1 | Ja |  | 20.06.2010 | 1.8.6 |
| 2 | Ja |  | 20.06.2010 | 1.8.6 |
| 3 | Ja |  | 20.06.2010 | 1.8.6 |
| 4 | Ja |  | 20.06.2010 | 1.8.6 |
| 5 | Ja |  | 20.06.2010 | 1.8.6 |
| 6 | Ja |  | 20.06.2010 | 1.8.6 |
| 7 | Ja |  | 20.06.2010 | 1.8.6 |
| 8 | Ja |  | 20.06.2010 | 1.8.6 |
| 9 | Ja |  | 20.06.2010 | 1.8.6 |
| 10 | Ja |  | 20.06.2010 | 1.8.6 |
| 11 | Ja |  | 20.06.2010 | 1.8.6 |
| 12 | Ja | Ausgabe:  Ab der siebten Stelle wird aufgerundet | 20.06.2010 | 1.8.6 |
| 13 | Ja | Ausgabe:  Ab der Eingabe eines Buchstabens wird die Eingabe abgebrochen und der Programmablauf setzt sich fort. | 20.06.2010 | 1.8.6 |
| 14 | Ja |  | 20.06.2010 | 1.8.6 |
| 15 | Ja |  | 20.06.2010 | 1.8.6 |
| 16 | Ja |  | 20.06.2010 | 1.8.6 |
| 17 | Ja |  | 20.06.2010 | 1.8.6 |
| 18 | Ja |  | 20.06.2010 | 1.8.6 |
| 19 | Ja |  | 20.06.2010 | 1.8.6 |
| 20 | Ja |  | 20.06.2010 | 1.8.6 |
| 21 | Ja |  | 20.06.2010 | 1.8.6 |
| 22 | Ja |  | 20.06.2010 | 1.8.6 |
| 23 | Ja |  | 20.06.2010 | 1.8.6 |
| 24 | Ja |  | 20.06.2010 | 1.8.6 |
| 25 | Ja |  | 20.06.2010 | 1.8.6 |
| 26 | Ja |  | 20.06.2010 | 1.8.6 |
| 27 | Ja |  | 20.06.2010 | 1.8.6 |
| 28 | Ja |  | 20.06.2010 | 1.8.6 |
| 29 | Ja |  | 20.06.2010 | 1.8.6 |
| 30 | Ja |  | 20.06.2010 | 1.8.6 |
| 31 | Ja |  | 20.06.2010 | 1.8.6 |
| 32 | Ja |  | 20.06.2010 | 1.8.6 |
| 33 | Ja |  | 20.06.2010 | 1.8.6 |
| 34 | Ja |  | 20.06.2010 | 1.8.6 |
| 35 | Ja |  | 20.06.2010 | 1.8.6 |
| 36 | Ja |  | 20.06.2010 | 1.8.6 |
| 37 | Ja |  | 20.06.2010 | 1.8.6 |
| 38 | Ja |  | 20.06.2010 | 1.8.6 |
| 39 | Ja |  | 20.06.2010 | 1.8.6 |
| 40 | Ja |  | 20.06.2010 | 1.8.6 |
| 41 | Ja |  | 20.06.2010 | 1.8.6 |
| 42 | Ja |  | 20.06.2010 | 1.8.6 |
| 43 | Ja |  | 20.06.2010 | 1.8.6 |
| 44 | Ja |  | 20.06.2010 | 1.8.6 |
| 45 | Ja |  | 20.06.2010 | 1.8.6 |
| 46 | Ja |  | 20.06.2010 | 1.8.6 |
| 47 | Ja |  | 20.06.2010 | 1.8.6 |
| 48 | Ja |  | 20.06.2010 | 1.8.6 |
| 49 | Ja |  | 20.06.2010 | 1.8.6 |
| 50 | Ja |  | 20.06.2010 | 1.8.6 |

# Klasse MatrixArithmetic

## Attribute

## Methoden

## Ausnahmebehandlungen

## Test

# Klasse MatrixMath

## Attribute

## Methoden

## Ausnahmebehandlung

## Test

# Klasse Matrix

Die Klasse Matrix ist durch Mehrfachvererbung (Multiple-inheritance) in Form einer virtuellen Vererbung von den Klassen MatrixArithmetic und MatrixMath abgeleitet und fasst diese somit zu einer Klasse zusammen. Des Weiteren enthält Sie den Gleichungslöser (solve).

