Gottfried-Wilhelm-Leibniz-Universität Hannover
Fakultät für Mathematik und Physik
Institut für Algebraische Geometrie

Automatische Erzeugung von Aufgaben mit ganzzahligen Lösungen

Hausarbeit im Rahmen der Ersten Staatsprüfung für das Lehramt an Gymnasien im Land Niedersachsen

angefertigt im Prüfungsfach
Mathematik

vorgelegt von

Steffen Lünne

Erstgutachter: Prof. Dr. Hans-Christian Graf von Bothmer
Zweitgutachter: Prof. Dr. Joachim Engel

Bearbeitungszeitraum: 8.4.2006 bis 8.8.2006

Mein Dank gilt all jenen,
die mir bei der Erstellung dieser Arbeit geholfen haben,

besonders

Jan und Marian
für Ratschläge und Hilfestellungen bei der Programmierung mit Java,

Heike, Jan, Maike, Markus, Michael, Sabine, Sören und Tim
für die Korrektur der Arbeit und weitere Hinweise

sowie Sönke
für Ratschläge und Hilfe bei der Arbeit mit TEX.

Besonders möchte ich mich bei Herrn von Bothmer für die hervorragende Betreuung sowohl während der Bearbeitungszeit als auch schon davor bedanken.

Vorwort

Mit der fortschreitenden Weiterentwicklung von Software und der zunehmenden Nutzung des Internets bieten sich den Schülern[1] und Studenten heute immer mehr Möglichkeiten, sich für ihre Hausaufgaben und Hausübungen zu organisieren. Im Internet existieren jetzt schon ganze Seiten, auf denen sich zum Beispiel die Übersetzungen zu lateinischen Übungstexten aus den Lehrbüchern finden lassen. Selbst wenn die Verlage der betroffenen Bücher die Schließung dieser Seiten erreichen können, dauert es nicht lange und der Inhalt findet sich an anderer Stelle wieder. Zudem werden die Lösungen ganzer Übungsblätter digitalisiert und ins Internet gestellt bzw. per Email verschickt. Die Gefahr besteht, dass Schüler und Studenten in solchen Fällen ihre Aufgaben gar nicht mehr selber machen, sondern allerhöchstens noch abschreiben. Die im eigentlichen Sinne geforderte eigene Leistung wird damit auf ein Minimum reduziert.
Auch in der Mathematik wird ein solches Verhalten umso wahrscheinlicher, je mehr Textverarbeitungsprogramme mathematische Zeichensätze zur Verfügung stellen. Allerdings existiert mit OKUSON[2] schon ein Programm, das in der Lage ist, individuelle Übungsblätter zu erstellen und auch einen Großteil der Lösungen zu kontrollieren. Die Anzahl von Studenten ist jedoch relativ groß, sodass viel Zeit benötigt wird, um eine solche Menge von Aufgaben manuell zu erzeugen, bis die Übungsblätter als weitestgehend individualisiert gelten können. Meine Aufgabe war es deshalb, ein Programm zu entwickeln, das Aufgaben mit ganzzahligen Lösungen zu einem bestimmten Thema ge-

[1] Im Gegensatz zu anderen empfinde ich eine Auftrennung in Schülerinnen und Schüler oder Studentinnen und Studenten als überholt. Eine solche Auftrennung impliziert aus meiner Sicht statt Gleichberechtigung eher einen Unterschied, den es nicht geben sollte. Begriffe wie Schüler oder Studenten sind in dieser Arbeit also sowohl im Singular als auch im Plural als geschlechtsunabhängig zu verstehen.

[2] OKUSON ist ein Programm, das es erlaubt individuelle Übungsblätter für jeden Teilnehmer einer Übungsgruppe zu erstellen. Der Abruf der Übungsblätter erfolgt über das Internet. Die Kontrolle von bestimmten Übungsaufgaben kann ebenfalls automatisch geschehen, sofern diese bestimmte Bedingungen erfüllen. Für weitere Informationen siehe www.math.rwth-aachen.de/∼OKUSON/.

neriert. Dieses Programm sollte mit OKUSON kompatibel sein, um damit die gewünschte Individualisierung zu erreichen.

Bevor man sich jedoch an das Schreiben eines solchen Programms macht, das einfach irgendwelche Aufgaben und die Lösungen dazu ausgibt, sollte man sich Gedanken machen, wie solche Aufgaben überhaupt benutzt werden können und sie benutzt werden sollen. Dazu ist natürlich die Wahl des Bereiches, aus dem die Aufgaben stammen sollen, unerlässlich. In dieser Arbeit werde ich den Bereich der ähnlichen Matrizen behandeln.
An dieser Stelle kommt dann auch der zweite Teil der Aufgabenstellung ins Spiel. Die Lösungen der Aufgaben sollen ganzzahlig sein. Diese Einschränkung ist insbesondere dadurch motiviert, dass es sich mit ganzen Zahlen wesentlich leichter rechnen lässt und Angaben in den Lösungsfeldern für OKUSON so eindeutig werden oder man vor Schwierigkeiten in der Hinsicht steht, wie man eine Wurzel in ein Textfeld eingibt, ohne dass die Eingabe gegebenenfalls komplizierter als die Lösung der zugehörigen Aufgabe wird.

In Kapitel 1 dieser Arbeit beschäftige ich mich dann sowohl mit den Anforderungen an ein solches Programm in Bezug auf Lernziele und deren Kontrolle, als auch mit der Gestaltung von Übungen und Übungsaufgaben. Weitere Anforderungen an mein Programm und seine Grenzen werde ich in Kapitel 2 betrachten und dort am Schluss die gestellten Anforderungen zusammenfassen.

Mit Hilfe von Lernzielen möchte ich dann in Kapitel 3 den Bereich der ähnlichen Matrizen derart gliedern, dass eine sinnvolle Vorgehensweise beim Lernen dieses Bereiches erkennbar wird. Dazu stelle ich zuerst die zugehörige Theorie vor und definiere dann aus den dargestellten Zusammenhängen heraus grobe Lernziele. Ich möchte mit den Lernzielen lediglich Zwischenschritte und die jeweiligen Voraussetzungen etwas verdeutlichen und überlasse es dem Anwender, diese Lernziele so weit zu operationalisieren[3], dass man am Ende das Erreichen des Lernziels beobachten kann. Dazu fehlt mir ein geeigneter Maßstab. Außerdem ist es auch nicht das Ziel dieser Arbeit, einen solchen Maßstab zu finden.
Die Aufgabenstellungen des Programms werden im Anschluss an die Lernziele vorgestellt, ihre Optionen werden beschrieben und die Funktionsweise des Programms wird in Bezug auf Aufgabenstellungen und Optionen dargestellt.

Eine einfache Möglichkeit, solche Aufgaben wie die in Kapitel 3 vorgestellten zu generieren, ist es, mit den Lösungen zu starten und daraus die Aufgaben zu

[3]Die Definitionen dieses Begriffs kann man ab Seite 11 nachlesen.

bauen. Dafür ist es notwendig, zwei zueinander inverse quadratische Matrizen beliebiger Größe zu finden, die beide nur ganzzahlige Einträge haben. Alle vorgestellten Aufgaben basieren auf einem Algorithmus, der solche Matrizen finden kann. Kapitel 4 arbeitet die Theorie für einen solchen Algorithmus auf und stellt ihn vor. Es bildet damit den mathematischen Schwerpunkt dieser Arbeit.

In Kapitel 5 folgt eine Kurzbeschreibung des Programms, in der auch alle bisher nicht wesentlichen Optionen wie die Ausgabe der Aufgaben noch einmal betrachtet werden. Kapitel 6 enthält dann meine persönliche Auswertung in Bezug auf die Anforderungen, die ich in Kapitel 1 und 2 an mein Programm gestellt habe.

Der Quelltext meines Java-Programms und Überlegungen zu kongruenten Matrizen, die es leider nicht in die Endversion geschafft haben, bilden den Anhang.

Ziel dieser Arbeit soll also nicht nur die Entwicklung eines lauffähigen Programms sein, sondern auch die Aufarbeitung der Theorie im Hintergrund des Programms sowie eine begründete Darstellung der einzelnen Aufgabentypen und ihrer Optionen.

Inhaltsverzeichnis

Kapitel 1

Bedingungen zur Aufgabenstellung

1.1 Übungen im Lernprozess

> „Lernvorgänge, insbesondere auch solche mathematischen Inhalts, können (...) außerordentlich unterschiedlich aussehen: (...) Dessenungeachtet, stimmen sie aber in wesentlichen Merkmalen ihres zeitlichen Verlaufs überein - den sog. »Lernphasen«: Jeder (bewußte) Lernprozeß muß mehr oder weniger angestoßen werden und hat Schwierigkeiten zu überwinden; das Gelernte muß gefestigt, geübt und angewendet werden, um zum »Besitz« des Lernenden (d.h. einsatzfähig in späteren Situationen) zu werden.“ [1]

1.1.1 Der Lernprozess in Phasen

1. Phase der Motivation

Ein bewusster und beabsichtigter Lernprozess im kognitiven Bereich muss nach Zech [Zech, 2002, Seite 182] motiviert werden, es muss einen Grund geben, sich Neues anzueignen. Man unterscheidet zwischen Motiven, verstanden als Wertungsdispositionen, *„die sich in immer wiederkehrenden Grundsituationen menschlichen Lebens im Lauf der Entwicklung allmählich herausbilden“* [2], und Motivation, die situationsabhängig ist. Hier wechselwirken Situations- und Motivfaktoren. Ein Lernprozess wird also durch die Situation und die allgemeine Lebenseinstellung motiviert.[3]

[1] [Zech, 2002, Seite 181]
[2] [Zech, 2002, Seite 187]
[3] Genauere Betrachtungen finden sich in [Zech, 2002, Seite 187 ff.].

2. Phase der Schwierigkeiten

Ist der Lernprozess in Gang gesetzt worden, wird der Lernende im Allgemeinen an verschiedensten Stellen, was wiederum abhängig vom Lernprozess ist, auf diverse Schwierigkeiten stoßen. Diese Schwierigkeiten können zum Beispiel bei Vektoren darin liegen, dass es schwierig ist, sich Vektoren in einem n-dimensionalen Raum zu veranschaulichen, oder dass das Erlernen von mathematischen Definitionen wie das Lernen von Vokabeln schwer fällt. Schwierigkeiten des Lernenden äußern sich in Fehlern, die gemacht werden. Um solche Schwierigkeiten zu entdecken und auszuräumen, muss sowohl der Lernende als auch der Lehrende eine Möglichkeit zum Fehler Machen schaffen.[4]

3. Überwindung der Schwierigkeiten (Lösungsphase)

Schwierigkeiten können überwunden werden. Dies geschieht teilweise mit Hilfestellung von anderen, die mit Erklärungen, Beispielen und Veranschaulichungen dem Lernenden bei seinen Schwierigkeiten helfen. Auch das Lesen eines Buches zum entsprechenden Thema kann Schwierigkeiten beseitigen. Wesentlich ist die Rückmeldung an den Lernenden, der über seinen Fortschritt bei der Bewältigung seiner Schwierigkeiten informiert werden muss.[5] Während besonders jüngere Schüler hier noch auf die Hilfe ihrer Lehrer angewiesen sind, sollten Studenten eigenständig ihren Lernfortschritt reflektieren können, um gegebenenfalls entsprechende Maßnahmen zu ergreifen.

Für Zech [Zech, 2002, Seite 183] hängen die nun folgenden Phasen eng zusammen.

4. Sicherung des Gelernten

Eine einmalige Überwindung der Schwierigkeiten reicht in der Regel nicht aus, wenn auch in Zukunft auf das Erlernte zurückgegriffen werden soll. Es ist deshalb wichtig, das Gelernte auf irgendeine Art zu sichern. Zusammenfassungen des Gelernten an einem Beispiel können hierbei hilfreich sein, um das Gelernte dauerhaft im Gedächtnis zu behalten und es später nutzen zu können.[6]

[4] Vergleiche auch [Zech, 2002, Seite 182].
[5] Vergleiche auch [Zech, 2002, Seite 182 f.].
[6] Vergleiche auch [Zech, 2002, Seite 183].

