FACHHOCHSCHULE JENA

Matrix

Informatik IIa - Projektarbeit

Jürgen Döffinger, Florian Hilbrecht, Hannes Kuschick, Christoph Dieck

22.06.2010

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} = (a_{ij})$$

1 Inhaltsverzeichnis

2 Analyse

2.1 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;
   cou   <<  mf1  * f  * mf2 << endl;   //   (Skalar)Multiplikation
   +
   cou   <<   mf3.det() << endl;   // Determinante
   +
   cou   <<   mf3.hls() << endl;    // Lösung des homogenen LGS
   +
```

```
// 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.

2.2 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

2.2.1 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.

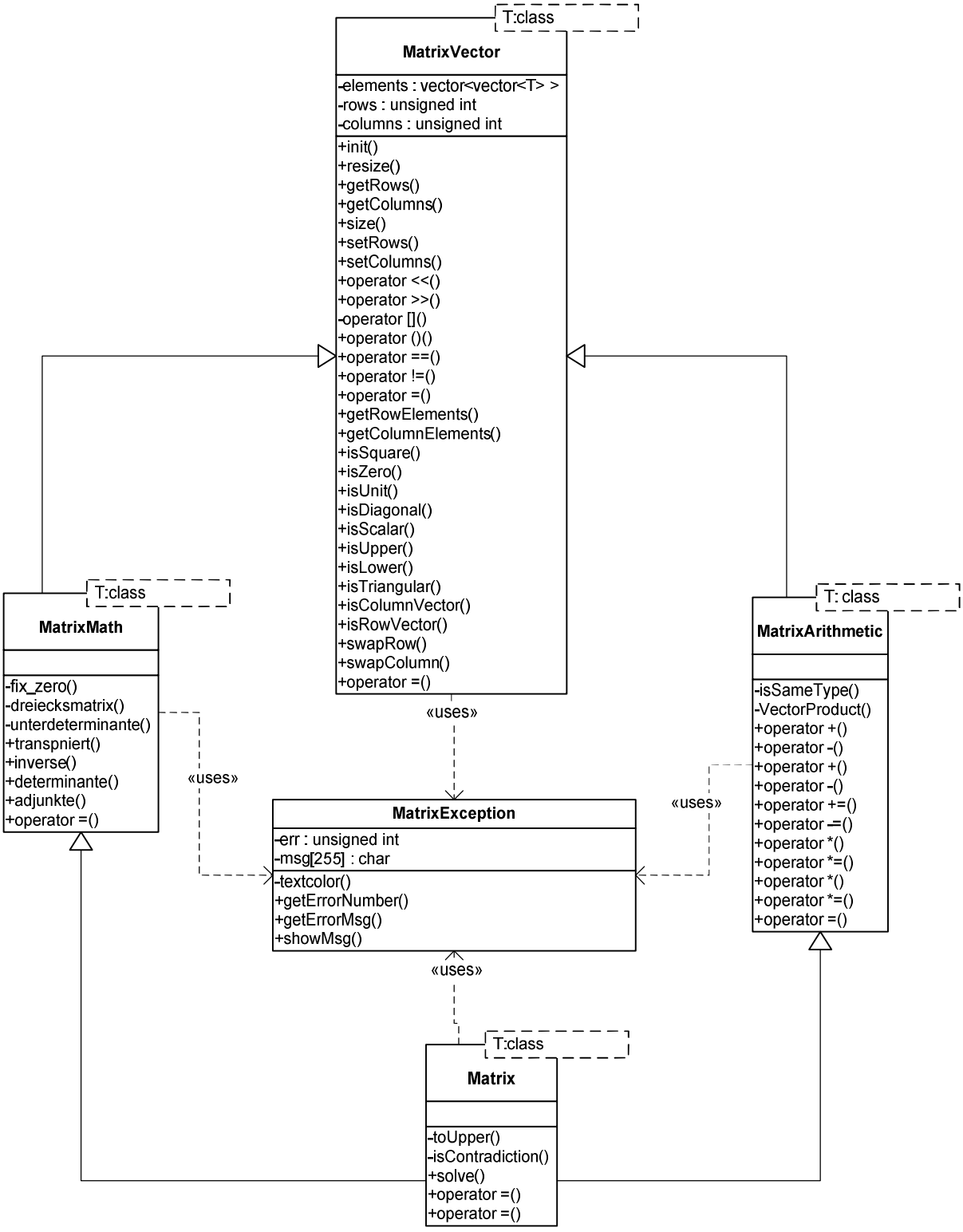

Bild 1 Klassendiagramm Übersicht[1]

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

2.2.2 Personalplanung

Folgende Klassen sind von folgenden Personen umgesetzt wurden.

Person	Klasse	Methode
Florian Hilbrecht (M.-Nr.:)	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 Operator =
Hannes Kuschick (M-Nr.:)	MatrixArithmetic	isSameType VectorProduct Operator + Operator – Operator * Operator += Operator -= Operator *= Operator =
Christoph Dieck (M.-Nr.:)	MatrixMath	fix_zero dreiecksmatrix unterdeterminante transponiert inverse determinante adjunkte Operator =
Jürgen Döffinger (M.-Nr.: 631551)	Matrix	toUpper isContradiction solve
	MatrixException	textcolor getErrorNumber getErrorMsg showMsg

3 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.

3.1 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

3.2 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

3.3 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.`	

3.4 Test

3.4.1 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()	$\begin{matrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{matrix}$	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)	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$ für dim=2 $\begin{pmatrix}i1.1 & \cdots & i1.x\\ \vdots & \ddots & \vdots\\ ix.1 & \cdots & ix.x\end{pmatrix}$ 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)	$\begin{matrix}0 & 0\\0 & 0\\0 & 0\end{matrix}$ 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()	$\begin{matrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{matrix}$	Rückgabe der Anzahl der Zeilen: 3
5		$\begin{matrix}0 & 0\\0 & 0\\0 & 0\end{matrix}$	Rückgabe der Anzahl der Zeilen: 3
6	getColumns()	$\begin{matrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{matrix}$	Rückgabe der Anzahl der Spalten: 3
7		$\begin{matrix}0 & 0\\0 & 0\\0 & 0\end{matrix}$	Rückgabe der Anzahl der Spalten: 2
8	size()	$\begin{matrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{matrix}$ Matrix 2x2 Matrix 10x10	Rückgabe der Anzahl der Elemente: 9 Rückgabe der Anzahl der Elemente: 4 Rückgabe der Anzahl der Elemente: 100

<table>
<tr><td>9</td><td>setRows()</td><td>0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
->
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0</td><td>Aus einer Matrix (6x4) wurde eine Matrix (5x4) erzeugt.
Initialisierung der Elemente mit Null.</td></tr>
<tr><td>10</td><td>setColumns()</td><td>0 0 0 0
0 0 0 0
0 0 0 0->
0 0 0 0
0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0</td><td>Aus einer Matrix (5x4) wurde eine Matrix (5x6) erzeugt.
Initialisierung der Elemente mit Null.</td></tr>
<tr><td rowspan="2">11</td><td rowspan="2">operator >>()
und
operator <<()</td><td>Eingabe für Matrixwerte:
1 2 3 4 5 6 7 8 9</td><td>Einfache Eingabe von Werten für eine Matrix</td></tr>
<tr><td colspan="2">Ausgabe:
1 2 3
4 5 6
7 8 9</td></tr>
<tr><td rowspan="2">12</td><td rowspan="2"></td><td>Eingabe für Matrixwerte
1 22 333 4444 55555
666666 7777777
88888888 999999999</td><td>Einfache Eingabe von Werten für eine Matrix</td></tr>
<tr><td colspan="2">Ausgabe:
1 22 333
4444 55555 666666
$7.77778e+006$ $8.88889e+007$ $1e+009$</td></tr>
<tr><td rowspan="2">13</td><td rowspan="2"></td><td>Eingabe für Matrixwerte
1 2 3 4 d r t e 4</td><td>Einfache Eingabe von Werten für eine Matrix
(Zahlen und Buchstaben)</td></tr>
<tr><td colspan="2">Ausgabe:
1 2 3
4 0 0
0 0 0</td></tr>
<tr><td rowspan="2">14</td><td rowspan="2"></td><td>Prinzip:
m1 >> m2
m1 << m2
Eingabe für Matrixwerte
1 2 3 4 5 6 7 8</td><td>Mehrfache Ein- und Ausgabe von Matrizen.</td></tr>
<tr><td colspan="2">Ausgabe:</td></tr>
</table>