## Attribute

Es wurden keine Attribute verwendet. Es wird über die Methoden der Klasse MatrixVector auf die Elemente einer Matrix zugegriffen.

## Methoden

Die Klasse Matrix stellt mit der Methode „solve“ den Gleichungslöser zur Verfügung. Der Prototype sieht wie folgt aus:

Unsigned short solve (Matrix<Datentype>& aV, Matrix<Datentype> ans)

Dabei ist die Funktion eine Memberfunktion und wird von einer Matrix aufgerufen. Hier ein Beispiel:

Matrix<float> A(3,3); // Koeffizientenmatrix

Matrix<float> x(3,1); // Lösungsvektor

Matrix<float> c(3,1); // Spaltenvektor aus den absoluten Gliedern des Gleichungssystems

unsigned short status; // Liefert später den Status der Lösung

cin >> A; // Eingabe der Koeffizienten

cin >> c; // Eingabe der absoluten Glieder

status = A.solve(c,x); // Gleichungssystem lösen

switch(status)

{

case 0 : cout << „Das Gleichungssystem hat keine Lösung.“ << endl; break;

case 1 : cout << x << endl; break;

case 2 : cout << “Das Gleichungssystem hat unendlich viele Lösungen.“ << endl; break;

}

Dabei sollte darauf geachtet werden, dass der Lösungsvektor und der Spaltenvektor mit den absoluten Gliedern, genau so viele Zeilen wie Koeffizienten besitzen und nur aus einer Spalte bestehen. Alles andere würde zu einer Ausnahmebehandlung führen, da so ein lösen des Gleichungssystems nicht möglich wäre.

Wird der Gleichungslöser aufgerufen erfolgt zunächst eine Prüfung, ob der Lösungsvektor und der Vektor mit den absoluten Gliedern Spaltenvektoren sind. Anschließen wird geprüft, ob die Anzahl der absoluten Glieder mit der Zeilenzahl des Lösungsvektors übereinstimmt. Des Weiteren wird geprüft, ob die Anzahl der Koeffizienten mit der Anzahl der absoluten Glieder übereinstimmt. Sollte dies alles nicht der Fall sein, werden entsprechende Ausnahme-behandlungen ausgelöst.

Der nächste Schritt ist das bilden einer oberen Dreiecksmatrix über die (private) Methode „toUpper“. Diese Methode ist gekapselt und kann nur von der Klasse Matrix aufgerufen werden, da Sie lediglich für den Gleichungslöser da ist. Diese Methode setzt das Verfahren nach Gauß um, wie folgendes Beispiel zeigt.

Als Beispiel wird von folgendem linearen Gleichungssystem ausgegangen:

Daraus ergibt sich die Koeffizientenmatrix

und der Spaltenvektor mit den absoluten Gliedern

.

Durch die Funktion toUpper, welche sich den Gaußschen Algorithmus zu Nutze macht, ergibt sich die Koeffizientenmatrix als obere Dreiecksmatrix.

Dieses Ergebnis wird der Methode „solve“ zurückgegeben. Diese prüft nun auf Wiederspruch. Ein Wiederspruch wäre beispielsweise folgender:

Hier liegt der Wiederspruch in der dritten Zeile, denn um das Ergebnis für den 3. Koeffizienten zu erhalten müsste man das 3. absolute Glied (-183) durch den Wert der Zeile 3 und Spalte 3 der Koeffizientenmatrix teilen. Dies würde aber zu einer Division durch Null führen, welche mathematisch nicht durchführbar ist. Somit liegt also ein Wiederspruch vor. Ist dies der Fall, wird die weitere Bearbeitung gestoppt und, als status eine 0 zurückgegeben.

Der nächste Schritt ist nun, den Lösungsvektor nach Gauß-Jordan zu bilden. Dabei ergibt sich für die Matrizen folgendes Ergebnis:

Dieses Ergebnis wird nun auf unendlich viele Lösungen überprüft. Ein Gleichungssystem mit unendlich vielen Lösungen würde in der Phase wie folgt aussehen:

Liegt solch eine Lösung vor, wird ebenfalls die weitere Ausführung der Methode „solve“ gestoppt und als Status eine 2 zurückgegeben.