5. Phase der Anwendung und Übung

Im Anschluss an die Sicherung des Gelernten wird es in verschiedenen Zusammenhängen angewendet und dadurch geübt.[7]

6. Transfer des Gelernten

> „Erst wenn der »Transfer« (die Anwendung des Gelernten in späteren (...) Lebenssituationen) gelingt, kann der Lernprozeß als erfolgreich abgeschlossen gelten.“ [8]

Der Sinn des Übens besteht also darin, das Gelernte später in anderen Zusammenhängen anzuwenden. Auch für Winter [Winter, 1993, Seite 4] liegt er im Transfer. Doch zwischen Beidem besteht eine wechselseitige Beziehung, denn *„Transfer setzt Übung, Übung setzt Transfer voraus. Übung und Transfer scheinen sich gegenseitig zu bedingen.“* [9] Soll das Gelernte in neuen Zusammenhängen oder überhaupt geübt werden, dann muss eine Transferleistung erbracht werden, weil das Lernen mathematischer Zusammenhänge nicht dadurch geschehen soll, dass man jeden einzelnen Fall wie die ersten Vokabeln einer Sprache auswendig lernt. Eher soll das Lernen so verlaufen, dass man sich mit Hilfe der ersten „Vokabeln“ ein Prinzip erschließt, welches dann auf die folgenden Situationen transferiert wird.[10]

Ein Ziel des Mathematikunterrichtes ist es, dass Schüler die Fähigkeit zum Problemlösen, Problemlösekompetenz, entwickeln.[11] Dazu müssen sie in der Lage sein, Gelerntes auf Neues zu übertragen.

> „*Problemlösen* (Regeln »höherer Ordnung« durch eigene Überlegung verstehen) (...) [Der] *Problemlöser* [lernt] durch *eigene* Überlegungen (...). Das bedeutet, daß der Lernende fähig ist, bereits gelernte Begriffe und Regeln zu einer neuen Regel zu kombinieren, d.h., das Gelernte in einer neuen Situation anzuwenden.“ [12]

1.1.2 Grundsätzliches zum Üben mathematischer Sachverhalte

Das Üben sollte nach Kampe [Kampe, 1993, Seite 18] folgendermaßen verstanden werden, *„als Arbeit mit Aufgaben, bei der die Schüler [Studenten]*

[7] Vergleiche auch [Zech, 2002, Seite 183].
[8] [Zech, 2002, Seite 184]
[9] [Winter, 1993, Seite 4]
[10] Vergleiche auch [Winter, 1993, Seite 4].
[11] Vergleiche auch [Kerncurriculum, 2006].
[12] [Zech, 2002, Seite 154], Hervorhebungen im Original

sowohl Wissen festigen und vertiefen als auch neues Wissen „entdecken" können, indem sie erworbenes Wissen zielgerichtet anwenden." [13] Es ergeben sich folgende Grundsätze bei der Gestaltung der Übung:

1. „Die Übung ist keine auf die Erarbeitung folgende Phase, sondern muß auch bereits bei der Gewinnung einer Zielstellung sowie bei der Erarbeitung neuen Wissens selbst vielfältige Tätigkeiten der Schüler [Studenten] auslösen.
2. Das Arbeiten mit Aufgaben in der Übung darf sich nicht auf das Festigen im Stoff beschränken, sondern muß auch die Anwendung von Wissen in unterschiedlichen Zusammenhängen sowie die Befähigung der Schüler [Studenten] zum Aufgabenlösen einschließen.
3. Beim Arbeiten mit Aufgaben in der Übung muß der ersten Lösungsphase, der Analyse und Ansatzfindung, besondere Bedeutung zukommen. Heuristische Vorgehensweisen sowie andere Hilfen müssen den Schülern [Studenten] begreifbar und nutzbar gemacht werden.
4. Beim Arbeiten mit Aufgaben in der Übung müssen die Schüler [Studenten] genügend Zeit und Gelegenheit haben, sich eingehend mit den Inhalten zu beschäftigen, notwendiges Wissen zu mobilisieren und eigene Gedanken zum Lösungsweg auszuprobieren."[14]

Diese Grundsätze dienen insbesondere dazu, die Entwicklung von Problemlösekompetenz zu unterstützen. Wichtig ist im Hinblick auf die automatische Erstellung von Aufgaben demzufolge der zweite Grundsatz. Die Aufgaben meines Programms sollten also nicht allein nur das Lösen einer Problemstellung erfordern, sondern das Wissen, das hinter den Aufgaben steckt, in unterschiedlichen Zusammenhängen abfragen, soweit dieses im Programm machbar ist. Es ist dabei darauf zu achten, dass man die Grundaufgaben nicht nur in einen „hübschen" Text verpackt, denn nach der Übersetzung des Textes in mathematische Formeln, die zwar Verständnis einfordert, bleibt es doch die gleiche Grundaufgabe.[15] Die Aufgaben sollten sich also dahingehend unterscheiden, dass bei Eingangsdaten, Lösungsweg und Ergebnis variiert wird, was gegeben und gesucht ist, weil man sonst Gefahr läuft, nur einen Lösungsweg und damit nur einen Algorithmus anzutrainieren, der dann

[13] [Kampe, 1993, Seite 18]

[14] [Kampe, 1993, Seite 18]

[15] Zum Beispiel ist die folgende Aufgabe nur ein zu lösendes lineares Gleichungssystem, dessen Koeffizientenmatrix zwei Zeilen und zwei Spalten hat: *Der Viehbestand eines Bauerhofes besteht nur aus Schafen und Gänsen. Die Tiere haben insgesamt 200 Beine und 150 Augen. Wie viele Schafe und Gänse gibt es auf dem Bauernhof?*

Tabelle der verschiedenen Aufgabentypen:

	Eingangsdaten	Lösungsweg	Ergebnis	
1.	gegeben	gegeben	gesucht	(Standard)
2.	gegeben	gesucht	gegeben	
3.	gegeben	gesucht	gesucht	
4.	gesucht	gegeben	gegeben	
5.	gesucht	gegeben	gesucht	
6.	gesucht	gesucht	gegeben	

Tabelle 1.1: Tabelle der verschiedenen Aufgabentypen in [Kampe, 1993, Seite 18]

abgespult wird, wenn alle Eingangsdaten passen. Die Lernenden würden somit in vielen Fällen wie ein Computerprogramm arbeiten, das zwar weiß, was es tun soll, aber nicht weiß, warum es etwas tun soll. Ein solches Verständnis ist in diesem Fall auch nicht notwendig, weil man es zur Lösung von Aufgaben nicht braucht, solange diese nur das Abspulen eines antrainierten Algorithmus verlangen. Gleichzeitig könnten die Lernenden dann in der Folge bei neuen Aufgaben wie ein Programm scheitern, wenn ihnen einfach nur ein einzelner Parameter bei der Angabe fehlt.

Natürlich habe ich es nicht in der Hand, zu verhindern, dass mein Programm auf diese Weise benutzt wird, allerdings habe ich mich bemüht, auch alternative Aufgaben zur Verfügung zu stellen, um die Gefahr eines solchen Einsatzes durch die Auswahl an Möglichkeiten etwas zu reduzieren. Einen Überblick über verschiedene Aufgabentypen gibt die Tabelle 1.1.[16]

Ich will mit dem folgenden Beispiel die Unterschiede veranschaulichen:

(1) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 5 \\ 6 \end{pmatrix} = ?$

(2) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} (?) \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$

(3) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} (?) \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = ?$

(4) $\begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix} = \begin{pmatrix} ? & 2 \\ 3 & ? \end{pmatrix} \circ \begin{pmatrix} 5 & ? \\ ? & 8 \end{pmatrix}$

[16] Vergleiche auch [Kampe, 1993, Seite 18 f.].

(5) $\begin{pmatrix} 4 & 1 \\ ? & 1 \end{pmatrix}^{-1} = ?$

(6) $\begin{pmatrix} 1 & -1 \\ -3 & 4 \end{pmatrix} = \left(\begin{pmatrix} 4 & 1 \\ ? & 1 \end{pmatrix} (?) \begin{pmatrix} 4 & 1 \\ ? & 1 \end{pmatrix} \right)^{?}$

Auch wenn solche Aufgabentypen die Aufgabenstellung variieren, in den meisten Fällen kann man diese Aufgaben mit Hilfe eines bestimmten Verfahrens lösen. Werden die gleichen Typen zu häufig gestellt, dann werden für diese ebenfalls Algorithmen erstellt, mit denen man wieder Gefahr läuft, dass das Verständnis auf der Strecke bleibt.

Es bleibt demzufolge festzuhalten, dass die automatische Aufgabenerstellung nicht dazu gedacht ist, dass Lernende möglichst viele Aufgaben des gleichen Typs lösen. Auf diese Weise stellt man nicht unbedingt sicher, dass die Lernenden wirklich die Inhalte üben und im Anschluss transferieren können. Als Lernender kann man sich nämlich schnell die Frage stellen, welche Anforderungen denn nun gestellt werden. Soll man nur einen Rechenalgorithmus anwenden können oder Verständnis des Inhalts entwickeln? Um eine solche Verwirrung gar nicht erst aufkommen zu lassen, sollte man als Lehrender gleich davon Abstand nehmen, das Üben so anzulegen, dass diese Unklarheit über die Anforderung entstehen kann. Das Ziel der Problemlösefähigkeit kann sonst aufgrund von mangelnden Voraussetzungen verfehlt werden.
Ein sinnvoller Einsatz von automatischer Aufgabenerstellung geschieht viel eher dadurch, dass man in die Lage versetzt wird, allen Mitgliedern einer Lerngruppe unterschiedliche Aufgaben zu geben, sodass die Einzelaktivität der Lernenden einer Lerngruppe höher ausfällt. Sie müssen ihre eigenen Aufgaben bearbeiten und können daher von der Lösung anderer weniger profitieren. Selbst wenn sie von anderen abschreiben, müssen sie immer noch die Lösungsschritte von der vorliegenden Aufgabe auf ihre Aufgabe übertragen, wodurch ein größerer Lerneffekt entsteht als beim Abschreiben ohne einen solchen Transfer. An dieser Stelle setzt OKUSON an, indem es die Aufgabenblätter individualisiert.

Die Frage ist nun, wie oft ein Aufgabentyp wiederholt werden soll. Meiner Meinung nach hängt das von der konkreten Situation ab. Die Frage muss lauten, was eigentlich erreicht werden soll. Angenommen, ein Lernender scheitert immer an der gleichen Stelle, sodass ausgeschlossen ist, dass es nur eine einzige Unaufmerksamkeit war, dann muss die Frage lauten, ob diese Stelle für das Verständnis und damit für die später zu erbringende Transferleistung wichtig ist. Sollte sie es sein, gibt es mit Sicherheit auch andere Aufgaben, bei denen die gleiche Schwierigkeit auftritt, sodass man hier in der Lage ist,

zu variieren, und keine Unklarheit über das Ziel aufkommen kann. Sollte sie es nicht sein, handelt es sich bei dieser Schwierigkeit meistens um einen Bereich, der zu einem anderen Themengebiet gehört. An dieser Stelle würde ich dem Lernenden raten, sich noch einmal selber mit diesem Bereich auseinanderzusetzen, weil der Transfer dieses Inhalts Voraussetzung für die weitere Arbeit ist. Je nach erwarteter Eigenständigkeit sollte der Lernende dann in beiden Fällen von der Lehrperson weitergehend unterstützt werden.
Es wird also immer eine Analyse der Fehler vorausgesetzt, um das weitere Vorgehen festzulegen. Doch gerade wenn man Aufgaben automatisch kontrollieren lässt, kann man eine solche Analyse eigentlich kaum leisten. Unter der Bedingung, dass es typische Fehler gibt und man zwischen den richtigen Lösungen und den Lösungen mit diesen typischen Fehlern wählen lässt, ist eine solche Analyse zwar machbar, allerdings wird das Erzeugen solcher fehlerhaften Lösungen abhängig von der Komplexität der Aufgaben immer schwieriger. Damit Schwierigkeiten erkannt werden können, muss es meiner Meinung nach je nach Eigenständigkeit der Lernenden neben Aufgaben mit automatischer Kontrolle auch Aufgaben mit manueller Kontrolle geben. Eine Alternative bei hoher Eigenständigkeit wäre eine Art Reflektionsmöglichkeit, wie man sie zum Beispiel über die Auswahl von Lösungen bei Mehrfachaufgaben[17] geben könnte, wobei man die falschen Lösungen nicht mit Absicht durch typische Fehler entstehen lässt, sodass die Lernenden mit Schwierigkeiten sich die Frage stellen, warum ihre Lösung nicht zur Auswahl steht.