		1 2 3 4 5 6 7 8	
15	operator []()	0 0 0 0 0 0 0 0 0	Zugriff auf einzelne Zeilen der Matrix.
16	operator ()() operator =()	0 0 0 0 0 0->0 0 0 0 0 0 0 0 99 0 0 0 0	Zuweisung eines Wertes zu einem Bestimmten Element. hier: (2,2)=99
17	getColumnElements()	1 2 3 4 5 6->1 7 8 9 4 7	Gibt eine Spalte als Spaltenvektor zurück
18	getRowElements()	1 2 3 4 5 6->1 2 3 7 8 9	Gibt eine Zeile als Zeilenvektor zurück
19	operator ==()	0 0 0 0 == 0 0 0 0	Prüft zwei Matrizen auf Gleichheit Hier true.
		0 0 0 0 == 1 2 3 4	Prüft zwei Matrizen auf Gleichheit Hier false.
20	operator !=()	0 0 0 0 != 0 0 0 0	Prüft zwei Matrizen auf Ungleichheit Hier false.
		0 0 0 0 != 1 2 3 4	Prüft zwei Matrizen auf Ungleichheit Hier true.
21	isSquare()	0 0 0 0	Prüft, ob Matrix quadratisch ist. Hier true.
22		0 0 0 0 0 0	Prüft, ob Matrix quadratisch ist. Hier false.
23	isZero()	0 0 0 0	Prüft, ob Matrix eine Nullmatrix ist. Hier true.
24		1 2 3 4	Prüft, ob Matrix eine Nullmatrix ist. Hier false.
25	isUnit()	1 0 0 1	Prüft, ob Matrix eine Einheitsmatrix ist. Hier true.
26		1 5 9 3	Prüft, ob Matrix eine Einheitsmatrix ist. Hier false.
27	isDiagonal()	2 0 0 1	Prüft, ob Matrix eine Diagonalmatrix ist. Hier true.

28		1 1 1 1	Prüft, ob Matrix eine Diagonalmatrix ist. Hier false.
29	isScalar()	1	Prüft, ob (Matrix) ein Skalar ist. Hier true.
30		1 1 1 1	Prüft, ob (Matrix) ein Skalar ist. Hier false.
31	isUpper()	1 1 0 1	Prüft, ob eine rechte/obere Dreiecksmatrix vorliegt. Hier true.
32		1 1 1 1	Prüft, ob eine rechte/obere Dreiecksmatrix vorliegt. Hier false.
33	isLower()	1 0 1 1	Prüft, ob eine linke/ untere Dreiecksmatrix vorliegt. Hier true.
34		1 1 1 1	Prüft, ob eine linke/ untere Dreiecksmatrix vorliegt. Hier false.
35	isTriangular()	1 1 0 1	Gibt 1 zurück. Diese Matrix ist eine obere Dreiecksmatrix
36		1 0 1 1	Gibt 2 zurück. Diese Matrix ist eine untere Dreiecksmatrix
37		1 2 3 4 5 6 7 8 9	Gibt 0 zurück. Diese Matrix ist eine keine Dreiecksmatrix
38		1 0 0 0 2 0 0 0 1	Gibt 3 zurück. Diese Matrix ist eine obere und untere Dreiecksmatrix - Diagonalmatrix
39	isColumnVector()	0 0 0	Prüft, ob Matrix ein Spaltenvektor ist. Hier true.
40		0 0 0	Prüft, ob Matrix ein Spaltenvektor ist. Hier false.
41	isRowVector()	0 0 0	Prüft, ob Matrix ein Zeilenvektor ist. Hier true.
42		0 0 0	Prüft, ob Matrix ein Zeilenvektor ist. Hier false.
43	swapRow()	1 2 3 4 5 6-> 7 8 9 1 2 3 7 8 9 4 5 6	Tauscht zwei Zeilen der Matrix aus.
44	swapColumn()	1 2 3 4 5 6-> 7 8 9 2 1 3 5 4 6 8 7 9	Tauscht zwei Spalten der Matrix aus.
	Tests für Fehler		
45	Matrix mit der Dimension (0,1) anlegen Matrix<float> q(0,1);		Fehler 100 wird ausgelöst.

46	Zeilenanzahl kleiner Eins. p.setRows(0)	Fehler 101 wird ausgelöst.
47	Spaltenanzahl kleiner Eins. p.setColumns(0)	Fehler 102 wird ausgelöst.
48	Spalte ausserhalb der definierten Dimension p.swapColumn(1,4)	Fehler 103 wird ausgelöst.
49	Zuweisung Matrizen unterschiedlicher Dimensionen r=n	Fehler 104 wird ausgelöst.
50	Zeilenangabe ausserhalb des Definitionsbereiches der Matrix. p.getRowElements(100)	Fehler 105 wird ausgelöst.
51	Spaltenangabe ausserhalb des Definitionsbereiches der Matrix. p.getColumnElements(100)	Fehler 106 wird ausgelöst.

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

3.4.2 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: 1 22 333 4444 55555 666666 $7.77778e+006$ $8.88889e+007$ $1e+009$ Ab der siebten Stelle wird aufgerundet	20.06.2010	1.8.6
13	Ja	Ausgabe: 1 2 3 4 0 0 0 0 0 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
51	Ja		20.06.2010	1.8.6

4 Klasse MatrixArithmetic

4.1 Attribute

Es sind keine Attribute in dieser Klasse definiert.

4.2 Methoden

Die Memberfunktion *isSameType()* überprüft die Gleichheit der Dimensionen zweier Matrizen und gibt bei gleicher Zeilen- und Spaltenanzahl eine 1 und bei ungleicher Dimension eine 0 als Wahrheitswert zurück.

Die Memberfunktion *VectorProduct()* beinhaltet die Berechnung des Vektorprodukts au s zwei Spaltenvektoren. Dazu wird zunächst geprüft, ob es sich bei beiden Matrizen um Spaltenvektoren handelt. Ist die darauf folgende Prüfung auf gleiche Anzahl der Zeilen erfolgreich, wird eine Ergebnismatrix erstellt, in die dann, mit Hilfe einer While-Schleife zur Durchführung des Vektorprodukts, der Ergebnisvektor eingetragen wird. Diese Ergebnismatrix wird dann an die Funktion zurückgegeben.

Operatoren-Overloading	
Operator	***Beschreibung***
+	Überprüfung auf Übereinstimmung der Matrizendimensionen (gleiche Zeilen- und Spaltenanzahl), Komponentenweise Addition
-	Überprüfung auf Übereinstimmung der Matrizendimensionen (gleiche Zeilen- und Spaltenanzahl), Komponentenweise Subtraktion
*	a) Multiplikation zweier zweidimensionaler Matrizen, Bedingung Anzahl Spalten Matrix 1 = Anzahl Zeilen Matrix 2, Falk-Schema
	b) Multiplikation zweier Spaltenvektoren (Kreuzprodukt)
	c) Multiplikation einer Matrix mit Skalar (komponentenweise)
+=	Addition des Elements vor Operator mit Elemtent nach Operator Speicherung des Ergebnisses in Variable vor Operator
-=	Subtraktion des Elements vor Operator mit Elemtent nach Operator Speicherung des Ergebnisses in Variable vor Operator
*=	a) Multiplikation Matrix mit Skalar
	b) Multiplikation zweier Matrizen

4.3 Ausnahmebehandlungen

Die Klasse MatrixArithmetic greift bei den von dieser Klasse ausgelösten Ausnahmebehandlungen auf die Klasse MatrixException zurück. Dabei sind die Fehlernummer 200 bis 299 für die Klasse MatrixArithmetic vorgesehen. Derzeit sind folgende Ausnahmen deklariert:

Fehlernummer	Fehlermeldung	ausgelöste Methode
200	Die Spaltenanzahl der Matrix 1 stimmt nicht mit der Zeilenanzahl der Matrix 2 ueberein.	
201	Bei den Spaltenvektoren liegt keine gleiche Anzahl an Zeilen vor, daher kann das Vektorprodukt nicht gebildet werden.	