Liegt aber eine Lösung vor, wird der Lösungsvektor auf die Lösung gesetzt und der status 1 zurückgegeben.

## Ausnahmebehandlung

Die Klasse Matrix greift bei den von dieser Klasse ausgelösten Ausnahmebehandlungen auf die Klasse MatrixException zurück. Dabei sind die Fehlernummer 400 bis 499 für die Klasse Matrix vorgesehen. Derzeit sind folgende Ausnahmen deklariert:

|  |  |  |
| --- | --- | --- |
| Fehlernummer | Fehlermeldung | auslösende Methode |
| 400 | Die Matrix mit den absoluten Gliedern ist kein Spaltenvektor. | solve |
| 401 | Die Ergebnismatrix ist kein Spaltenvektor. |
| 402 | Die Anzahl absoluter Glieder stimmt nicht mit der Zeilenanzahl des Lösungsvektors überein. |
| 403 | Die Anzahl der absoluten Glieder stimmt nicht mit der Anzahl Gleichungen in der Koeffizientenmatrix ueberein. |

## Test

### Testfälle

|  |  |  |  |
| --- | --- | --- | --- |
| **Funktionstest** | | | |
| Nr. | Datentyp | Gleichungssystem | Beschreibung |
| 1 | float | Ergebnis: | In diesem Test wird geprüft, ob der Gleichungslöser ein lösbares Gleichungssystem richtig löst. |
| 2 | double |
| 3 | complex<double> |
| 4 | Bruch |
| 5 | float | Ergebnis:  Man kann erkennen, dass die Zeile drei ein Vielfaches der Zeile zwei ist und somit ein Gleichungssystem mit unendlich vielen Lösungen vorliegt. | Prüfung des Gleichungslöser auf erkennen eines Gleichungssystems mit unendlich vielen Lösungen. |
| 6 | double |
| 7 | complex<double> |
| 8 | Bruch |
| 9 | float | Ergebnis:  Es ist zu erkennen, dass in der dritten Zeile ein Widerspruch vorliegt, denn. Das Gleichungssystem hat somit keine Lösung. | Dieser Test prüft das Gleichungssystem auf das erkennen eines Gleichungssystem, zu dem es keine Lösung gibt. |
| 10 | double |
| 11 | complex<double> |
| 12 | Bruch |
| **Extremfalltest** | | | |
| 13 | Float | Ergebnis:  Das Gleichungssystem besitzt keine Lösung. (Widerspruch in der dritten Zeile) | Der Test soll die Extremfälle ausfindig machen, welche durch das Problem der nicht-darstellbaren Zahlen auftreten könnte. So kommt es während der Bildung der oberen Dreiecksmatrix zu einem Faktor von , welcher nicht als Gleitkommazahl aufgelöst werden kann, sondern nur gerundet wiedergegeben werden kann. |
| 14 | double |
| 15 | complex<double> |
| 16 | Bruch |
| 17 | float | Ergebnis: | In diesem Test ist zu prüfen, wie der Gleichungslöser bzw. die Klasse Matrix mit kleinen Zahlen im Ergebnis umgehen kann. |
| 18 | double |
| 19 | complex<double> |
| 20 | Bruch |