1.1.3 Formen des Übens mathematischer Sachverhalte

Zech [Zech, 2002, Seite 208 ff.] unterscheidet zwischen den folgenden Übungsformen.

1. Verständnisübungen

Verständnisübungen haben zum Ziel, das begriffliche Grundverständnis zu sichern, indem der Verständniskern des zu lernenden mathematischen Sachverhaltes angesprochen wird. *„Mit »Verständniskern« eines mathematischen Sachverhalts werde seine möglichst »tief« in der kognitiven Struktur des Schülers [des Studenten] verankerbare Bedeutung bezeichnet.“* [18] Es handelt sich dabei um eine möglichst anschauliche Vorstellung, die den zu lernenden Sachverhalt mit den den Lernenden schon bekannten und von den Lernenden begreifbaren sowie schon begriffenen Sachverhalten beschreibt.[19]

[17] Zur Definition von Mehrfachaufgaben siehe Seite 17.
[18] [Zech, 2002, Seite 132], Hervorhebungen im Original
[19] Vergleiche auch [Zech, 2002, Seite 132 und 208].

2. Stabilisierendes Üben

Beim stabilisierenden Üben soll der zu lernende Sachverhalt kleinschrittig nachvollzogen werden. Es soll insbesondere dabei helfen, Standardschwierigkeiten zu beseitigen. Dafür werden ähnliche Aufgaben gestellt, deren Schwierigkeitsgrad dadurch gesteigert wird, dass nacheinander die Stellen betrachtet werden, bei denen die Standardfehler bei der Bearbeitung auftreten können. Die Schwierigkeiten der Lernenden sollen durch dieses Vorgehen isoliert bearbeitet und dadurch ausgeräumt werden.[20]

3. Operatives Üben

Das operative Üben zielt darauf ab, schon erarbeitetes Verständnis weiter zu vertiefen, indem der zu lernende Sachverhalt in variierten Aufgabenstellungen vorkommt. Die Variation findet bei den Darstellungsebenen und den für diesen Sachverhalt unwesentlichen Aspekten statt, um gegebenenfalls starre Betrachtungsweisen eines Sachverhaltes aufzubrechen, sowie zwischen den neu eingeführten und bereits bekannten Sachverhalten, um Gemeinsamkeiten und Unterschiede von diesen herauszuarbeiten.[21]

4. Automatisierendes Üben

Automatisierendes Üben hat das Lernen von Techniken und Verfahren zum Ziel, die zur Lösung von bestimmten Aufgaben zu einem Sachverhalt genutzt werden. Dabei wird dafür gesorgt, dass die Aufgaben immer den gleichen Ablauf haben. Zusätzlich kann man noch eine gewisse Struktur in der Lösung vorgeben, also zusätzliche Hinweisreize geben.[22] In der Psychologie spricht man von einem gleichen Kontext beim Enkodieren und beim Abruf. Der spätere Abruf der Informationen wird dann aufgrund der Enkodierspezifität erleichtert.

> „Enkodierspezifität | Das Prinzip, dass der spätere Abruf von Informationen verbessert wird, wenn die Hinweisreize beim Abruf mit jenen bei der Enkodierung übereinstimmen.“ [23]

Eine weitere psychologische Erklärung des Erfolgs einer solchen Übung liegt im Gesetz des Effekts.

[20] Vergleiche auch [Zech, 2002, Seite 208 f.].
[21] Vergleiche auch [Zech, 2002, Seite 114 ff., 208 f.].
[22] Vergleiche auch [Zech, 2002, Seite 173 f., 208 und 210].
[23] [Zimbardo & Gerrig, 2004, Seite 310]

> „Gesetz des Effekts (law of effect) | Ein grundlegendes Lerngesetz, dass die Kraft eines Stimulus, eine Reaktion hervorzurufen, verstärkt wird, wenn der Reaktion eine Belohnung folgt, und geschwächt wird, wenn keine Belohnung folgt.“ [24]

Positive Erlebnisse beim Lösen von gleichen Aufgaben können also zur Folge haben, dass versucht wird, die folgenden Aufgaben auf die gleiche Weise zu lösen. Eine solche Lernform nennt man Operantes Konditionieren.

> „Operantes Konditionieren | Eine Lernform, bei der sich die Wahrscheinlichkeit einer Reaktion auf Grund einer Veränderung ihrer Konsequenzen ändert.“ [25]

Allerdings ist nicht gesagt, was genau eine Verstärkung eines Verhaltens zur Folge hat. Es kann sowohl die Erfahrung sein, dass man eine Aufgabe lösen kann, weil man verstanden hat, warum man etwas tun muss, es kann aber auch die Erfahrung sein, dass man eine Aufgabe lösen kann, wenn man sich an bestimmte Regeln hält, auch wenn man den Sinn der Regeln nicht versteht.

Aufgaben für solche Übungen lassen sich sehr einfach mit Programmen generieren. Allerdings sollte man nicht vergessen, *„daß man Verständnis durch noch so viele Automatisierungsübungen nicht ersetzen kann.“* [26] Denn, wie schon erwähnt, setzen solche Algorithmen kaum Verständnis voraus, insbesondere dann nicht, wenn man die Notwendigkeit ihrer Anwendung nicht herleitet, sondern lediglich Hinweisreize ohne Beachtung der eigentlichen Aufgabe für die Anwendung eines Verfahrens sorgen. Dann ist es nämlich so, als ob man eine Begriffsreihenfolge gelernt hat, bei der die einzelnen Begriffe eigentlich nichts miteinander zu tun haben. Das Ziel war aber, genau diese Begriffe bewusst sinnvoll anzuordnen, und nicht nur die Wiedergabe in der gelernten Reihenfolge.

5. Anwendungsorientiertes Üben

> „*Anwendungsorientiertes Üben* wird hier in einem weiteren Sinne von Anwendungen gemeint: Es wird hier nicht nur an außermathematische, reale Anwendungen gedacht (...), sondern auch an innermathematische Anwendungen, insofern die weitere Anwendung des Gelernten in der (Schul-)Mathematik angestrebt wird.
> In beiden Fällen geht es um eine hinreichend breite Anwendung des Gelernten in neuen Fragestellungen und Situationen in einer Richtung,

[24] [Zimbardo & Gerrig, 2004, Seite 262]
[25] [Zimbardo & Gerrig, 2004, Seite 263]
[26] [Zech, 2002, Seite 174]

die vor allem eine Übertragung auf vorgestellte spätere Anwendungsmöglichkeiten begünstigt.“ [27]

6. Wiederholungen

„Gemeint sind [mit Wiederholen] (...) »Neuaufnahmen« von Lernprozessen nach längeren Lernabschnitten, die vor allem periodisch wiederkehrenden Zusammenfassungen dienen, dem Einordnen und Abgrenzen (...). Dementsprechend könnte man »Wiederholungen« im hier gemeinten Sinne vor allem von »Übungen« nach einer ersten Einführung einzelner Operationen und Aufgabentypen unterscheiden. Wiederholungen nehmen häufig auch Elemente anderer Übungsformen wieder auf (z.B. Verständnisaufgaben zu besonderen Schwierigkeien, die wichtigsten Routinesituationen, jetzt aber meist in »gemischter« Form; (...)).“ [28]

1.2 Lernziele und deren Kontrolle

Mein Programm soll insbesondere in Zusammenarbeit mit OKUSON eingesetzt werden. OKUSON hilft, wöchentliche individuelle Übungsblätter für Studenten zu erstellen, die diese bearbeiten sollen. Für die Bearbeitung der Aufgaben werden Punkte vergeben, wobei die Anzahl der erreichten Punkte oft über die Zulassung zur abschließenden Klausur am Ende des Semesters entscheidet und/oder eine Aufwertung der Klausurnote bedeuten kann. Es wird also zur Kontrolle des Erreichens von Lernzielen eingesetzt, indem es Aufgaben für eben diese Kontrolle generieren soll.

1.2.1 Zum Begriff des Lernziels

Definition

Die Definition des Lernzielbegriffs ist umstritten.[29] Oft wird eine sehr enge Definition für Lernziele genutzt, dort sind sie eine *„sprachlich artikulierte Vorstellung über die durch Unterricht (oder andere Lernveranstaltungen) zu bewirkende gewünschte Verhaltensänderung eines Lernenden“* [30].
Unter Lernzielen wird im Gegensatz dazu in dieser Arbeit eine *„sprachlich*

[27] [Zech, 2002, Seite 210], Hervorhebungen im Original
[28] [Zech, 2002, Seite 211]
[29] Vergleiche auch [Meyer, 2003, Seite 137 f.].
[30] [Meyer, 2003, Seite 137]

*artikulierte Vorstellung über die durch Unterricht (oder andere Lernveranstaltungen) zu bewirkende gewünschte Verhaltens*disposition *eines Lernenden*“ [31] verstanden.
Der Unterschied liegt im Detail, eine Verhaltensänderung ist beobachtbar, eine Verhaltensdisposition *„ist nicht unmittelbar beobachtbar. Mit Dispositionsangaben werden Fähigkeiten bezeichnet, Gelerntes auch in nicht eindeutig voraussehbaren Situationen „sinngemäß“ richtig zu beherrschen.*“ [32] Ich schließe mich hier der Überlegung an, dass das Erreichen von Lernzielen auch unabhängig von einer beobachtbaren Verhaltensänderung geschehen kann.

Lernzieldimensionierung

Man kann die Bereiche, auf die sich Lernziele beziehen, nach Bloom[33] auf die folgende Art unterscheiden:

> - „*kognitive Lernziele* (sie beziehen sich auf Denken, Wissen, Problemlösen, auf Kenntnisse und intellektuelle Fähigkeiten),
> - *affektive Lernziele* (sie beziehen sich auf die Veränderung von Interessenlagen, auf die Bereitschaft, etwas zu tun oder zu denken und auf die Entwicklung dauerhafter Werthaltungen),
> - *psychomotorische Lernziele* (sie beziehen sich auf die manipulativen und motorischen Fertigkeiten eines Schülers [eines Lernenden]).“ [34]

Lernzieloperationalisierung

Auch der Begriff der Lernzieloperationalisierung mehrere Bedeutungen.[35] Angenommen, man möchte einer Lerngruppe einen bestimmten Sachverhalt beibringen, dann ist es sinnvoll, diesen Sachverhalt immer weiter in kleinere Lernziele zu gliedern, die Stück für Stück vermittelt werden. Diesen Prozess nennt man Lernzieloperationalisierung.

> „*Lernzieloperationalisierung* = Kleinarbeitung einer ungenauen Lernzielangabe bis zur sprachlich möglichst eindeutigen Angabe der beobachtbaren Elemente der gewünschten Verhaltensdisposition des Lernenden.“ [36]

[31] [Meyer, 2003, Seite 138], Hervorhebungen im Original
[32] [Meyer, 2003, Seite 139]
[33] Vergleiche auch Bloom, Benjamin u.a. (1972): *Taxonomie von Lernzielen im kognitiven Bereich.* Weinheim (Beltz).
[34] [Meyer, 2003, Seite 143], Hervorhebungen im Original
[35] Vergleiche auch [Meyer, 2003, Seite 139].
[36] [Meyer, 2003, Seite 140], Hervorhebungen im Original

Dabei unterscheidet man zwischen Richtzielen, Grobzielen und Feinzielen, wobei die Lernziele in genau dieser Reihenfolge immer konkreter werden und immer weniger Alternativen zulassen, bis zum Feinziel, das so eindeutig, wie man das mit Sprache nur tun kann, formuliert sein sollte. Das Problem einer solchen Einteilung ist allerdings die Frage der Trennung dieser Bereiche. Die Auftrennung geschieht mit Hilfe der Bestimmung des Abstraktionsniveaus. Jeder Lehrende ist jedoch ein Individuum und bringt eigene Voraussetzungen mit, was wiederum heißt, dass das *„Abstraktionsniveau einer Lernzielformulierung (...) immer nur* relativ *zu anderen Zielformulierungen und abhängig vom Wissensstand des Formulierers dieses Lernziels bestimmt werden“* [37] kann. Die Bestimmung des Abstraktionsniveaus und damit die Auftrennung in Richtziele, Grobziele und Feinziele kann also nicht auf immer gleiche Weise geschehen.