202	Die beiden zu addierenden Matrizen stimmen nicht in der Anzahl der Zeilen und Spalten ueberein.	
203	Die beiden zu subtrahierenden Matrizen stimmen nicht in der Anzahl der Zeilen und Spalten ueberein.	
204	Es wurde versucht ein Vektorprodukt zu berechnen, obwohl keine Spaltenvektoren vorlagen.	

4.4 Test

Funktionstest			
Test der Fehlermeldungen			
NR.	Datentyp	Operation	Beschreibung (Erwartung)
01	float, complex, bruch	a+d	Dimensionen (Zeilen u. Spalten) der beiden Matrizen ist unterschiedlich > Ausgabe Fehlermeldung erwartet
02	float, complex, bruch	a-d	Dimensionen (Zeilen u. Spalten) der beiden Matrizen ist unterschiedlich > Ausgabe Fehlermeldung erwartet
03	float, complex, bruch	c*d	Dimensionen (Zeilen u. Spalten) der beiden Matrizen unpassend > Ausgabe Fehlermeldung erwartet
04	float, complex, bruch	h*i	ungleiche Anzahl von Zeilen in beiden Vektoren > Ausgabe Fehlermeldung erwartet

Test Addition / Subtraktion			
NR.	Datentyp	Operation	Beschreibung (Erwartung)
05	float, complex, bruch	a+b	Ausgabe Ergebnismatrix, selbe Dimension wie a und b, richtige Komponentenweise Addition
06	float, complex, bruch	a+b+c	Ausgabe Ergebnismatrix, selbe Dimension wie a, b und c richtige Komponentenweise Addition
07	float, complex, bruch	c=a+b	Ausgabe Ergebnismatrix, selbe Dimension wie a und b, richtige Komponentenweise Addition, Ergebnis in c
08	float, complex, bruch	a+=b	Ausgabe Ergebnismatrix, selbe Dimension wie a und b, richtige komponentenweise Additon, Ergebnis in a
09	float, complex, bruch	a-b	Ausgabe Ergebnismatrix, selbe Dimension wie a und b, richtige Komponentenweise Subtraktion
10	float, complex, bruch	a-b-c	Ausgabe Ergebnismatrix, selbe Dimension wie a, b und c richtige komponentenweise Subtraktion
11	float, complex, bruch	a-=b	Ausgabe Ergebnismatrix, selbe Dimension wie a und b, richtige komponentenweise Subtraktion, Ergebnis in a
12	float, complex, bruch	a+b-c	Ausgabe Ergebnismatrix, selbe Dimension wie a, b und c richtige Komponentenweise Addition / Subtraktion
13	float, complex, bruch	a-b+c	Ausgabe Ergebnismatrix, selbe Dimension wie a, b und c richtige Komponentenweise Addition / Subtraktion
14	float, complex, bruch	a+b-a	Ausgabe Ergebnismatrix, selbe Dimension wie a, b und c richtige Komponentenweise Addition / Subtraktion
15	float, complex, bruch	c=a=b	Ausgabe Ergebnismatrix, selbe Dimension wie a, b und c in a und c gleiche Elemente wie in b
16	float, complex,	a=a	Ausgabe a

	bruch		
17	float, complex, bruch	a==b	Ausgabe 0 wenn ungleich, Ausgabe 1 wenn gleich
18	float, complex, bruch	a!=b	Ausgabe 1 wenn ungleich, Ausgabe 0 wenn gleich

Test Multiplikation			
NR.	Datentyp	Operation	Beschreibung (Erwartung)
19	float, complex, bruch	a*b	Ausgabe Ergebnismatrix mit Spaltenanz. von a und Zeilenanz. von b
20	float, complex, bruch	b*a	Ausgabe Ergebnismatrix mit Spaltenanz. von b und Zeilenanz. von a
21	float, complex, bruch	k*a	Ausgabe Ergebnismatrix in Dimension von a jedes Element mit sklararer Konstante k multipliziert
22	float, complex, bruch	a*k	Ausgabe Ergebnismatrix in Dimension von a jedes Element mit sklararer Konstante k multipliziert
23	float, complex, bruch	f*g	Ausgabe Ergebnismatrix (Vektor)
24	float, complex, bruch	g*f	Ausgabe Ergebnismatrix (Vektor)
25	float, complex, bruch	f*g*h	Ausgabe Ergebnismatrix (Vektor)
26	float, complex, bruch	(f*g)*h	Ausgabe Ergebnismatrix (Vektor)
27	float, complex, bruch	a*=b	Ausgabe Ergebnismatrix mit Spaltenanz. von a und Zeilenanz. von b Ergebnismatrix in a vorhanden
28	float, complex, bruch	a*b*c	

Test Kombination Multiplikation und Addition / Subtraktion

NR.	Datentyp	Operation	Beschreibung (Erwartung)
29	float, complex, bruch	a+b*c	Ausgabe Ergebnismatrix, erst Multiplikation b und c dann Addition des Ergebnisses mit a
30	float, complex, bruch	(a+b)*c	Ausgabe Ergebnismatrix. Erst Addition a und b dann Multiplikation des Ergebnisses mit c
31	float, complex, bruch	a*(b=a+c)+c	Ausgabe Ergebnismatrix, a und c addiert, Ergebnis mit a multipliziert, Ergebnis mit c addiert, ergebnis von addition a und c in b

Testprotokoll

NR	Bestanden	Bemerkung	Datum	Version
01	ja		20.06.2010	1.8.6
02	ja		20.06.2010	1.8.6
03	ja		20.06.2010	1.8.6
04	ja		20.06.2010	1.8.6
05	ja		20.06.2010	1.8.6
06	ja		20.06.2010	1.8.6
07	ja		20.06.2010	1.8.6
08	ja		20.06.2010	1.8.6
09	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		20.06.2010	1.8.6
13	ja		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

5 Klasse MatrixMath

Die Klasse „MatrixMath“ ist durch eine virtuelle Vererbung von der Klasse „MatrixArithmetic“ abgeleitet und fasst diese zu einer Klasse zusammen.
Die Klasse beinhaltet folgende Funktionen:

- Bestimmung der Determinante einer Matrix
- Bildung der Adjunkte einer Matrix
- Transponieren einer Matrix
- Bildung der Inversen einer Matrix

5.1 Attribute

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

5.2 Methoden

5.2.1 Hilfsfunktionen

Alle erstellten Hilfsfunktionen können nur innerhalb der Klasse MatrixMath verwendet werden, sie wurden mit dem Schlüsselwort „private“ implementiert.

5.2.1.1 dreiecksmatrix

Sie ist eine Hilfsfunktion für die Determinanten Funktion. Ihre Aufgabe besteht darin die übergebene Matrix A(j,i) in eine Dreiecksmatrix umzuwandeln.
Die Bildung der Drecksmatrix beruht auf dem Gaußschen Eliminationsverfahren. Die Funktion bewegt sich auf der mittleren Diagonalen der Matrix und dividiert unterliegende Spaltenelemente. Bei der Division wird der Wert des Quotienten anschließend mit den aktuellen Elementen auf der Zeile multipliziert und von den unterliegenden Elementen abgezogen.
Bei einem solchen Durchlauf wird durch die Hilfsfunktion „fix_zero“ die Division durch Null umgangen. Nach erfolgreichem Durchlauf ergibt sich eine ober Dreiecksmatrix.