### Testprotokoll

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testfall | Bestanden | Bemerkung | Datum | Version |
| 1 | Ja |  | 16.06.2010 | 1.8.1 |
| 2 | Ja |  | 16.06.2010 | 1.8.1 |
| 3 | Ja |  | 16.06.2010 | 1.8.1 |
| 4 | Ja |  | 16.06.2010 | 1.8.1 |
| 5 | Ja |  | 16.06.2010 | 1.8.1 |
| 6 | Ja |  | 16.06.2010 | 1.8.1 |
| 7 | Ja |  | 16.06.2010 | 1.8.1 |
| 8 | Ja |  | 16.06.2010 | 1.8.1 |
| 9 | Ja |  | 16.06.2010 | 1.8.1 |
| 10 | Ja |  | 16.06.2010 | 1.8.1 |
| 11 | Ja |  | 16.06.2010 | 1.8.1 |
| 12 | Ja |  | 16.06.2010 | 1.8.1 |
| 13 | Nein | Der Test gab vor, dass zu erkenne ist, dass es sich um ein nichtlösbares Gleichungssystem handelt. Dies wurde nicht erkannt, sondern als Lösung  ausgegeben. | 16.06.2010 | 1.8.1 |
| 14 | Ja |  | 16.06.2010 | 1.8.1 |
| 15 | Ja |  | 16.06.2010 | 1.8.1 |
| 16 | Ja |  | 16.06.2010 | 1.8.1 |
| 17 | Ja | Es werden die Lösungen wie folgt gerundet:  Dies entspricht einer Abweichung um,  vom tatsächlichen Ergebnis.  und werden zwar gerundet, entsprechen aber den vorgegebenen Ergebnissen, wenn eine Genauigkeit kleiner gefordert ist. | 16.06.2010 | 1.8.1 |
| 18 | Ja | 16.06.2010 | 1.8.1 |
| 19 | Ja | 16.06.2010 | 1.8.1 |
| 20 | Nein | Ausgegebene Lösung:  Bei der Berechnung des ersten Lösungswertes kommt es zu einer Bereichsüberschreitung des Datentyps long, welcher in der Klasse Bruch für Nenner und Zähler verwendet werden. Es entsteht dabei im Nenner ein Wert von .  Dieser Fehler könnte durch die Verwendung des Datentyps long long beseitigt werden. | 16.06.2010 | 1.8.1 |
| 20 | Ja | Die Lösung entspricht nun der tatsächlichen Lösung:  Durch verwenden des Datentyps long long für Nenner und Zähler in der Klasse Bruch, konnte somit der Fehler in der Version 1.8.1 beseitigt werden. | 17.06.2010 | 1.8.2 |

# Klasse MatrixException

Die Klasse MatrixException dient den Matrixklassen zum auslösen von Ausnahmebehandlungen.

## Attribute

Die Klasse MatrixException beinhaltet die Attribute err und msg. Das Attribut err ist vom Typ unsigned int und enthält die Fehlernummer. Das Attribut msg ist vom Typ string und beinhaltet die Fehlermeldung.

## Methoden

Es wurden folgende Methoden umgesetzt:

* getErrorNumber
* getErrorMsg
* showMsg

Die Methode getErrorNumber gibt die Fehlernummer zurück, welche in err gespeichert ist. Mithilfe der Methode getErrorMsg kann die Fehlermeldung, welche in msg gespeichert ist ausgelesen werden. Soll die Fehlermeldung direkt angezeigt werden, dann kann dazu die Methode showMsg verwendet werden.

## Ausnahmebehandlung

Für die Klasse MatrixException sind keine Ausnahmebehandlungen vorgesehen.

## Test

# Verzeichnisse

## Quellenverzeichnis

Bjarne Stroustrup

Die C++ - Programmiersprache

ISBN 0-201-70073-5

Addision-Wesley Verlag

Ulrich Breymann

Designing Components with the C++ STL – A New Approach to Programming

ISBN 0-201-17816-8

Addison Wesley Longman Limited

Jürgen Wolf

C++ von A bis Z – Das umfassende Handbuch

2. Auflage, 2009

ISBN 978-3-8362-1429-2

Galileo Press

Christoph Kecher

UML 2 – Das umfassende Handbuch

3. Auflage, 2009

ISBN 978-3-8362-1419-3

Galileo Press

Thomas Grechin, Mario Bernhart, Roland Breiteneder, Karin Kappel

Softwaretechnik – Mit Fallbeispielen aus realen Entwicklungsprojekten

ISBN 978-3-86894-007-7

Pearson Education Deutschland GmbH

Bronstein, Semendjajew, Musiol, Mühlig

Taschenbuch der Mathematik

7. Auflage, 2008

ISBN 978-3-8171-2017-8

Wissenschaftlicher Verlag Harri Deutsch GmbH

Lothar Papula

Mathematische Formelsammlung – Für Ingenieure und Naturwissenschaftler

9. Auflage, 2006

ISBN 978-3-8348-0156-2

Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH

<http://www.cppreference.com/wiki/>

<http://msdn.microsoft.com/de-de/library/60k1461a%28v=VS.90%29.aspx>

Prof. Dr.-Ing. Jack

Fachhochschule Jena

Fachbereich Elektrotechnik/Informationstechnik

Vorlesungsscripte Informatik IIa

1. Das Klassendiagramm wurde mithilfe des Programmes *Microsoft® Office Visio® Professional 2003 (11.8323.8324) SP3* erstellt. [↑](#footnote-ref-1)