Wie oben erwähnt, gibt es noch ein weiteres Verständnis des Begriffs der Lernzieloperationalisierung.

> „*Lernzieloperationalisierung* = Angabe der Meßoperation, mit der ein beobachtbares Element einer gewünschten Veränderung des Schülerverhaltens [Verhaltens des Lernenden] kontrolliert werden kann.“ [38]

In diesem Fall besteht der Prozess der Lernzieloperationalisierung darin, dass einem Lernziel ein Messverfahren zugeordnet wird, das bestimmen soll, ob das Lernziel erreicht ist. Ich gehe noch einmal in Unterabschnitt 1.2.2 genauer darauf ein.

Lernzielhierarchisierung

Lernziele können nach ihrer Operationalisierung aufgrund ihres Schwierigkeitsgrades hierarchisiert werden. Eine solche Hierarchisierung findet sich zum Beispiel im Unterabschnitt 1.2.2 auf Seite 13.

In dieser Arbeit werde ich darauf verzichten, die entwickelten Lernziele weiter als nötig zu operationalisieren und zu hierarchisieren. Dies muss Aufgabe des Anwenders des Programms sein, nicht seines Programmierers. Die Lernziele, die ich formuliere, dienen in erster Linie zur Zusammenfassung der Voraussetzungen für weitere Lernziele. Sie sind zudem weder repräsentativ noch vollständig, weil ich mich allein mit Matrizen beschäftige und den eigentlich dazugehörigen Bereich der linearen Abbildungen außer Acht lasse.[39]

[37] [Meyer, 2003, Seite 140], Hervorhebungen im Original
[38] [Meyer, 2003, Seite 141], Hervorhebungen im Original
[39] Man kann lineare Abbildungen mit Matrizen identifizieren und umgekehrt. Mein Pro-

1.2.2 Zur Entwicklung kognitiver Lernziele

Auch wenn die anderen Arten von Lernzielen aus meiner Sicht insbesondere für die Schule eine mindestens genauso hohe Bedeutung haben wie die kognitiven Lernziele, beschäftige ich mich im Weiteren ausschließlich mit diesen, weil sich die Aufgabenstellung dieser Arbeit meinem Verständnis nach nur auf die Abfrage kognitiver Lernziele bezieht.

Die kognitiven Lernziele als solche werden in Bezug auf die Lernzieltaxonomie nach Bloom noch einmal weiter in Wissen, Verstehen, Anwenden, Analyse, Synthese und Bewertung differenziert und damit auch hierarchisiert.

> 1) „»Wissen« (knowledge) heißt die Kenntnis (vor allem im psychologischen Sinne des Erinnernkönnens) von irgendwelchen Fakten oder Verfahren.
> 2) »Verstehen« (comprehension) heißt die Fähigkeit, eine mitgeteilte Information sachgerecht aufzunehmen, in eine andere Form zu übertragen, zu interpretieren oder zu verallgemeinern.
> 3) »Anwenden« (application) heißt die Fähigkeit, von allgemeinen Regeln und Verfahren in speziellen Situationen Gebrauch zu machen.
> 4) »Analyse« (analysis) heißt die Fähigkeit, eine Information in Teile zu zerlegen, so daß ihre Beziehungen zueinander bzw. ihre Organisation deutlich wird.
> 5) »Synthese« (synthesis) heißt die Fähigkeit, Teile zu einem neuen Ganzen zusammenzusetzen.
> 6) »Bewertung« (evaluation) heißt die Fähigkeit, Urteile über den Wert von Materialien und Methoden abgeben zu können.“ [40]

Angewendet auf den Satz von Pythagoras, hieße dies Folgendes:

> 1) „*Kenntnis* des Satzes von Pythagoras hieße z.B.: den Inhalt des Satzes in eigenen Worten wiedergeben zu können.
> 2) *Verstehen* des Satzes von Pythagoras hieße z.B.: den Gedankengang eines Beweises des Satzes wiedergeben zu können.
> 3) *Anwenden* des Satzes von Pythagoras hieße z.B.: in einer Sachaufgabe von ihm Gebrauch machen zu können.

gramm beschäftigt sich jedoch nur mit Matrizen, weil es mit diesen rechnen kann. Die Interpretation von Matrizen als lineare Abbildungen lasse ich außer Acht. Es würde den Rahmen dieser Arbeit sprengen.

[40] [Zech, 2002, Seite 67 f.]

4) *Analyse* hieße z.B.: einen vorgelegten Beweis des Satzes auf seine logische Stichhaltigkeit untersuchen zu können.

5) *Synthese* hieße z.B.: einen Beweis des Satzes selber finden zu können.

6) *Bewertung* hieße z.B.: verschiedene Beweise des Satzes bezüglich der benutzten Beweismittel vergleichen zu können.“ [41]

Man sieht, dass diese Einteilung einer gewissen Hierarchie in der Weise folgt, *„daß jede folgende Kategorie die vorhergehende i.a. voraussetzt.“* [42]

Nun könnte man hier behaupten, dass ich mir widerspreche, habe ich doch behauptet, Anwendung setze kein Verständnis voraus. Ein Lernziel, das sich auf die Anwendung bezieht, will ich deshalb immer so verstanden wissen, dass es sich dabei um eine bewusste Anwendung handelt. Der Unterschied zwischen einer bewussten und einer unbewussten Anwendung eines Sachverhaltes besteht für mich darin, dass derjenige, der einen Sachverhalt bewusst anwendet, auch eine Begründung aus dem Verständnis dieses Sachverhaltes geben kann, warum er ihn anwendet. Jemand, der den Sachverhalt unbewusst anwendet, kann dies nicht, er hat lediglich gelernt, dass er in einer bestimmten Situation etwas Bestimmtes tun muss.

Man kann die obige Einteilung noch weiter verfeinern, um das ganze Spektrum der unterschiedlichen Charaktere der Lernziele darzustellen, wie es auch Zech [Zech, 2002, Seite 69 ff.] tut. Hier reicht aus meiner Sicht die dargestellte Einteilung aus, ein erstes Lernziel zu formulieren.

Lernziel 1.1

Die Studenten sollen den Rang einer Matrix bestimmen können.[43]

Das bedeutet nach meinen bisherigen Ausführungen, dass sie um ein Lösungsverfahren wissen und es anwenden können, es bedeutet nicht unbedingt, dass sie das Verfahren auch verstehen. Also ergänze ich dieses Lernziel um das Verständnis des Verfahrens im oben genannten Sinn.

Lernziel 1.2

Die Studenten sollen die Bestimmung des Rangs einer beliebigen Matrix beherrschen.

[41] [Zech, 2002, Seite 68], Hervorhebungen im Original
[42] [Zech, 2002, Seite 68]
[43] Zum Begriff „Rang“ siehe Definition 3.23.

Wenn man das Erreichen eines Lernziels überprüfen möchte, muss man unter anderem auch Kriterien entwickeln, unter welchen Bedingungen es erreicht sein soll. Das Lernziel muss deshalb bezüglich der folgenden Richtungen noch weiter präzisiert werden.

1. „Die Beschreibung des Lernziels als *Endverhalten* des Lernenden;
2. die *eindeutige Beschreibung* des Endverhaltens;
3. die genauere Angabe der *Voraussetzungen und Bedingungen* des Endverhaltens;
4. die *Angabe eines Beurteilungsmaßstabes* für die Güte des Endverhaltens.“ [44]

Nun ist Sprache nicht immer so eindeutig wie mathematische Formeln, sodass nicht immer klar ist, ob alle vier dieser Bedingungen erfüllt sind. Lernziel 1.1 beschreibt das Endverhalten des Studierenden nicht eindeutig. Soll er nur das Verfahren anwenden können oder soll er auch wissen, warum es funktioniert? Lernziel 1.2 erfüllt auch nur die ersten beiden Bedingungen. Das Endverhalten wird hier dadurch beschrieben, dass ein Verfahren beherrscht werden soll. Damit ist auch klar, dass man es nicht nur anwenden können soll, sondern auch um die Hintergründe wissen und sie verstanden haben muss. Dieses Lernziel ist in Bezug auf das Endverhalten also eindeutig, wenn man sich nicht am Begriff des Beherrschens aufhängt. Dieser kann wie in Sprache üblich von jedem anders definiert werden, sodass die geforderte Eindeutigkeit eigentlich nicht mehr vorhanden ist, auch wenn es zwischen den jeweiligen individuellen Deutungen eine große Schnittmenge gibt, auf die man zurückgreifen kann und dies auch tun sollte, damit Lernziele möglichst eindeutig sind. Aus diesem Blickwinkel heraus ist die Eindeutigkeit ohne eine klare Definition von „beherrschen“ nicht gegeben. Ich verstehe den Begriff „beherrschen“ nun auf die Weise, dass man ein Thema genau dann beherrscht, wenn man über alles, was man beherrschen können soll, genau Auskunft geben, es auch anwenden und auf andere Bereiche übertragen kann.
Erst wenn das erreicht wäre, wäre dann auch das Lernziel erreicht, und man könnte eine solche Definition als Bedingung für das Endverhalten gelten lassen. Doch gerade wenn eine im Allgemeinen heterogene Lerngruppe ein Lernziel erreichen soll, ist eine solche Definition eher utopisch und nicht mit der Realität vereinbar, weil der Lernfortschritt der ganzen Gruppe dann gefährdet wird, wenn man versucht, dass alle Mitglieder einer solchen Gruppe dieses Ziel erreichen. Einige werden mehr, andere weniger Zeit brauchen, weshalb eine flexiblere Handhabung bei der Beurteilung, wann ein Lernziel erreicht

[44] [Zech, 2002, Seite 81], Hervorhebungen im Original

wird, notwendig ist. Weiterhin ist niemand perfekt; eine solche Definition, die eine immer perfekte Bearbeitung einfordert, ignoriert also, dass Menschen Fehler machen. Die obige Formulierung würde heißen, dass man durch einen simplen Rechenfehler oder das Vergessen eines einzigen Vorzeichens an entscheidender Stelle, das Lernziel nicht erreicht hätte, wenn man „beherrschen“ so scharf definiert. Es ist also klar, dass man weiter ergänzen muss, um eventuelle Flüchtigkeitsfehler, die nicht aufgrund von mangelndem Verständnis passiert sind, zu berücksichtigen.

Lernziel 1.3

Die Studenten sollen die Bestimmung des Rangs einer beliebigen Matrix beherrschen. Das Lernziel gilt als erreicht, wenn die Stundenten auf ihrem wöchentlichen Übungsblatt 6 von 8 Aufgaben richtig lösen können.

Die Angabe der genaueren Bedingungen des Lernverhaltens wird in diesem Fall durch den Zusatz „beliebig“ erfüllt. Alternativ könnte man hier schreiben, dass die Matrizen zum Beispiel nur Einträge aus $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ oder $R[X]$, wobei R ein Ring ist, haben dürfen.[45]

Das Lernziel sieht am Ende also folgendermaßen aus.

Lernziel 1.4

Die Studenten sollen die Bestimmung des Rangs einer beliebigen Matrix mit Einträgen aus $\mathbb{Q}[X]$ beherrschen. Das Lernziel gilt als erreicht, wenn die Studenten auf ihrem wöchentlichen Übungsblatt 6 von 8 Aufgaben richtig lösen.

Über den Sinn eines solchen Lernziels lässt sich sicherlich streiten. Hier dient es nur der Veranschaulichung und hat mit realen Anforderungen eher weniger zu tun. Ich werde später in dieser Arbeit Lernziele zu bestimmten Themen aufstellen. Dazu werde ich erst eine Art Richtziel formulieren, aus dem sich dann Grobziele und etwas „feinere“ Grobziele, aber keine Feinziele ableiten. Ich werde die Lernziele auch nicht im Hinblick auf die Angabe eines Messverfahrens operationalisieren, da ich keine geeigneten Maßstäbe für eine solche Messung entwickelt habe. Deshalb wird der vierte Punkt der obigen Auflistung später außer Acht gelassen.
Ebenso lasse ich dann auch den dritten Punkt bei der Entwicklung der Lernziele in gewisser Hinsicht außer Acht. Allerdings sei hier vermerkt, dass mein Programm im Allgemeinen nur mit ganzen Zahlen arbeiten kann, sodass die realisierbaren Aufgaben, nicht die Lernziele, auf diese Weise eingeschränkt sind.

