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.

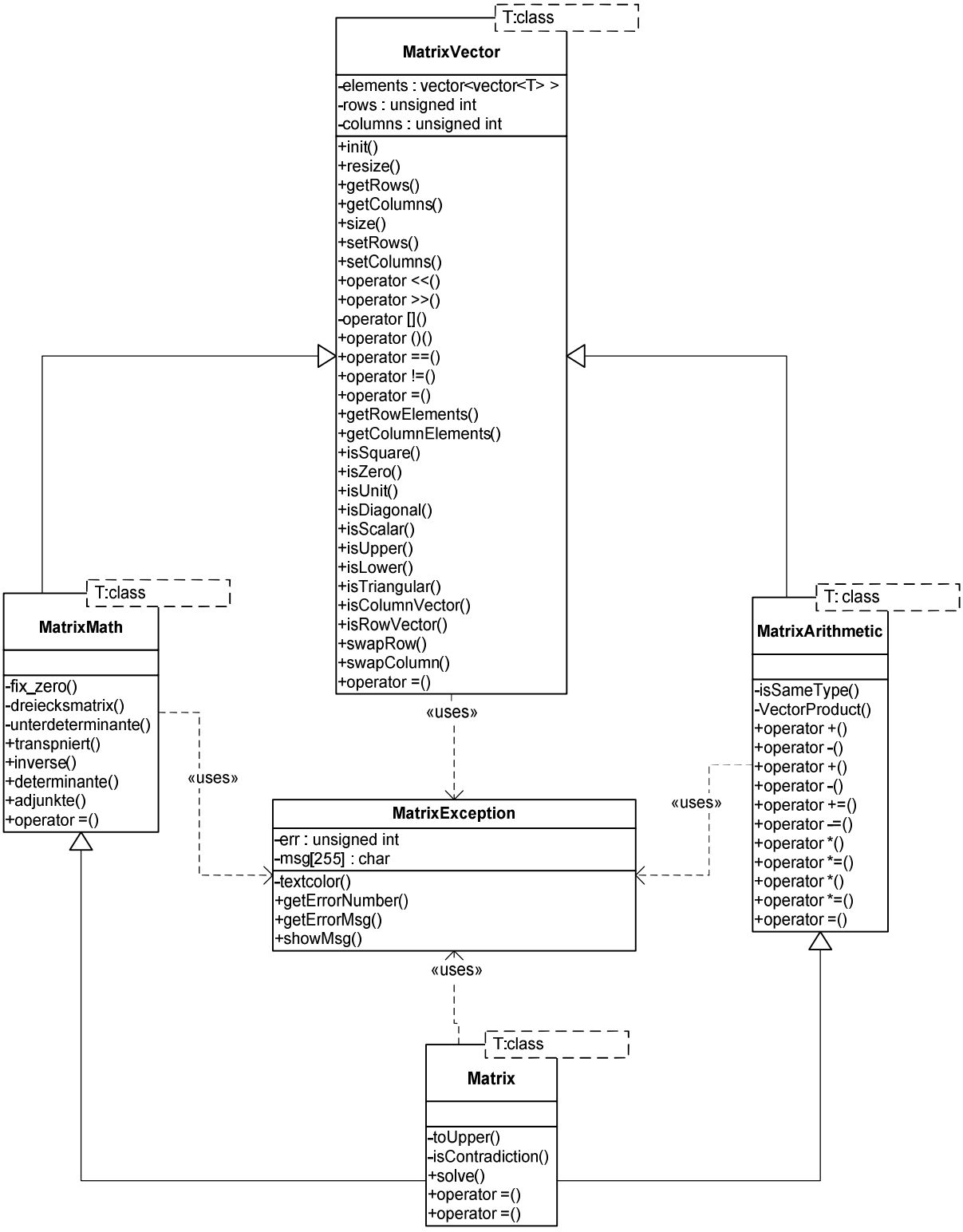

Bild 1 Klassendiagramm Übersicht[1]

[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 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.