Anmerkung

Durch die Ungenauigkeit der Fließkomma Datentypen entstehen Rundungsfehler. Diese können z.B. den Wert der Determinante verfälschen. Um dies zu beheben, müssten für die betroffen Datentypen Rundungsfunktionen erstellt werden, dies wurde in der vorliegenden Version nicht umgesetzt. Alternativ kann aber der Datentyp „Bruch“ genutzt werden, hier entstehen keine Abweichungen.

5.2.1.2 fix_zero

Sie wird von der Hilfsfunktion „Dreiecksmatrix“ benötigt, um eine mögliche Division durch Null abzufangen.
Wenn der aktuelle Divisor der erhaltenen Matrix A(j,i) Null ist, wird die Spalte solange getauscht, bis der Divisor ≠ Null ist. Nach jedem Tausch wird das Vorzeichen gewechselt. Wenn nach diesem Prozess immer noch eine Null als Divisor vorliegt, wird die Matrix zurückgegeben, da eine komplette Nullzeile vorliegt, die Determinante ist in diesem Fall = Null.

5.2.1.3 *unterdeterminante*

Sie wird von der Methode „Adjunkte“ verwendet und erhält bei Aufruf die Koordinate zum Streichen der Spalte/Zeile der Matrix A(j,i). Es wird nach dem Aufruf eine neue Matrix mit den Eigenschaften A(j-1,i-1) erstellt. Nach Streichen der *j*-ten Zeile und der *i*-ten Spalte der Matrix A(j,i) werden die übrigen Elemente der Matrix A(j-1,i-1) zugeordnet .

Zum Schluss wird A(j-1,i-1) zurückgegeben.

5.2.2 Hauptfunktionen

5.2.2.1 *transponiert*

Nach erhalten der Matrix A(j,i) wird eine Matrix A(i,j) erstellt.
Die Transponierte der Matrix A=(aij) vom Format m x n ist die Matrix A^T=(aij) vom Format n x m,
das heißt zu

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

ist die Transponierte

$$A^T = \begin{pmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{pmatrix}$$

Die Methode schreibt also die erste Zeile als erste Spalte und die zweite Zeile als zweite Spalte usw.

5.2.2.2 *inverse*

Die Inverse Matrix wird nach folgenden Mathematischen Model erstellt:

$$A^{-1} = \frac{1}{\det(A)} adj(A)$$

Die Funktion „inverse“ benötigt folgenden Funktionen:

1. determinate
2. adjunkte

Wenn die Determinante der übergebenen Matrix A(j,i) ≠ Null ist wird eine temporäre Matrix mit gleichen Eigenschaften erstellt. Anschließend wird die Adjunkte der Matrix mithilfe der Methode „Adjunkte“ gebildet. Folgend werden alle Elementen von adj(A) durch die Hauptdeterminante der Matrix A(j,i) dividiert.
Zum Schluss wird die Inverse Matrix zurückgegeben.

5.2.2.3 determinante

Die Methode „determinant“ benötigt folgende Funktionen:

1. dreiecksmatrix

Die Funktion „Determinante“ übergibt die erhaltene Matrix an die Funktion „Dreiecksmatrix“, sie bildet eine obere Dreiecksmatrix. Wenn eine solche vorliegt, werden all die Elemente auf die Diagonale multipliziert.
Die Funktion gibt anschließend die ermittelte Determinante zurück.

5.2.2.4

5.2.2.5 adjunkte

Die Funktion „Adjunkte“ benötigt folgende Funktionen:

1. Determinante
2. Unterdeterminante
3. transponiert

Zu Beginn wird die vorliegende Matrix nach quadratischem Zustand abgefragt, wenn eine quadratische Matrix vorliegt, startet der Prozess.
Um die Adjunkte einer Matrix zu bilden, werden sämtliche Unterdeterminanten der Matrix A(j,i) benötig.
Die benötigten Werte zur Bildung der Unterdeterminante werden in der Hilfsfunktion „Unterdeterminante“ zusammengefast.
Die Hilfsfunktion gibt die benötigten Elemente in Form einer Matrix A(j-1,i-1) zurück. Die Matrix wird an die Funktion „determinante“ übergeben. Der zurückgegebene Determinantenwert der Matrix A(j-1,i-1) wird der Matrix A(j,i) zugeordnet.
Zum Schluss wird die erstellte Matrix A(j,i) transponiert und zurückgegeben.

Anmerkung

Die Funktion ist bei großen Matrizen ineffizient, da die Schleifendurchläufe enorm sind. Alternativ wäre, die Inverse über das Gaußsche Eliminationsverfahren zu bilden.

5.3 Ausnahmebehandlung

Die Klasse MatrixMath greift bei den von dieser Klasse ausgelösten Ausnahmebehandlungen auf die Klasse MatrixException zurück. Dabei sind die Fehlernummer 300 bis 399 für die Klasse MatrixMath vorgesehen. Derzeit sind folgende Ausnahmen deklariert:

Fehlernummer	Fehlermeldung	auslösende Methode
300	Es liegt keine quadratische Matrix vor.	determinante adjunkte
301	Zu dieser Matrix gibt es keine Inverse (Determinante = 0).	inverse

5.4 Test

Getestet mit:

Microsoft Visual Studio 2010

Version 10.0.30319.1 RTMRel

Microsoft .NET Framework

Version 4.0.30319 RTMRel

Installierte Version: VC Express

5.4.1 Testfälle

Nr.	TF	Erw. Ausgabe
1	Determinante mit float $\begin{bmatrix}0 & 0 & 1\\2 & 4 & 6\\1 & 2 & 8\end{bmatrix}$	0
2	Determinante mit float $\begin{bmatrix}0 & 0 & 0\\2 & 4 & 6\\1 & 2 & 8\end{bmatrix}$	0
3	Determinante mit float $\begin{bmatrix}0 & 0 & 1\\2 & 0 & 6\\1 & 2 & 0\end{bmatrix}$	4
4	Determinante mit float $\begin{bmatrix}2 & 0 & 1\\4 & 2 & 6\\2 & 1 & 8\end{bmatrix}$	20
5	Inverse mit float $\begin{bmatrix}-1 & 3 & -1\\5 & -1 & 1\\1 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}0.5 & 0.5 & -1\\-0.5 & -0.5 & 2\\-3 & -2 & 7\end{bmatrix}$
6	Determinante mit float $\begin{bmatrix}1 & 3\\2 & 2\\3 & 1\end{bmatrix}$	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
7	Adjunkte mit float $\begin{bmatrix}1 & 3\\2 & 2\\3 & 1\end{bmatrix}$	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
8	Inverse mit float $\begin{bmatrix}0 & 0 & 0\\2 & 4 & 6\\1 & 2 & 8\end{bmatrix}$	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).
9	Determinante mit double $\begin{bmatrix}0 & 0 & 1\\2 & 4 & 6\\1 & 2 & 8\end{bmatrix}$	0
10	Determinante mit double $\begin{bmatrix}0 & 0 & 0\\2 & 4 & 6\\1 & 2 & 8\end{bmatrix}$	0
11	Determinante mit double $\begin{bmatrix}0 & 0 & 1\\2 & 0 & 6\\1 & 2 & 0\end{bmatrix}$	4