[45] Zur Definition von Ringen siehe Abschnitt 4.1.

> „Eine derartige Präzisierung eines Lernziels als kontrollierbare Verhaltensweise des Lernenden wird als »Lernzieloperationalisierung« bezeichnet. (Man spricht gelegentlich von einer »Lernzieloperationalisierung im engeren Sinne«, wenn alle vier Bedingungen (...) erfüllt sind, und von einer »Lernzieloperationalisierung im weiteren Sinne«, wenn nur die ersten beiden Bedingungen erfüllt sind (...).)“ [46]

Also werde ich unter diesem Gesichtspunkt eine Lernzieloperationalisierung im weiteren Sinne betreiben, wenn ich in Kapitel 3 Lernziele formuliere.

1.2.3 Kontrolle mathematischer Lernziele

> „Lernzieloperationalisierung und Lernzielkontrolle hängen eng zusammen: Wird die Lernzieloperationalisierung zu Ende geführt, hat man unmittelbar Angaben darüber, an welchem Schülerverhalten [Verhalten] unter welchen gegebenen Bedingungen das Lernziel als erreicht gelten kann. Der letzte Schritt zur Lernzielkontrolle besteht jetzt »nur« noch darin, diese Angaben in lernzielorientierte Testaufgaben (...) umzusetzen.“ [47]

Wenn der Sinn solcher Aufgaben also darin besteht, ein Lernziel abzufragen, dann sollte dies auch ihr einziger Inhalt sein. Weil sie am leichtesten auszuwerten sind, sind Mehrfachaufgaben, Zusatzaufgaben und Ergänzungsaufgaben die Aufgabenvarianten, die bei einer solchen vielleicht auch kurzfristigen Abfrage eventuell im Unterricht, in einer Vorlesung oder einer Übung am ehesten in Frage kommen.

Bei **Mehrfachaufgaben** stehen zu einer Aufgabe mehrere fertige Lösungen zur Verfügung, von denen mindestens eine richtig ist. Die richtigen Lösungen sollen angegeben werden. Dabei kann man auch typische Fehler in die falschen Lösungen integrieren.[48]

Zuordnungsaufgaben geben ebenfalls mehrere Lösungen vor, allerdings auch mehrere Aufgaben. Hier sollen die Lösungen den Aufgaben zugeordnet werden.

Ergänzungsaufgaben bestehen aus einer Art Lückentext, vielleicht auch nur einem einzigen Satz, dessen Lücken ergänzt werden müssen.[49]

[46] [Zech, 2002, Seite 81]
[47] [Zech, 2002, Seite 84]
[48] Man bezeichnet diese Aufgaben auch als Multiple-Choice-Aufgaben.
[49] Vergleiche auch [Zech, 2002, Seite 84].

Für alle diese Aufgabenvarianten gilt allerdings immer noch, dass die Aufgabenstellung, wie in Tabelle 1.1 dargestellt, variiert werden sollte, um der Gefahr zu entgehen, dass die Lernenden einen Algorithmus lernen, anstatt in die Lage versetzt werden, das zu Lernende transferieren zu können.

Kapitel 2

Grenzen und Forderungen

Mit einer kurzen Darstellung des Übens und der Lernziele habe ich die Voraussetzungen für die Entwicklung der Aufgabenstellungen geschaffen. Leider ist es utopisch anzunehmen, dass ein Programm in der Lage wäre, alle Optionen für Aufgaben zu bieten, die man sich zu auch nur zu einem Thema denken könnte. Ein Programm hat Grenzen und diese hängen vor allen Dingen von der Zeit ab, die der Programmierer in das Programm investieren kann. Je mehr Zeit man als Programmierer hat, umso mehr Optionen kann man realisieren. Doch die zeitliche Begrenzung ist nicht die einzige.

2.1 In der Aufgabenstellung begründete Anforderungen

Eigentlich gehört dieses nur auf eine gewisse Weise in den Bereich der programmiertechnischen Grenzen. Das Thema dieser Arbeit lautet „Automatische Erzeugung von Aufgaben mit ganzzahligen Lösungen“. Die wohl wichtigste Anforderung an mein Programm ist deshalb die, dass es invertierbare finden können muss, die nur ganze Zahlen als Einträge haben, wobei dies auch auf die Inverse einer solchen Matrix zutreffen muss, damit gewährleistet ist, dass Lösungen und eventuell wichtige Zwischenlösungen immer noch ganzzahlig sind.

Beispiel 2.1
Man betrachte die folgende Matrix:

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

Die Inverse zu A ist

$$A^{-1} = \frac{1}{2} \cdot \begin{pmatrix} -1 & 3 \\ -2 & 4 \end{pmatrix}.$$

Wenn man nun eine dritte Matrix B, die nur ganzzahlige Einträge hat, von links mit A und von rechts mit A^{-1} multiplizieren möchte, dann kann man nicht mehr garantieren, dass das Produkt nur ganzzahlige Einträge hat. Es könnte zum Beispiel

$$B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

gelten, dann wäre

$$A \circ B \circ A^{-1} = \frac{1}{2} \cdot \begin{pmatrix} -3 & 7 \\ -3 & 7 \end{pmatrix}.$$

Mein Programm muss also einen Algorithmus beinhalten, der es erlaubt alle Paare von zueinander inversen Matrizen zu finden, die nur ganzzahlige Einträge besitzen.

2.2 Grenzen bei der Aufgabenerstellung

In Tabelle 1.1 waren die verschiedenen Aufgabentypen dargestellt. An sich sind alle dort dargestellten Aufgabentypen programmiertechnisch zu erzeugen. Probleme gibt es lediglich bezüglich der Eindeutigkeit der Lösungen in der Hinsicht, dass ein Programm vielleicht eine Lösung finden kann, aber nicht in der Lage ist, alle zu finden. Damit werde ich mich aber erst im folgenden Unterabschnitt genauer beschäftigen. Außerdem ist zu fragen, was eigentlich eine Standardaufgabe ist, da sich die Einteilung in der besagten Tabelle genau darauf bezieht. Auch wenn ich der dargestellten Argumentation durchaus zustimme, muss ich auch sagen, dass das Variieren der Aufgabentypen schwieriger wird, je komplexer die Aufgaben sind.

Für die dargestellten Aufgabenvarianten gilt Folgendes: Es ist im Allgemeinen schwieriger, sinnvolle falsche Lösungen zu erzeugen als richtige Lösungen, denn man muss garantieren, dass die falschen Lösungen, die man erzeugt, nicht doch richtig sind, insbesondere dann, wenn die richtige Lösung nicht eindeutig ist. Die Schwierigkeit bei der Erstellung von Mehrfachaufgaben besteht also darin, falsche Lösungen zu bauen, was aber realisierbar ist. Die beiden anderen vorgestellten Aufgabenvarianten habe ich nicht realisiert. Ergänzungsaufgaben haben im Allgemeinen keine ganzzahligen Lösungen, weil sie eigentlich Lückentexte sind, Zuordnungsaufgaben können leicht aus mehreren Mehrfachaufgaben manuell gebaut werden, indem man nur die richtigen

Lösungen verwendet. Es verbleibt eine Aufgabenstellung, die ich bisher etwas vernachlässigt habe, die sogenannte Standardaufgabe, bei der etwas berechnet werden soll, wobei die Lösung der Aufgabe dann über ein bestimmtes Verfahren läuft. Standardaufgaben lassen sich einfach durch mein Programm generieren.

Bei der Aufgabenstellung kann noch ein weiteres Problem auftreten. Angenommen man findet einen Algorithmus, mit dem man Aufgaben generieren kann, dann muss sichergestellt werden, dass dieser Algorithmus unter gewissen Einschränkungen wie der, dass die Aufgaben nur ganzzahlige Lösungen haben sollen, trotzdem alle möglichen Aufgaben des gleichen Typs liefern kann, da es ansonsten Verfahren geben könnte, mit denen man, ohne die Aufgaben auf die eigentlich beabsichtigte Art und Weise zu lösen, die Lösungen gewinnen kann.

Weitere Dinge, die bedacht werden sollten, sind die folgenden. Ein Computerprogramm spult ohne Nachfrage einen Algorithmus ab. Die erstellten Aufgaben müssen nicht unbedingt sinnvoll sein, weil die beteiligten Zahlen einfach zu groß oder die Aufgaben trivial sind. Ein einfaches Übernehmen der erstellten Aufgaben ist daher wenig sinnvoll. Man kann zwar ein einfaches Kriterium entwickeln, das alle Aufgaben aussortiert, die eine Zahl größer oder kleiner als bestimmte Werte beinhalten, doch ist es schwierig, dies auch für triviale Aufgaben zu tun, sodass man diese besser selber aussortiert.

2.3 Grenzen bezüglich der Eindeutigkeit von Lösungen

Das Problem einiger Aufgaben liegt, wie schon dargestellt, in der Eindeutigkeit der Lösung. Angenommen, die Lösung ist ein m-dimensionaler Untervektorraum des $\mathbb{Q}^n$, dann ist es unmöglich alle Lösungen anzugeben, auch wenn man die Angabe auf Elemente aus $\mathbb{Z}^n$ einschränkt. Man könnte nun zwar noch weiter fordern, dass der größte gemeinsame Teiler aller Einträge in diesen Vektoren nur 1 sein darf, aber auch das hilft nur in dem Fall, dass der Untervektorraum eindimensional ist und reduziert die Anzahl der Lösungen auf zwei. Ist er schon zweidimensional, dann hat man im Allgemeinen immer noch unendlich viele Varianten.
Dieses Problem taucht auch bei ähnlichen Matrizen auf. Angenommen man soll zeigen, dass zwei Matrizen $A, B \in M(2 \times 2; \mathbb{Z})$ ähnlich zueinander sind, indem man eine Matrix $P \in GL(2; \mathbb{Z})$ angibt, sodass $B = P \circ A \circ P^{-1}$ gilt. Die Frage muss lauten, ob P eindeutig ist.

Beispiel 2.2
P ist nicht eindeutig. Nehmen wir zum Beispiel

$$A := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix},$$

setzen P^{-1} aus dem Eigenvektor von A und $\binom{x}{y}$ zusammen, also

$$P^{-1} := \begin{pmatrix} 1 & x \\ 0 & y \end{pmatrix}.$$

Weil $P \in GL(n;\mathbb{Z})$ gilt, folgt unverzüglich $y \in \{\pm 1\}$. Damit ist

$$P = \begin{pmatrix} 1 & \mp x \\ 0 & \pm 1 \end{pmatrix} \text{ und } B = \begin{pmatrix} 1 & \pm 1 \\ 0 & 1 \end{pmatrix}.$$

Man sieht schnell, dass man $x \in \mathbb{Z}$ beliebig wählen kann, ohne dass die Invertierbarkeit von P oder die Matrix B beeinflusst wird. Es existieren also auch hier unendlich viele Lösungen für P.

Man kann jede Matrix auch als einen Vektor betrachten; im obigen Beispiel ist P ein vierkomponentiger Vektor, alle Lösungen für P entstammen einem affinen Unterraum.[1]

Was in diesem Fall bleibt, ist, dass solche Aufgaben sich nur dadurch automatisch kontrollieren lassen, dass man zur Kontrolle Rechenoperationen durchführen muss statt eines Abgleichs mit allen Lösungen.[2]

2.4 Forderungen an das Programm

Nach den ersten allgemeinen Betrachtungen ergeben sich die ersten Anforderungen an mein Programm, denen es gerecht werden muss.

Im Hinblick auf die automatische Erstellung von Aufgaben und deren Anwendung im Übungsprozess weise ich darauf hin, dass das Programm zwar Aufgaben liefern, aber keinerlei Einfluss auf die Gestaltung einer Übungsphase durch den Lehrenden oder den Lernenden nehmen kann. Es ist deshalb umso wichtiger, dass es die Optionen eines solchen Programms zulassen,

[1] Zum Begriff „affiner Unterraum“ siehe [Wille, 2001, Seite 45].