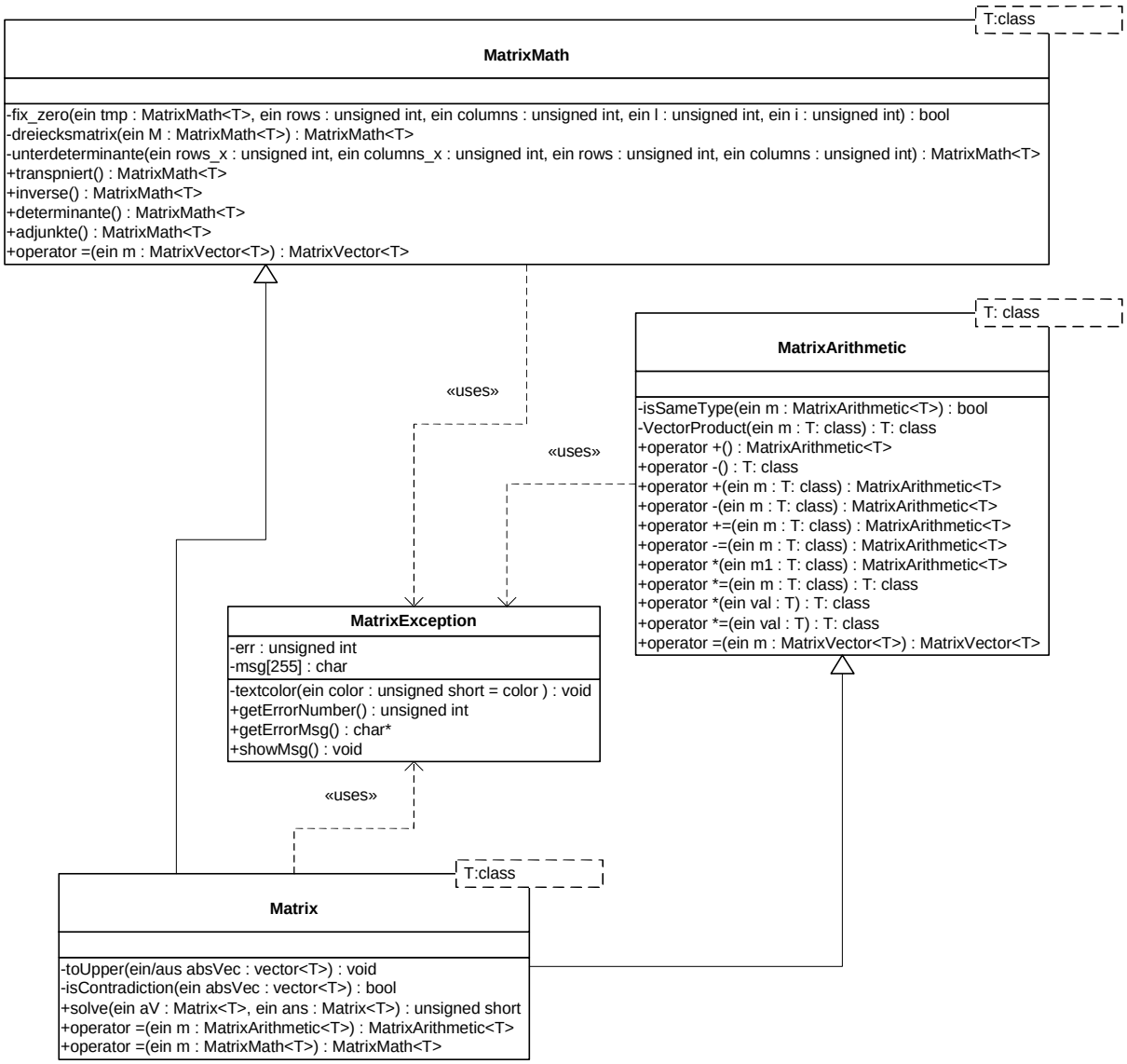

Bild 2 Klassendiagramm Matrix[2]

3.1 Attribute

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

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

3.2 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.

3.3 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.	

3.4 Test

3.4.1 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.

3.4.1.1 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		

3.4.1.2 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	$\begin{array}{cccc\|c} 1 & 2 & 3 & 4 \\ 3 & 10 & 14 & 15 \\ 2 & 12 & 1000016 & 15 \end{array}$ 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		

3.4.1.3 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.

3.4.1.4 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.

3.4.2 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

3.4.3 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%.

3.4.4 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.

4 Klasse MatrixException

Die Klasse MatrixException dient den Matrixklassen zum auslösen von Ausnahmebehandlungen.

4.1 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.

4.2 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.

4.3 Ausnahmebehandlung

Für die Klasse MatrixException sind keine Ausnahmebehandlungen vorgesehen.

4.4 Test

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

5 Verzeichnisse

5.1 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