Nr.	TF	Erw. Ausgabe
12	Determinante mit double $\begin{bmatrix} 2 & 0 & 1 \\ 4 & 2 & 6 \\ 2 & 1 & 8 \end{bmatrix}$	20
13	Inverse mit double $\begin{bmatrix} -1 & 3 & -1 \\ 5 & -1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.5 & 0.5 & -1 \\ -0.5 & -0.5 & 2 \\ -3 & -2 & 7 \end{bmatrix}$
14	Determinante mit double $\begin{bmatrix} 1 & 3 \\ 2 & 2 \\ 3 & 1 \end{bmatrix}$	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
15	Adjunkte mit double $\begin{bmatrix} 1 & 3 \\ 2 & 2 \\ 3 & 1 \end{bmatrix}$	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
16	Inverse mit double $\begin{bmatrix} 0 & 0 & 0 \\ 2 & 4 & 6 \\ 1 & 2 & 8 \end{bmatrix}$	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).
17	Determinante mit bruch $\begin{bmatrix} 0/0 & 0/0 & 1/2 \\ 2/4 & 4/7 & 6/8 \\ 1/2 & 2/3 & 8/7 \end{bmatrix}$	1/42
18	Determinante mit bruch $\begin{bmatrix} 1/2 & 2/4 & 3/7 \\ 2/9 & 4/5 & 6/8 \\ 1/2 & 2/3 & 8/10 \end{bmatrix}$	23/144
19	Inverse mit bruch $\begin{bmatrix} -1/2 & 3/8 & -1/2 \\ 5/12 & -1/6 & 1/8 \\ 1/5 & 1/3 & 2/7 \end{bmatrix}$	1800/1547 5520/1547 105/221 1896/1547 864/1547 420/221 -496/221 -696/221 210/221
20	Determinante mit bruch $\begin{bmatrix} 1/2 & 3/2 \\ 2/9 & 2/8 \\ 3/2 & 1/3 \end{bmatrix}$	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
21	Adjunkte mit bruch $\begin{bmatrix} 1/7 & 3/2 \\ 2/8 & 2/9 \\ 3 & 1 \end{bmatrix}$	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
22	Inverse mit bruch $\begin{bmatrix} 1/1 & 1/1 & 1/1 \\ 2/1 & 2/1 & 2/1 \\ 1/9 & 2/8 & 8/9 \end{bmatrix}$	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).
23	Determinante komplex $(2,3)\ \ (1,-3)\ \ (4,-2)$ $(1,-8)\ \ (6,2)\ \ (2,-1)$ $(1,1)\ \ \ (1,5)\ \ (1,2)$	(108,-74)

Nr.	TF	Erw. Ausgabe

24	Determinante mit komplex (0,0) (0,0) (0,0) (2,1) (7,1) (2,8) (−2,9)(1,8) (−9,8)	(0,0)
25	Determinante komplex (0,0) (0,0) (1,2) (2,1) (0,0) (2,8) (−2,9)(1,8) (0,0)	(-40,5)
26	Determinante komplex (1,1) (2,−1) (1,−2) (2,−1) (1,2) (2,−1) (2,1) (2,2) (1,2)	20
27	Adjunkte komplex (2,3) (1,−3) (4,−2) (1,−8) (6,2) (2,−1) (1,1) (1,5) (1,2)	(−5,5) (7,19) (−29,−3) (−14,7) (−10,5) (−19,−38) (37,−11) (17,−15) (29,33)
28	Adjunkte komplex (1,1) (2,−1) (1,−2) (2,−1) (1,2) (2,−1) (2,1) (2,2) (1,2)	(−9,2) (2,−5) (−2,−4) (1,−3) (−5,6) (−3,−6) (6,−3) (5,−4) (−4,7)
29	Determinante komplex (2,3) (1,−3) (2,3) (1,−3) (2,3) (1,−3)	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
30	Adjunkte komplex (2,3) (1,−3) (2,3) (1,−3) (2,3) (1,−3)	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.
31	Inverse komplex (0,0) (0,0) (0,0) (2,1) (7,1) (2,8) (−2,9)(1,8) (−9,8)	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).

Extremwerttest

E1	Determinante mit float	1474

$$\begin{bmatrix} 2 & -1 & -2 & -5 & 2 & 0 \\ -2 & 2 & 2 & 1 & 0 & 3 \\ 1 & 4 & 4 & 0 & -3 & -3 \\ 0 & 2 & 4 & -3 & -2 & 2 \\ -4 & -4 & 3 & -3 & -3 & 3 \\ 2 & 1 & 2 & -2 & -2 & 4 \end{bmatrix}$$

E2	Determinante mit Double 2 4 -5 3 3 -4 3 4 4 -3 3 -4 -3 -5 0 0 2 -4 -3 1 2 4 1 -5 3 2 4 4 4 -1 -2 -2 3 -2 2 -1 0 0 4 3 3 -5 -1 -5 -2 2 3 -5 1 4 -1 -3 0 0 -3 0 4 -4 1 3 1 -5 4 4 0 -4 3 -1 -1 -5 -3 -3 3 1 4 0 4 -5 2 0 2 -4 -3 2 -4 3 1 1 -4 -1 2 1 -5 2 -5 4 -1 0 -1 -3	-85025626
E3	Determinante mit Bruch 1/2 2/4 3/7 1/2 2/3 8/7 2/9 4/5 6/8 4/5 2/9 4/5 6/8 2/4 3/7 1/2 2/3 8/7 2/9 4/5 1/2 2/3 8/7 1/2 2/3 8/7 1/2 2/4 3/7 2/4 1/2 2/3 8/7 1/2 3/7 1/2 2/3 8/7 2/9 4/5 2/4 3/7 2/4 1/2 3/7 1/2 2/3 1/2 2/4 3/7 8/7 1/2 2/4 3/7 2/4 1/2 3/7 1/2 2/3 8/7 3/7 1/2 2/3 1/2 2/4 3/7 1/2 3/7 1/2 2/3 1/2 2/3 8/7 1/2 3/7 6/8 2/4 3/7 1/2 2/3 3/7 1/2 2/3 8/7 2/9 4/5 1/2 2/4 3/7 1/2 6/8 2/3 8/7 2/9 4/5 6/8 4/5 3/7 1/2 2/3	257804657785218855/1146665551578284032
E4	Inverse mit Bruch 1/2 2/4 3/7 1/2 2/3 2/9 4/5 6/8 2/4 3/7 1/2 2/3 8/7 1/2 2/3 1/2 2/3 8/7 1/2 3/7 2/4 3/7 2/4 1/2 3/7	9486/925 -18/5 -9486/925 2367/185 -1701/185 378/37 0 -378/37 420/37 -420/37 -140/37 0 140/37 -98/37 98/37 -14086/925 18/5 10756/925 -3011/185 3381/185 0 0 21/5 -21/5 0
E5	Determinante mit float -5 -4 -1 -2 2 -4 -2 4 -2 3 2 0 -5 3 -5 0 1 -4 -3 0 -3 1 -4 -4 0 -1 -5 -5 4 -5 -2 -4 -2 1 3 -5 -3 -3 4 -3 -2 2 -4 0 2 -5 -1 2 -4 -4 -2 3 -3 1 -1 -1 -3 -2 -1 -5 1 -2 -2 -5 -5 -5 3 -1 -2 4 -2 0 4 1 0 0 3 2 -2 3 -4 2 0 3 1 4 -5 -2 -5 -5 4 -1 4 3 3 0 3 -2 -2 -1	-7762391

5.4.2 Testprotokoll

 - bestanden  -nicht bestanden  - Bug

Nr.	TF	Erg.	Akt. Ausgabe	Datum	Version
1	1		0	15.06.2010	1.0
2	2		0	15.06.2010	1.0
3	3		4	15.06.2010	1.0
4	4		20	15.06.2010	1.0
5	E1		1354	15.06.2010	1.0
6	5			15.06.2010	1.0
7	6		-1.#IND	15.06.2010	1.0
8	7		Programm stürzt ab	15.06.2010	1.0
9	8		Programm stürzt ab	15.06.2010	1.0
10	6		Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	18.06.2010	1.8.1
11	7		Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	18.06.2010	1.8.1
12	8		Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).	18.06.2010	1.8.1

- bestanden -nicht bestanden - Bug

Nr.	TF	Erg.	Akt. Ausgabe	Datum	Version
13	9	[2]	-0	21.06.2010	1.8.6
14	10	✓	0	21.06.2010	1.8.6
15	11	✓	4	21.06.2010	1.8.6
16	12	✓	20	21.06.2010	1.8.6
17	E2	✓	-8.50256e+007	21.06.2010	1.8.6
18	13	✓		21.06.2010	1.8.6
19	14	✓	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	21.06.2010	1.8.6
20	15	✓	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	21.06.2010	1.8.6
21	16	✓	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).	21.06.2010	1.8.6