[2] Dies ist mit OKUSON nicht machbar. Also sind solche Aufgaben für die Anwendung mit OKUSON unbrauchbar. Schön wäre es, wenn OKUSON um eine Routine zu erweitert werden würde, die solche Rechenoperationen durchführen kann.

dass die verschiedenen Arten des Übens unterstützt werden. Mein Programm sollte also Aufgaben liefern, die sich in möglichst vielen der oben genannten Übungsformen vielfältig einsetzen lassen.
Gleichzeitig sollten die Aufgaben trotz der automatischen Kontrolle Rückschlüsse auf eigene Fehler zulassen. Dazu sind besonders Mehrfachaufgaben geeignet, bei denen die vorgegebenen falschen Lösungen entweder auf Standardfehlern basieren, um somit einen Hinweis bezüglich der Fehlerquelle zu haben, oder Standardfehler vermeiden, um die Lernenden zur Reflektion zu zwingen.

Die Aufgaben, die erstellt werden, sollten sich dazu eignen, das Erreichen von Lernzielen zu überprüfen. Also muss es auch ein Ziel dieser Arbeit sein, zumindest das Themengebiet der Aufgaben mit Grobzielen so weit zu gliedern, dass man aus diesen Grobzielen Aufgaben ableiten kann.

Eine wichtige weitere Forderung an die Aufgaben ist es, dass sie mit einem möglichst einfachen Algorithmus erstellt werden sollten, der aber trotzdem in der Lage ist, auf flexible Art und Weise eingesetzt zu werden, denn es kann nicht das Ziel sein, für jede Aufgabe ein komplett eigenes Programm schreiben zu müssen. Außerdem sollte ein Algorithmus im Programm wenn schon nicht alle auftretenden Varianten einer Aufgabe, dann doch möglichst viele dieser Varianten liefern können.

Das Problem ist weiterhin die Eindeutigkeit der Lösungen, wenn diese eigentlich Vektorräume oder affine Unterräume sind. Selbst wenn Startwerte und Lösung als Vektoren oder Matrizen nur Einträge aus $\mathbb{Z}$ besitzen, gibt es selten endlich viele Lösungen. Zu jeder Aufgabenstellung, bei der die Lösung uneindeutig sein kann, muss also ein Algorithmus vorgestellt werden, mit dem nachprüfbar ist, ob die errechnete Lösung auch richtig ist.

Es bleibt die letzte Forderung, die am meisten Arbeit bedarf. Mein Programm muss Matrizen finden, die nur ganzzahlige Einträge haben, die invertierbar sind und deren Inverse ebenfalls nur ganzzahlige Einträge besitzen. Zusätzlich soll es genau diese Inversen berechnen und auch alle möglichen Matrizen dieser Art finden können.

Kapitel 3

Die Aufgabengenerierung

Ziel meines Programms soll es sein, das Üben von Aufgaben zu Teilgebieten der Linearen Algebra zu ermöglichen, indem es verschiedene Aufgaben generiert, die in verschiedenen Übungsformen eingesetzt werden können. Mir wurde für diese Arbeit der Bereich der ähnlichen Matrizen vorgegeben.
Die erste Hauptaufgabe dieses Kapitels ist es also, die theoretischen Hintergründe aufzuarbeiten. Ich werde deshalb Definitionen und wichtige Eigenschaften kurz vorstellen, um zu zeigen, wie einige notwendige Lernziele dazu aussehen könnten. Erst nachdem ich auf diese Weise Zusammenhänge und Voraussetzungen dargestellt habe, entwickle ich aus diesen Aufgabenstellungen. Im Prinzip lassen sich noch weitere Aufgabenstellungen unter der Verwendung aller Optionen meines Programms finden als die, die ich vorstelle. Ich beschränke mich hier jedoch auf solche, die mir wesentlich erschienen - es ist also eine subjektive Auswahl, die deshalb weder repräsentativ noch vollständig ist - und auf solche, die sich mit einem möglichst immer gleichen Algorithmus erzeugen lassen, bei dem man nur wenige Schritte verändern muss, um die Aufgaben zu verändern.

Ein kleiner Abschnitt am Ende dieses Kapitels bleibt zusätzlich dem Bereich der linearen Gleichungssysteme vorbehalten, um zu zeigen, in welcher Weise man das Programm noch einsetzen kann.

3.1 Ähnliche Matrizen

3.1.1 Definition und Eigenschaften von ähnlichen Matrizen

Matrizen stehen stellvertretend für lineare Abbildungen zwischen Vektorräumen. Diesen Sachverhalt spare ich in dieser Arbeit aber weitgehend aus.

Dabei ist es mitnichten so, dass ich ihn unwichtig finde, mein Programm arbeitet allerdings mit Matrizen und nicht mit linearen Abbildungen. Tatsächlich ist es so, dass man mit Hilfe der Matrizen eigentlich Aussagen über lineare Abbildungen machen möchte. Matrizen als Darstellungen solcher Abbildungen sind ein Hilfsmittel, um Aussagen über lineare Abbildungen treffen zu können.
Auch die Lernziele, die ich vorstelle, lassen eine solche Betrachtung vermissen. Sie sind deshalb auch nicht vollständig und daher mit Vorsicht zu genießen. In dieser Arbeit haben Sie lediglich die Funktion, die notwendigen aufeinander aufbauenden Schritte in Bezug auf Matrizen zusammenzufassen. Trotzdem halte ich sie für wichtig, denn wenn man sich einem solchen Themenbereich nähert, um Aufgaben dazu zu stellen, dann muss man wissen, welche Dinge aufeinander aufbauen und wie die Zusammenhänge sind. Die Formulierung von Lernzielen erlaubt es, ein solches Themengebiet auf diese Weise zu gliedern.

Definition 3.1

Zwei Matrizen $A, B \in M(n \times n; K)$ sind ähnlich über dem Körper K, wenn es eine Matrix $P \in GL(n; K)$ gibt, sodass $B = P \circ A \circ P^{-1}$ gilt.[1]
P heißt dann konjugierende Matrix.

Diese Definition gilt natürlich nur über einem Körper K. Die ganzen Zahlen, in denen nach Aufgabenstellung die Lösungen liegen sollen, bilden lediglich einen Integritätsring[2], $\mathbb{Z}$. Allerdings gilt diese Definition im Quotientenkörper[3] von $\mathbb{Z}$, nämlich in $\mathbb{Q}$. Damit gibt es Matrizen aus $M(m \times n; \mathbb{Q})$ mit Einträgen, die nur in $\mathbb{Z}$ liegen. Mit Hilfe der Elementarmatrizen[4] von $\mathbb{Z}$ kann man sogar über $\mathbb{Z}$ invertierbare Matrizen zusammenbauen.[5] Man kann mit einer Matrix starten, die nur Einträge aus $\mathbb{Z}$ hat, sie von links mit einer über $\mathbb{Z}$ invertierbaren Matrix und von rechts mit der dazugehörigen Inversen multiplizieren, dann erhält man eine zur Ausgangsmatrix ähnliche Matrix, die nur Einträge aus $\mathbb{Z}$ hat. Demnach ist die folgende Definition sinnvoll.

Definition 3.2

Zwei Matrizen $A, B \in M(n \times n; R)$, R ein Ring, sind ähnlich über R, wenn es eine Matrix $P \in GL(n; R)$ gibt, sodass $B = P \circ A \circ P^{-1}$ gilt.[6]

[1] Zur Definition von „Körper“ und der Mengen $M(n \times n; K)$, $GL(n; K)$ siehe Abschnitt 4.1.
[2] Zur Definition von „Integritätsring“ siehe Abschnitt 4.1.
[3] Zur Definition von „Quotientenkörper“ siehe Unterabschnitt 4.4.2.
[4] Zur Definition von „Elementarmatrizen über $\mathbb{Z}$“ siehe Bemerkung 4.51.
[5] Zur Definition von „über einem Ring R invertierbare Matrix“ siehe Abschnitt 4.1.
[6] Zur Definition von R und der Mengen $M(n \times n; R)$, $GL(n; R)$ siehe Abschnitt 4.1.

Im Allgemeinen möchte man jedoch die Ähnlichkeit von Matrizen über Körpern beschreiben, diese Definition ist also eher in Bezug auf mein Programm wichtig, da dieses nicht mit Brüchen arbeiten soll. Erwähnenswert ist noch, dass durch die Definition der Ähnlichkeit über $\mathbb{Z}$ nicht automatisch gesagt ist, dass auch alle Zwischenergebnisse bei den Rechnungen in $\mathbb{Z}$ liegen. Das Gegenteil ist häufig der Fall, wie unter anderem auch das spätere Beispiel 3.20 zeigt. Je nach Rechnung können sie in $\mathbb{Q}$ oder gar in $\mathbb{R}$ bzw. $\mathbb{C}$ liegen.

Die nach Holz und Wille [Holz & Wille, 2002, Seite 60] wichtigen Sätze zur Ähnlichkeit sind die folgenden:

Satz 3.3
Seien $A, B \in M(n \times n; K)$, dann gilt:

(i) Die folgenden Aussagen sind äquivalent:

(a) A ist ähnlich zu B über K.

(b) Es gibt eine über K invertierbare Matrix P mit $B = P \circ A \circ P^{-1}$.

(c) Es gibt eine Basis C des K^n, sodass $A = M_C^C(\varphi_B)$.

(ii) Ähnliche Matrizen haben das gleiche charakteristische Polynom p und damit auch die gleichen Eigenwerte.
Sind A, B ähnlich, dann gilt also $p_A = p_B$.

(iii) Ähnliche Matrizen haben das gleiche Minimalpolynom m.
Sind A, B ähnlich, dann gilt also $m_A = m_B$.

(iv) Sind A, B diagonalisierbar, so ist A genau dann ähnlich zu B, wenn $p_A = p_B$ gilt.

(v) Zerfällt p_A über K in Linearfaktoren, so ist A über K ähnlich zu einer Matrix in Jordanscher Normalform.

(vi) Zerfallen p_A, p_B über K in Linearfaktoren, so sind A und B genau dann ähnlich über K, wenn sie bis auf die Reihenfolge der Jordankästchen, die gleiche Jordansche Normalform besitzen.

(vii) Sei L ein Oberkörper von K. Dann sind A, B genau dann ähnlich über K, wenn sie ähnlich über L sind.[7]

[7] Zur Definition von „Oberkörper“ siehe [Bosch, 2004, Seite 88].

(viii) *Ist L der algebraische Abschluss von K oder zerfallen p_A, p_B über L, einem Oberkörper von K, in Linearfaktoren, dann sind A, B genau dann ähnlich über L, wenn sie in L die gleiche Jordansche Normalform haben.*[8]

[9]

Beweis:

(i) Ein Beweis findet sich in [Holz & Wille, 2002, Seite 63].

(ii) Ein Beweis findet sich in [Holz & Wille, 2002, Seite 93 f.].

(iii) Ein Beweis findet sich in [Holz & Wille, 2002, Seite 98].

(iv) Diese Eigenschaft folgt direkt aus *(viii)*.

(v) Ein Beweis findet sich in [Fischer, 2002, Seite 259 ff.].

(vi) Beweisskizze: Aus *(v)* weiß man, dass A, B ähnlich zu den Jordanschen Normalformen J_A, J_B sind. Das Vertauschen von Zeilen bzw. Spalten geschieht mit einer Matrix aus $GL(n; K)$, die zu sich selbst invers ist.[10] Damit Blöcke vertauscht werden können müssen in der richtigen Reihenfolge Zeilen und Spalten getauscht werden. Das Produkt dieser Vertauschungsmatrizen für die Zeilenvertauschungen, die nötig sind, um J_A zu J_B umzuformen, sei nun P, dann gilt:

$$A = P_A \circ J_A \circ {P_A}^{-1} \text{ mit } P_A \in GL(n; K)$$
$$B = P_B \circ J_B \circ {P_B}^{-1} \text{ mit } P_B \in GL(n; K)$$
$$J_B = P \circ J_A \circ P^{-1}$$

$$\begin{aligned} \Rightarrow B &= P_B \circ P \circ J_A \circ P^{-1} \circ {P_B}^{-1} \\ &= P_B \circ P \circ {P_A}^{-1} \circ A \circ P_A \circ P^{-1} \circ {P_B}^{-1} \\ &= P_B \circ P \circ {P_A}^{-1} \circ A \circ \left(P_B \circ P \circ {P_A}^{-1}\right)^{-1} \end{aligned}$$

Also sind A, B ähnlich über K.