[2] Funktion fix_zero führt nach jeden Spaltentausch ein Vorzeichen Wechsel aus

✔ - bestanden ✖ -nicht bestanden 🐞 - Bug

Nr.	TF	Erg.	Akt. Ausgabe	Datum	Version
22	17	✖ [3]	Integer division by zero	21.06.2010	1.8.6
23	18	✔	23/144	21.06.2010	1.8.6
24	E3	✔	257804657785218855/ 1146665551578284032	21.06.2010	1.8.6
18	19	✔	1800/1547 5520/1547 105/221 1896/1547 864/1547 420/221 -496/221 -696/221 210/221	21.06.2010	1.8.6
22	20	✔	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	21.06.2010	1.8.6
23	21	✔	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	21.06.2010	1.8.6
24	22	✔	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).	21.06.2010	1.8.6

[3] Eingaben wie 0/0 oder 0 werden in der Klasse Bruch nicht gesondert gehandelt

- bestanden -nicht bestanden - Bug

Nr.	TF	Erg.	Akt. Ausgabe	Datum	Version
25	23	✓	(108,-74)	21.06.2010	1.8.6
26	24	🐞 [4]	(-0,0)	21.06.2010	1.8.6
27	25	✓	(-40,5)	21.06.2010	1.8.6
28	27	✓		21.06.2010	1.8.6
29	28	✓		21.06.2010	1.8.6
30	26	✓		21.06.2010	1.8.6
31	29	✓	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	21.06.2010	1.8.6
32	30	✓	Es ist folgender Fehler aufgetreten: Es liegt keine quadratische Matrix vor.	21.06.2010	1.8.6
33	31	✓	Es ist folgender Fehler aufgetreten: Zu dieser Matrix gibt es keine Inverse (Determinante = 0).	21.06.2010	1.8.6
34	E4	✓	Wie bei TF E4	21.06.2010	1.8.6
35	E5	✓	-7.76239e+006	21.06.2010	1.8.6
36	9	✓		21.06.2010	1.8.7
37	24	✓	(0,0)	21.06.2010	1.8.7

[4] Funktion fix_zero führt nach jeden Spaltentausch ein Vorzeichen Wechsel aus

6 Klasse Matrix

Die Klasse Matrix ist durch Mehrfachvererbung (Multiple-inheritance) von den Klassen MatrixArithmetic und MatrixMath abgeleitet und fasst diese somit zu einer Klasse zusammen. Des Weiteren enthält Sie den Gleichungslöser (solve). Die Klasse MatrixVector wurde als virtuelle Klasse eingerichtet, sodass bei der Ableitung diese nicht zweimal vorhanden ist, sondern es nur eine Instanz gibt.

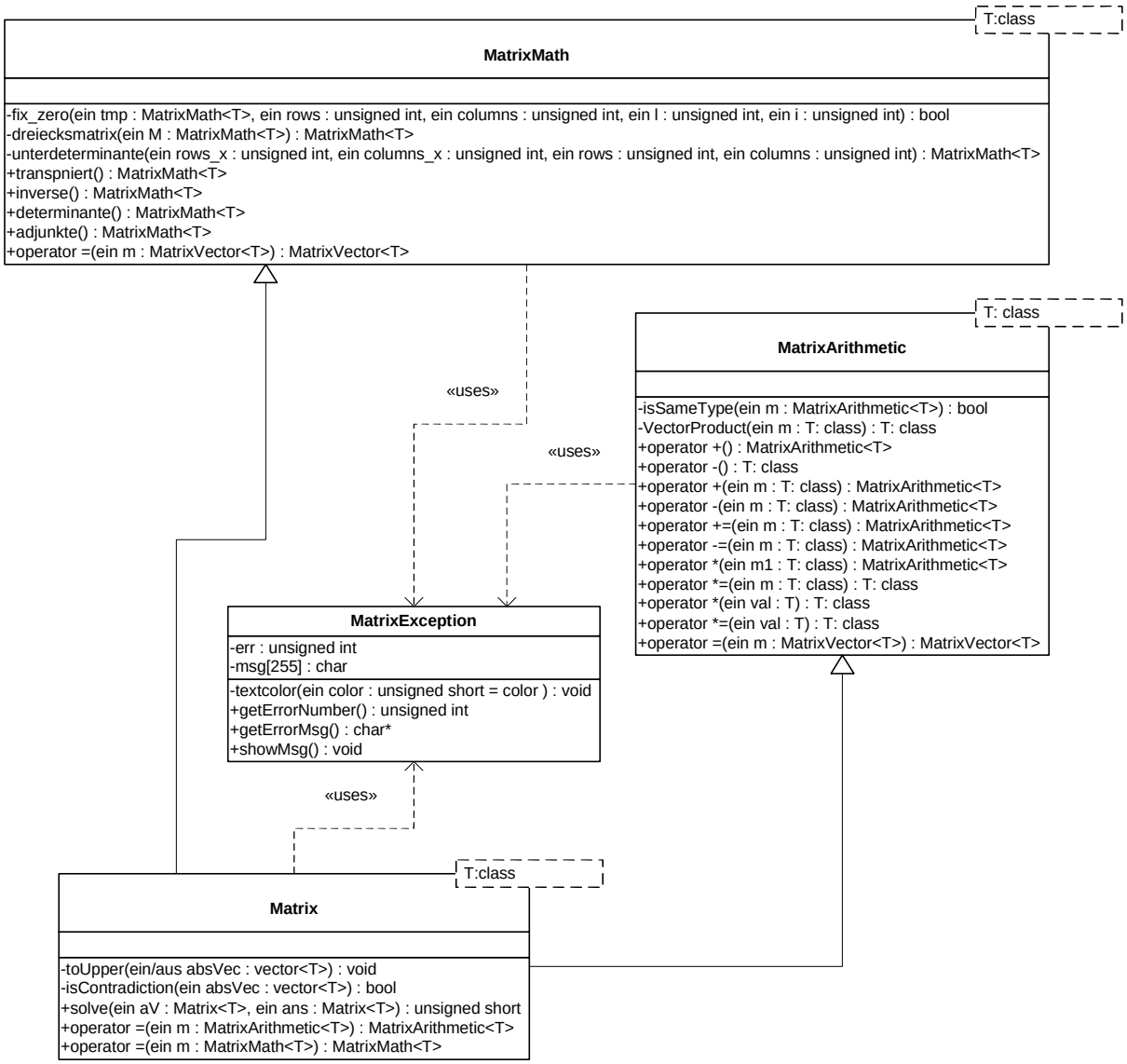

Bild 2 Klassendiagramm Matrix[5]

6.1 Attribute

Es wurden keine Attribute verwendet, da auf die Attribute der Klasse MatrixVector zurückgegriffen wird und weitere Attribute nicht notwendig sind.

[5] Das Klassendiagramm wurde mithilfe des Programmes *Microsoft® Office Visio® Professional 2003 (11.8323.8324) SP3* erstellt.