(vii) Ein Beweis findet sich in [Reineke, 2000, Seite 31].

(viii) Diese Eigenschaft folgt aus *(v)* und *(vi)*.

[8] Zur Definition von „algebraischer Abschluss“ siehe [Bosch, 2004, Seite 95].
[9] Vergleiche auch [Holz & Wille, 2002, Seite 61].
[10] Siehe auch Unterabschnitt 4.2.1.

11

Für symmetrische Matrizen gelten noch weitere Eigenschaften, die ich hier aber weglasse, weil ich keinerlei Optionen in mein Programm integriert habe, die explizit mit symmetrischen Matrizen arbeiten.

Will man diese Eigenschaften verstehen, dann muss man zuvor auch alle Voraussetzungen verstanden haben. Angenommen, man definiert unter den Voraussetzungen, dass die Stundenten schon den Bereich der linearen Abbildungen abgeschlossen haben, das folgende Lernziel,

Lernziel 3.1

Die Studenten sollen den Beweis der Ähnlichkeit von zwei Matrizen beherrschen.

dann sind zum Erreichen dieses Zieles sind mehrere Zwischenschritte nötig, von denen ich einige im Folgenden darstellen möchte.
Es ist unter anderem wichtig, um die Begriffe charakteristisches Polynom und Jordansche Normalform zu wissen, sie bestimmen zu können, ihre Eigenschaften zu beherrschen und diese in verschiedenen Aufgaben anwenden zu können. Das heißt, dass man sofort Lernziele angeben kann, deren Erreichen Voraussetzung für das Erreichen von Lernziel 3.1 ist.

Lernziel 3.2

Die Studenten sollen den Begriff des charakteristischen Polynoms kennen, um seine Eigenschaften (vor allem in Bezug auf die Ähnlichkeit von zwei Matrizen) wissen, es zu einer gegebenen Matrix bestimmen und seine Eigenschaften in verschiedenen Aufgaben ausnutzen können.

Lernziel 3.3

Die Studenten sollen den Begriff der Jordanschen Normalform kennen, um ihre Eigenschaften (vor allem in Bezug auf die Ähnlichkeit von zwei Matrizen) wissen, sie zu einer gegebenen Matrix bestimmen und ihre Eigenschaften in verschiedenen Aufgaben ausnutzen können.

Man kann an dieser Stelle sicherlich noch weitere Lernziele aufführen, deren Erreichen Voraussetzung für das Erreichen der von Lernziel 3.1 ist. Für mich fassen diese beiden jedoch die Voraussetzungen gut zusammen, zu denen

[11] Beweise, bei denen ich explizit auf eine Quelle verweise, die ich also nicht selber führe, schließe ich nicht □ ab.

mein Programm Aufgaben generieren kann.[12] Die obigen Lernziele besitzen ebenfalls Voraussetzungen, die ich im Folgenden ausarbeiten möchte, wobei das Erreichen von Lernziel 3.2 auch Voraussetzung für das Erreichen von Lernziel 3.3 ist.

3.1.2 Eigenwerte und Eigenvektoren

Definition 3.4

(i) Ist $A \in M(n \times n; K)$, K ein Körper, so heißt $\lambda \in K$ ein Eigenwert von A, falls es ein $\vec{x} \in K^n$ mit $\vec{x} \neq \vec{0}$ gibt, sodass $A \circ \vec{x} = \lambda \cdot \vec{x}$ gilt.
$\vec{x}$ heißt dann Eigenvektor von A zum Eigenwert λ.
Zusammen mit dem Nullvektor bilden die Eigenvektoren zu einem Eigenwert λ einen Untervektorraum des K^n, den Eigenraum zum Eigenwert λ, $V_\lambda(A)$.

(ii) Ist $A \in M(n \times n; K)$, K ein Körper, so heißt $p_A := \det(A - \lambda \cdot E_n)$ das charakteristische Polynom der Matrix A.[13]
A heißt zerfallend über K, wenn sein charakteristisches Polynom p_A über K in Linearfaktoren zerfällt.

[14]

Wie Beispiel 3.20 zeigen wird, ist die Anzahl der Eigenwerte einer Matrix A von dem Körper K abhängig. In jenem Beispiel hätte A über $\mathbb{Q}$ nur einen Eigenwert, über $\mathbb{C}$ jedoch drei. Eigenwerte einer Matrix müssen also nicht in dem Ring oder dem Körper liegen, über dem die Matrix definiert ist.[15]

Folgende Sätze stellen wichtige Eigenschaften von Eigenwerten, Eigenvektoren und dem charakteristischen Polynom dar.

[12] Für die Berechnung Jordanscher Normalformen ist zum Beispiel auch die Bestimmung von Haupträumen notwendig, wie sie in [Fischer, 2002, Seite 259 ff.] dargestellt wird, die ich aber nicht erwähne, weil ich keine Aufgaben dazu generiert habe.

[13] Die Determinantenfunktion ist eigentlich nur über einem Körper definiert. (Siehe Definition 4.24.) $K[\lambda]$ ist lediglich ein Polynomring. Allerdings ist jeder Polynomring $R[X]$ ein Integritätsring, wenn R ein Integritätsring ist. Ein Beweis dieser Behauptung findet sich in [Bosch, 2004, Seite 32]. Jeder Körper ist ein Integritätsring, also ist auch $K[\lambda]$ ein Integritätsring. Und damit lässt sich zu $K[\lambda]$ ein Quotientenkörper konstruieren, nämlich $K(\lambda)$, in den $K[\lambda]$ eingebettet wird, sodass $K[\lambda] \subset K(\lambda)$ gilt. Deshalb ist die Determinantenfunktion auch über Integritätsringen und damit auch über $K[\lambda]$ definiert.

[14] Vergleiche auch [Holz & Wille, 2002, Seite 86].

[15] Wäre dem so, dann könnten mit Hilfe von Nullstellen von Polynomen auch keine Körpererweiterungen vorgenommen werden, denn der folgende Satz zeigt, dass die Eigenwerte die Nullstellen des charakteristischen Polynoms sind.

Satz 3.5

Sei $A \in M(n \times n; K)$, K ein Körper, dann gilt:

$$\lambda \in K \text{ ist ein Eigenwert von } A \Leftrightarrow p_M(\lambda) = 0.$$

Die Nullstellen des charakteristischen Polynoms einer Matrix A sind also deren Eigenwerte und umgekehrt.

Beweis:

$\Rightarrow$ Wenn λ ein Eigenwert von A ist, dann gilt:

$$A \circ \vec{x} = \lambda \cdot \vec{x} \Leftrightarrow (A - \lambda \cdot E_n) \circ \vec{x} = \vec{0}.$$

Da nach Voraussetzung $\vec{x} \neq \vec{0}$ ist, liegt $\vec{x}$ im Kern von $(A - \lambda \cdot E_n)$. Damit ist $\operatorname{rang}(A - \lambda \cdot E_n) < n$ und es folgt:

$$\det(A - \lambda \cdot E_n) = p_A(\lambda) = 0.$$

$\Leftarrow$ Wenn $p_A(\lambda) = \det(A - \lambda \cdot E_n) = 0$, dann ist der Rang der Matrix $(A - \lambda \cdot E_n)$ kleiner als n und es gibt Vektoren $\vec{x} \in K^n$ mit $\vec{x} \neq \vec{0}$, sodass gilt:

$$(A - \lambda \cdot E_n) \circ \vec{x} = \vec{0} \Leftrightarrow A \circ \vec{x} = \lambda \cdot \vec{x}.$$

Also ist λ ein Eigenwert von A.

□

Satz 3.6

Sei $A \in M(n \times n; K)$, K ein Körper, dann gilt:

(i) Eigenvektoren zu verschiedenen Eigenwerten von A sind linear unabhängig.

(ii) Satz von CAYLEY-HAMILTON: $p_A(A) = \vec{0}$

Beweis:

(i) Ein Beweis findet sich in [Fischer, 2002, Seite 226].

(ii) Ein Beweis findet sich in [Fischer, 2002, Seite 251].

Hier ergeben sich eine ganze Reihe von Lernzielen, es wäre müßig alle aufzuzählen, weshalb ich mich auf die für mich wesentlichsten beschränke. Voraussetzung für das Erreichen von Lernziel 3.2 ist demnach das Erreichen von folgenden Lernzielen,

Lernziel 3.4

Die Studenten sollen den Begriff des Eigenwertes kennen, um dessen Eigenschaften (vor allem in Bezug auf das charakteristische Polynom) wissen, die Eigenwerte zu einer gegebenen Matrix bestimmen und ihre Eigenschaften in verschiedenen Aufgaben ausnutzen können.

Lernziel 3.5

Die Studenten sollen den Begriff des Eigenvektors kennen, um dessen Eigenschaften (vor allem in Bezug auf die Diagonalisierbarkeit einer Matrix) wissen, die Eigenvektoren zu einer gegebenen Matrix bestimmen und ihre Eigenschaften in verschiedenen Aufgaben ausnutzen können.

wobei die wesentlichen Eigenschaften mit Hilfe der obigen Sätze schon dargestellt worden sind.

Eigentlich könnte man sich an dieser Stelle, auch damit beschäftigen, unter welchen Bedingungen Polynome über Körpern zerfallen, weil die Nullstellen mit den Eigenwerten identisch sind. Dies ist allerdings Teil der Algebra und nicht der linearen Algebra. Hier reicht es auch zu wissen, dass über $\mathbb{C}$ alle Polynome zerfallen.

Lernziel 3.6

Die Studenten sollen wissen, dass alle Polynome über $\mathbb{C}$ in Linearfaktoren zerfallen.

Tatsächlich reichen die obigen Eigenschaften sogar schon aus, um spezielle Jordansche Normalformen zu finden, deren Jordankästchen alle die Größe 1 haben.

Definition 3.7

(i) Eine Matrix $A \in M(n \times n; K)$, K ein Körper, heißt diagonalisierbar, wenn A ähnlich zu einer Matrix in Diagonalgestalt ist.

(ii) Sei $A \in M(n \times n; K)$, K ein Körper, und $\lambda \in K$ ein Eigenwert von A.

  (a) Gilt $(x - \lambda)^k$ teilt p_A, aber $(x - \lambda)^{k+1}$ teilt p_A nicht, $k \in \mathbb{N}$, so heißt k die algebraische Vielfachheit von λ.

(b) Die geometrische Vielfachheit von λ ist:

$$\dim\left(V_\lambda(A)\right) = n - \operatorname{rang}\left(A - \lambda \cdot E_n\right).$$

[16]

Folgerung 3.8

Ist A ähnlich zu einer Diagonalmatrix D, dann stehen auf der Hauptdiagonalen der Diagonalmatrix D die Eigenwerte von D und damit auch die Eigenwerte von A, weil ähnliche Matrizen das gleiche charakteristische Polynom haben.

Satz 3.9

Sei $A \in M(n \times n; K)$, K ein Körper, dann sind die folgenden Aussagen äquivalent und dienen damit als Kriterium der Diagonalisierbarkeit:

(i) A ist über K diagonalisierbar.

(ii) Der K^n besitzt eine Basis aus Eigenvektoren von A.

(iii) p_A, das charakteristische Polynom von A, zerfällt über K in Linearfaktoren und für jeden Eigenwert $\lambda \in K$ von A ist seine geometrische Vielfachheit gleich seiner algebraischen Vielfachheit.

(iv) m_A, das Minimalpolynom von A, zerfällt über K in paarweise verschiedene Linearfaktoren.