6.2 Methoden

Die Klasse Matrix stellt mit der Methode „solve“ den Gleichungslöser zur Verfügung.

unsigned short solve (Matrix& c, Matrix& x)

Das folgende Beispiel soll zeigen wie in einem Programm der Gleichungslöser verwendet wird.

```
//     Beispiel zur Verwendung des Gleichungslösers

#include "Matrix.h"

int main()
{
       Matrix<float> A(3,3);       // Koeffizientenmatrix
       Matrix<float> x(3,1);       // Lösungsvektor
       Matrix<float> c(3,1);       // Spaltenvektor aus den absoluten Gliedern

       cin >> A;     // Eingabe der Koeffizienten
       cin >> c;     // Eingabe der absoluten Glieder

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

       catch (MatrixException ex) { ex.showMsg(); }

       switch(status)
       {
              case   0 :  cout  << "keine Lösung" << endl; break;
              case   1 :  cout << x << endl;  break;
              case   2 :  cout << "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 dem Gauß – Algorithmus um, wie folgendes Beispiel zeigt.

Als Beispiel wird von folgendem linearen (inhomogenen) Gleichungssystem ausgegangen:

$$2a - 4b - 10c = -38$$
$$-a + 3b - 2c = -1$$
$$-3a - b + 3c = 14$$

Daraus ergibt sich die Koeffizientenmatrix

$$A = \begin{pmatrix} 2 & -4 & -10 \\ -1 & 3 & -2 \\ -3 & -1 & 3 \end{pmatrix}$$

und der Spaltenvektor mit den absoluten Gliedern

$$c = \begin{pmatrix} -38 \\ -1 \\ 14 \end{pmatrix}$$

.

Durch die Funktion toUpper, ergibt sich die Koeffizientenmatrix als obere Dreiecksmatrix.

$$A = \begin{pmatrix} 2 & -4 & -10 \\ 0 & 1 & -7 \\ 0 & 0 & -61 \end{pmatrix} \quad c = \begin{pmatrix} -38 \\ -20 \\ -183 \end{pmatrix}$$

Dieses Ergebnis wird der Methode „solve“ zurückgegeben. Diese bildet durch multiplizieren der Diagonalelemente die Determinante. Ist die Determinante ´nicht gleich Null, liegt genau eine Lösung vor und die Verarbeitung der Funktion „solve“ wird fortgesetzt.

Sollte die Determinante Null ergeben, wird in die (private) Methode „isContradiction“ gesprungen und geprüft ob ein Wiederspruch vorliegt. Einen Wiederspruch zeigt das folgende Beispiel:

$$\begin{array}{ccc|c} 2 & -4 & -10 & -38 \\ 0 & 1 & -7 & -20 \\ 0 & 0 & 0 & -183 \end{array}$$

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 definiert ist. Somit liegt also ein Wiederspruch vor. Ist dies der Fall, wird die weitere Bearbeitung gestoppt und der Methode „solve“ eine Null zurückgegeben. Diese wiederum bricht die weitere Verarbeitung der Methode ab und gibt die Null, an das aufrufende Programm, als Status zurück.

Sollte jedoch kein Widerspruch vorliegen, so ergibt sich, dass es unendlich viele Lösungen gibt. In diesem Fall wird, über die zuvor beschriebene Weise, dem aufrufenden Programm der Status 2 zurückgegeben.

Sollte jedoch die Determinante nicht gleich Null gewesen sein, wird mit der Verarbeitung der Methode „solve“ fortgefahren und die Koeffizientenmatrix in eine Einheitsmatrix umgewandelt, womit im Anschluss im Vektor der absoluten Gliedern die Lösungen des Gleichungssystems stehen.

Für unser Beispiel ergibt sich:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad c = \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix}$$

Die Methode „solve“ gibt in diesem Fall eine 1 für den Status der Verarbeitung zurück und dem Lösungsvektor x wird die Lösung übergeben.

Sodass das aufrufende Programm oder die aufrufende Funktion nur noch den Rückgabewert prüfen braucht und somit entweder eine Meldung (unendlich viele Lösungen / keine Lösung) oder das Ergebnis ausgeben kann.

Hier noch mal eine Übersicht der möglichen Statuscodes:

Rückgabewert	Deutung
0	Es gibt keine Lösung gefunden werden.
1	Es gibt genau eine Lösung, welche dem Lösungsvektor übergeben wurde
2	Es gibt unendlich viele Lösungen.

6.3 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 in der Klasse Matrix 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 überein.	

6.4 Test

6.4.1 Testfälle

Die Testfälle sind in drei Typen eingeteilt:

- Funktionstest
- Extremfalltest
- Exceptiontest
- Massentest für die Datentypen float, double, complex<double> und Bruch

Die Funktionstests dienen dem Test auf Funktionalität des Gleichungslösers. Die Extremwerttests wird die Funktion „solve“ auf ihr Verhalten im Umgang mit Gleichungssystemen, welche mit (für die Verarbeitung mit einem Computer) ungünstigen Werten gefüllt sind. Die Exceptiontests beziehen sich auf die von der Klasse Matrix verwendeten Ausnahmebehandlungen. In den Massentests soll die Genauigkeit des Gleichungslösers für die verschiedenen Datentypen festgestellt werden, um Rückschlüsse zu ziehen, welcher der günstigste Datentyp, für welchen Anwendungsfall ist.

6.4.1.1 Funktionstest

Funktionstest			
Nr.	Datentyp	Gleichungssystem	Beschreibung
1	Float	$\left(\begin{array}{ccc\|c} 2 & -4 & -10 & -38 \\ -1 & 3 & -2 & -1 \\ -3 & -1 & 3 & 14 \end{array}\right)$ Ergebnis: $\begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix}$	In diesem Test wird geprüft, ob der Gleichungslöser ein lösbares Gleichungssystem richtig löst.
2	Double		
3	complex<double>		
4	Bruch		
5	Float	$\left(\begin{array}{ccc\|c} 2 & -4 & -10 & -38 \\ -1 & 3 & -2 & -1 \\ -3 & 9 & -6 & -3 \end{array}\right)$ Ergebnis: $\left(\begin{array}{ccc\|c} 2 & -4 & -10 & -38 \\ 0 & 1 & -7 & -20 \\ 0 & 1 & -7 & -20 \end{array}\right)$ 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		

Funktionstest			
Nr.	Datentyp	Gleichungssystem	Beschreibung
9	Float	$\left(\begin{array}{ccc\|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 9 & 7 \end{array}\right)$ Ergebnis: $\left(\begin{array}{ccc\|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -2 \\ 0 & 0 & 0 & 4 \end{array}\right)$ Es ist zu erkennen, dass in der dritten Zeile ein Widerspruch vorliegt, denn$0 \cdot x_3 \neq 4$. 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		

6.4.1.2 Extremfalltest

Extremfalltest			
Nr.	Datentyp	Gleichungssystem	Beschreibung
13	Float	$\left(\begin{array}{ccc\|c} -1 & 3 & 1 & 1 \\ 2 & 0 & 4 & -2 \\ 3 & -1 & 5 & 2 \end{array}\right)$ Ergebnis: $\left(\begin{array}{ccc\|c} -1 & 3 & 1 & 1 \\ 0 & 6 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{array}\right)$ 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 $\frac{1}{3}$, welcher nicht als Gleitkommazahl aufgelöst werden kann, sondern nur gerundet wiedergegeben werden kann.
14	Double		
15	complex<double>		
16	Bruch		

Extremfalltest			
Nr.	Datentyp	Gleichungssystem	Beschreibung
17	Float	$\begin{array}{ccc\|c} 1 & 2 & 3 & 4 \\ 3 & 10 & 14 & 15 \\ 2 & 12 & 1000016 & 15 \end{array}$ Ergebnis: $\begin{pmatrix} \frac{4999999}{2000000} \\ \frac{599999}{800000} \\ \frac{1}{1000000} \end{pmatrix}$	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		

6.4.1.3 Exceptiontest

Exceptiontest		
Nr.	Exception	Beschreibung
21	400 „Die Matrix mit den absoluten Gliedern ist kein Spaltenvektor“	Hier wird die Funktionalität, der Ausnahmebehandlung 400 getestet. Diese wird ausgegeben wenn die als Parameter übergebene Matrix mit den absoluten Gliedern sich nicht als Spaltenvektor darstellt.
22	401 „Die Ergebnismatrix ist kein Spaltenvektor“	Die MatrixException 401 wird ausgelöst, wenn sich die als Parameter übergebene Matrix für die Lösungen sich nicht als Spaltenvektor darstellt.
23	402 „Die Anzahl absoluter Glieder stimmt nicht mit der Zeilenanzahl des Lösungsvektors überein.“	Diese Ausnahmebehandlung wird ausgelöst, wenn die Matrizen mit den Lösungen und absoluten Gliedern nicht vom selben Typ sind.
24	403 „Die Anzahl der absoluten Glieder stimmt nicht mit der Anzahl Gleichungen in der Koeffizientenmatrix überein.“	Sollte die Zeilenanzahl (Anzahl Gleichungen) nicht mit der Zeilenanzahl (Anzahl absolute Glieder) der absoluten Glieder übereinstimmen, wird die Ausnahmebehandlung 403 ausgelöst.

6.4.1.4 Massentest

Der Massentest wurde für folgende Datentypen durchgeführt:

- float
- double
- complex<double>
- Bruch

Dabei wurden 10000 Gleichungssysteme in zufälliger Dimension (Koeffizienten), zufälligen Zahlenbereich (0 – 1000000) und zufälligen Zahlen erzeugt und durch eine Rückrechnung (Probe) auf die absoluten Glieder der Gleichungssysteme überprüft und die Abweichung prozentual festgestellt.

6.4.2 Testprotokoll

Zunächst werden die

Testfall	Bestanden	Bemerkung	Datum	Version
1	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
2	Ja		16.06.2010	1.8.1
			22.06.2010	1.8.6
3	Ja		16.06.2010	1.8.1
			22.06.2010	1.8.6
4	Ja		16.06.2010	1.8.1
			22.06.2010	1.8.6
5	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
6	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
7	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
8	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
9	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
10	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
11	Ja		16.06.2010	1.8.1
			22.062010	1.8.6
12	Ja		16.06.2010	1.8.1
			22.062010	1.8.6

Testfall	Bestanden	Bemerkung	Datum	Version
13	Nein	Der Test gab vor, dass zu erkennen ist, dass es sich um ein nichtlösbares Gleichungssystem handelt. Dies wurde nicht erkannt, sondern als Lösung $\begin{pmatrix} 4.1943 \cdot 10^7 \\ 2.09715 \cdot 10^7 \\ -2.09715 \cdot 10^7 \end{pmatrix}$ ausgegeben.	16.06.2010 22.062010	1.8.1 1.8.6
14	Ja		16.06.2010 22.062010	1.8.1 1.8.6
15	Ja		16.06.2010 22.062010	1.8.1 1.8.6
16	Ja		16.06.2010 22.062010	1.8.1 1.8.6
17	Ja	Es werden die Lösungen wie folgt gerundet: $\begin{pmatrix} 2{,}5 \\ 0{,}749999 \\ 1 \cdot 10^{-6} \end{pmatrix}$ Dies entspricht einer Abweichung um, $\begin{pmatrix} 5 \cdot 10^7 \\ 2{,}5 \cdot 10^{-7} \\ 0 \end{pmatrix}$ vom tatsächlichen Ergebnis. x_1 und x_2 werden zwar gerundet, entsprechen aber den vorgegebenen Ergebnissen, wenn eine Genauigkeit kleiner 10^{-6} gefordert ist.	16.06.2010 22.062010	1.8.1 1.8.6
18	Ja		16.06.2010 22.062010	1.8.1 1.8.6
19	Ja		16.06.2010 22.062010	1.8.1 1.8.6

Testfall	Bestanden	Bemerkung	Datum	Version
20	Nein	Ausgegebene Lösung: $\begin{pmatrix} -\frac{11368437}{8875648} \\ \frac{599999}{800000} \\ \frac{1}{1000000} \end{pmatrix}$ 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 $5{,}99999 \cdot 10^{11}$. 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: $\begin{pmatrix} \frac{4999999}{2000000} \\ \frac{599999}{800000} \\ \frac{1}{1000000} \end{pmatrix}$ 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 22.062010	1.8.2 1.8.6
21	Ja		22.062010	1.8.6
22	Ja		22.062010	1.8.6
23	Ja		22.062010	1.8.6
24	Ja		22.062010	1.8.6

6.4.3 Auswertung Massentest

Der Massentest diente der Ermittlung, wie stark die Ergebnisse des Gleichungslösers von tatsächlichen Ergebnis abweichen. Dabei ergaben sich für die verwendeten Datentypen folgende mittlere Abweichungen:

Datentyp	Mittlere Abweichung in Prozent
float	1,12
double	$3{,}13 \cdot 10^{-8}$
complex<double>	$1{,}26 \cdot 10^{-7}$
Bruch	9,17

Dabei ist deutlich zu erkennen, dass der Datentyp double die genauesten Ergebnisse liefert. Sollte es auf weniger Genauigkeit ankommen, kann auch auf den Datentyp float zurückgegriffen werden.

Kritisch anzusehen ist der Datentyp Bruch. Hier ist zu überlegen, woher die hohe Ungenauigkeit von über 9 % kommt. Eine mögliche Ursache, welches einer genaueren Untersuchung bedarf, sind die hohen Zahlenwerte im Nenner und Zähler, welche bei der Lösung des Gleichungssystems entstehen. Hier liegt die Vermutung nahe, dass es zu Bereichsüberschreitungen kommt.

Eine mögliche Abhilfe könnte, dass ersetzen des Datentyps long long für Zähler und Nenner durch den Datentyp double sein, welcher weit höhere Zahlen darstellen kann.

Ansonsten lässt der Massentest für den Datentyp Bruch trotzdem erkenne, dass es der genaueste Datentyp ist, denn da wo es nicht zu hohen Zahlenwerten im Nenner und Zähler kommt, liegt die Abweichung von der tatsächlichen Lösung bei 0%.

6.4.4 Aussichten für eine weitere Entwicklung

Der Gleichungslöser wurde nach dem Gauß-Jordan-Verfahren entwickelt. Leider ist in den Tests zu erkennen, dass dieses Verfahren seine Schwachstellen hat.

Diese liegen in der Erkennung von unendlich vielen Lösungen und keiner Lösung, sowie in doch teilweise hohen Abweichung vom tatsächlichen Ergebnis.

Da das Erkennen von möglichen oder keinen Lösungen mithilfe der Determinante erfolgt, liegt die Schwachstelle in den Rundungsfehlern der verwendeten Datentypen, da diese nicht in jedem Fall eine Null liefern, sondern eine minimale von der Null abweichendes Ergebnis. Somit nimmt der Gleichungslöser fälschlicherweise an, dass es sich um ein Gleichungssystem mit nur einer Lösung handelt. Eine mögliche Abhilfe könnte hier, das einrichten eines Bereiches um die Null herum, welcher noch zu definieren wäre, wo das Ergebnis als Null angenommen wird. Dies könnte auch durch den Nutzer der Klasse Matrix geschehen, indem er dem Gleichungslöser mittels Parameter den Bereich übergibt.

Das Gleiche Problem liegt auch im weiteren Erkennen (bei Determinante gleich Null) auf keine Lösung oder unendlich vielen Lösungen vor. Auch hier ist das Problem in der Abweichung vom Ergebnis Null. Abhilfe könnte das ersetzen der Prüfung auf Wiederspruch, durch das Berechnen des Ranges der Matrizen schaffen, welche auch als hinreichende Bedingung zum erkennen auf unendlich vielen Lösungen oder keiner Lösung einsetzbar ist.

Eine weitere Verbesserung des Gleichungslöser liegt bei den unendlichen Lösungen. Hier wäre eine weitere Angabe, als nur das es unendlich viele Lösungen gibt, welcher Koeffizient für die unendlichen Lösungen verantwortlich ist und die entsprechende Ausgabe des Lösungsvektors.

7 Klasse MatrixException

Die Klasse MatrixException dient den Matrixklassen zum auslösen von Ausnahmebehandlungen.

7.1 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 ein Datenfeld vom Typ char und ist auf 255 Zeichen begrenzt. Es beinhaltet die Fehlermeldung.

7.2 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.

7.3 Ausnahmebehandlung

Für die Klasse MatrixException sind keine Ausnahmebehandlungen vorgesehen.

7.4 Test

Die Ausnahmebehandlungen der Klasse Matrix werden in den jeweiligen Tests der Klasse mit vorgenommen.

8 Verzeichnisse

8.1 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