(v) Der K^n ist die direkte Summe der Eigenräume von A. Sind also die $\lambda_1, ..., \lambda_k \in K$ die paarweise verschiedenen Eigenwerte von A, so gilt:

$$K^n = V_{\lambda_1}(A) \oplus V_{\lambda_2}(A) \oplus ... \oplus V_{\lambda_k}(A).$$

[17]

Beweis: Ein Beweis der Äquivalenz der Eigenschaften *(i)*, *(iii)* und *(v)* findet sich in [Fischer, 2002, Seite 235 f.].
(v)⇔(ii) ist klar.(i)⇔(vi) folgt aus der Eigenschaft, dass der größte Wert für Größe eines Jordanblockes zu einem Eigenwert die Vielfachheit des zugehörigen Linearfaktors im Minimalpolynom festlegt und umgekehrt.

Eine Matrix in Diagonalform ist eigentlich nur eine Matrix, bei der alle Jordanblöcke die Größe 1 haben. Das Diagonalisieren einer diagonalisierbaren Matrix ist also nur ein einfacher Fall der Berechnung der Jordanschen Normalform. Um die Problemstellung zu motivieren, für jede Matrix

[16] Vergleiche auch [Holz & Wille, 2002, Seite 104].
[17] Vergleiche auch [Holz & Wille, 2002, Seite 105].

$A \in M(n \times n; K)$ eine möglichst einfache Matrix $J \in M(n \times n; K)$ zu finden, die ähnlich zu A ist, kann man also folgendes Lernziel definieren,

Lernziel 3.7

Die Studenten sollen den Begriff der Diagonalisierbarkeit kennen, wissen, unter welchen Umständen eine Matrix diagonalisierbar ist, eine diagonalisierbare Matrix diagonalisieren und ihre Eigenschaften in verschiedenen Aufgaben ausnutzen können.

wobei man auf die Erkenntnisse, die man beim Erreichen dieses Lernzieles gewinnt, später zurückgreifen muss, wenn man Jordansche Normalformen per Matrizenaddition in eine diagonalisierbare und eine nilpotente Matrix zerlegt. Auch das Erreichen dieses Lernziels ist demnach Voraussetzung dafür, dass die Studenten in die Lage versetzt werden, die Jordansche Normalform zu verstehen.

3.1.3 Die Jordansche Normalform

Verschiedene Eigenschaften der Jordanschen Normalform folgen aus den Eigenschaften der nilpotenten Matrizen, weil man jede Matrix $A \in M(n \times n; K)$ additiv in eine diagonalisierbare und eine nilpotente Matrix zerlegen kann.[18] Also stellt auch das Erreichen von Lernzielen im Bereich der nilpotenten Matrizen eine Voraussetzung für das Erreichen von Lernziel 3.3 dar.

Definition 3.10

Sei $A \in M(n \times n; K)$, K ein Körper, dann heißt A nilpotent, falls $A^n = 0$ gilt. Das kleinste $k \in \mathbb{N}, k \leq n$, mit $A^k = 0$ nennt sich dann der Nilpotenzgrad von A.[19]

Wichtige Eigenschaften nilpotenter Matrizen sind die folgenden:

Satz 3.11

Es sei $A \in M(n \times n; K)$, K ein Körper, dann sind die folgenden Aussagen äquivalent:

(i) *A ist nilpotent.*

(ii) $p_A = (-1)^n \cdot x^n$

(iii) $m_A = x^k$ *für ein* $k \in \mathbb{N}$ *mit* $k \leq n$

[18] Vergleiche auch [Fischer, 2002, Seite 259 ff.].
[19] Vergleiche auch [Holz & Wille, 2002, Seite 125].

(iv) *A ist ähnlich zu einer oberen Dreiecksmatrix, deren Eigenwerte alle gleich Null sind.*

(v) $A^n = 0$

[20]

Beweis: Ein Beweis findet sich in [Holz & Wille, 2002, Seite 126 f.].

Lernziel 3.8

Die Studenten sollen den Begriff der Nilpotenz einer Matrix kennen, um seine Eigenschaften (vor allem in Bezug auf die Jordansche Normalform und das Minimalpolynom) wissen, den Nilpotenzgrad einer Matrix berechnen und die Eigenschaften der Nilpotenz in verschiedenen Aufgaben ausnutzen können.

Mit dem Minimalpolynom kann man die Größe der einzelnen Jordanblöcke in einer Jordanschen Normalform bestimmen. Das Minimalpolynom stellt also eine weitere diesmal aber nicht unbedingt notwendige Voraussetzung dar, eine Jordansche Normalform zu berechnen.[21]

Definition 3.12

Es sei $A \in M(n \times n; K)$, dann heißt m_A das Minimalpolynom von A, wenn Folgendes gilt:

(i) $m_A(A) = 0$

(ii) m_A ist das Polynom $f \in K[X], f \neq 0$, mit dem kleinsten Grad, für das $f(A) = 0$ gilt.

(iii) m_A ist normiert, das heißt, der Leitkoeffizient von m_A ist 1.

[22]

Wichtige Sätze über das Minimalpolynom sind die folgenden:

Satz 3.13

Sei $A \in M(n \times n; K)$, K ein Körper, dann gilt:

(i) *$\lambda \in K$ ist genau dann ein Eigenwert von A, wenn $m_A(\lambda) = 0$ gilt.*

(ii) *Ist $f \in K[X]$, der Grad von f größer Null und $f(A) = 0$, so teilt m_A f. Insbesondere teilt m_A auch p_A.*

[20] Vergleiche auch [Holz & Wille, 2002, Seite 126].
[21] Fischer [Fischer, 2002, Seite 259 ff.] benutzt es zum Beispiel nicht dafür.
[22] Vergleiche auch [Holz & Wille, 2002, Seite 96].

(iii) p_A *und* m_A *besitzen dieselben irreduziblen Teiler in* $K[X]$.

(iv) Die Dimension von $K[A]$ *ist gleich dem Grad von* m_A.

[23]

Beweis:

(i) Folgt aus (ii) und p_A teilt ${m_A}^n$ nach [Fischer, 2002, Seite 257].

(ii) Ein Beweis findet sich in [Holz & Wille, 2002, Seite 97].

(iii) Folgt aus (ii) und p_A teilt ${m_A}^n$ nach [Fischer, 2002, Seite 257].

(iv) Ist klar, weil $m_A(A) = 0$ gilt und m_A das Polynom mit dem kleinsten Grad ist, für das diese Eigenschaft gilt.

Lernziel 3.9

Die Studenten sollen den Begriff des Minimalpolynoms kennen, um seine Eigenschaften (vor allem in Bezug auf die Jordansche Normalform und das charakteristische Polynom) wissen, es berechnen und seine Eigenschaften in verschiedenen Aufgaben ausnutzen können.

Eine letzte Voraussetzung, die ich hier vorstellen möchte, wären dann die Jordankästchen oder Jordanblöcke, da sich die Jordanschen Normalformen genau aus diesen zusammensetzen.

Definition 3.14

(i) Eine Matrix der Form

$$\begin{pmatrix} \lambda & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda & 1 & \cdots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \cdots & \lambda & 1 \\ 0 & 0 & 0 & \cdots & 0 & \lambda \end{pmatrix}$$

heißt Jordankästchen oder Jordanblock.
Auf der Hauptdiagonalen steht immer das gleiche $\lambda \in K$, K ein Körper. In der oberhalb gelegenen Nebendiagonalen stehen Einsen, ansonsten sind alle Einträge gleich Null.

[23] Vergleiche auch [Holz & Wille, 2002, Seite 96].

(ii) Eine Matrix $A \in M(n \times n; K)$, K ein Körper, hat Jordansche Normalform, wenn sie das folgende Aussehen hat:

$$\begin{pmatrix} J_1 & 0 & \cdots & 0 \\ 0 & J_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & J_m \end{pmatrix}$$

Dabei sind die Blockmatrizen J_i, $i \in \{1, ..., m\}$, Jordankästchen.

(iii) Ist A über K ähnlich zu einer Matrix $B \in M(n \times n; K)$, so heißt A eine Jordansche Normalform von B.
Die Jordansche Normalform ist bis auf die Reihenfolge der Jordankästchen eindeutig bestimmt.

[24]

Lernziel 3.10

Die Studenten sollen den Begriff des Jordanblocks (Jordankästchens) kennen, um seine Eigenschaften (vor allem in Bezug auf die Jordansche Normalform und das Minimalpolynom) wissen, alle Jordanblöcke zur Berechnung der Jordanschen Normalform bestimmen und ihre Eigenschaften in verschiedenen Aufgaben ausnutzen können.

Als ich Lernziel 3.3 definiert habe, habe ich auch noch Eigenschaften der Jordanschen Normalform angesprochen. Bevor ich zu den Aufgaben komme, will ich diese noch kurz vorstellen.

Satz 3.15

Die folgenden Aussagen sind äquivalent:

(i) Eine Matrix $A \in M(n \times n; K)$, K ein Körper, besitzt eine Jordansche Normalform über K.

(ii) Das charakteristische Polynom p_A von A zerfällt über K in Linearfaktoren.

(iii) Das Minimalpolynom m_A von A zerfällt über K in Linearfaktoren.

[25]

[24] Vergleiche auch [Holz & Wille, 2002, Seite 165].
[25] Vergleiche auch [Holz & Wille, 2002, Seite 166].

Beweis: Für (i)⇐(ii) findet sich ein Beweis in [Fischer, 2002, Seite 259 ff.]. (i)⇒(ii) gilt, weil ähnliche Matrizen das gleiche charakteristische Polynom besitzen und A eine Jordansche Normalform besitzt, zu der A dann auch ähnlich ist. (ii)⇔(iii), weil m_A und p_A über K dieselben irreduziblen Teiler besitzen.

Ich will hier noch einmal die Eigenschaft herausstellen, mit der man immer entscheiden kann, ob zwei Matrizen zueinander ähnlich sind, auch wenn ich sie oben schon erwähnt habe.

Satz 3.16

(i) Zwei Matrizen, deren charakteristische Polynome über K in Linearfaktoren zerfallen, sind genau dann ähnlich, wenn sie dieselbe Jordansche Normalform besitzen.

(ii) Zwei Matrizen $A, B \in M(n \times n; K)$ sind genau dann ähnlich über K, wenn es einen Oberkörper von K gibt, über dem beide Matrizen dieselbe Jordansche Normalform besitzen.
Dieser Oberkörper ist spätestens der algebraische Abschluss von K, weil über dem algebraischen Abschluss von K alle Polynome aus $K[X]$ in ihre Linearfaktoren zerfallen.

Beweis: Die Quellen der zugehörigen Beweise habe ich zu Beginn dieses Abschnittes schon aufgeführt.

3.1.4 Entwicklung der Aufgaben

Mit Hilfe der Lernziele und deren Voraussetzungen habe ich nun also eine mögliche Reihenfolge im Vorgehen vorgestellt. Für die Aufgaben bedeutet dies, dass sie eine solche Reihenfolge möglich machen und die Lernziele im Hinblick auf den Transfer des zu Lernenden unterstützen müssen. Doch bevor ich mit der Vorstellung der Aufgaben beginne, will ich noch das folgende Ergebnis sichern:

Satz 3.17

Sei $A \in M(n \times n; K)$ eine Matrix in Jordanscher Normalform,

$$A = \begin{pmatrix} J_1 & 0 & \cdots & 0 \\ 0 & J_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & J_m \end{pmatrix}$$

wobei die J_i die Jordanblöcke zum Eigenwert λ_i sind, dann gilt:

$$p_A(x) = \prod_{i=1}^{m} (\lambda_i - x)^{g_i},$$

wobei g_i die Größe des i-ten Jordanblockes J_i in A ist.

$$m_A(x) = \prod_{j=1}^{r} (x - \lambda_j)^h,$$

wobei r die Anzahl der verschiedenen Eigenwerte $\lambda_j$von A ist und h die Größe des größten Jordanblockes zu λ_j in A angibt.

Beweis: $p_A(x) = \prod_{i=1}^{m} (\lambda_i - x)^{g_i}$ ergibt sich sofort aus der Definition des charakteristischen Polynoms.
Der Exponent der Linearfaktoren im Minimalpolynom bestimmt sich aus der Zerlegung in Haupträume wie in [Holz & Wille, 2002, Seite 156 und Seite 162 f.] gezeigt wird. Mit dem Beweis des Satzes über die Hauptraumzerlegung wie in [Fischer, 2002, Seite 259 ff.] folgt dann die Behauptung.

Beispiel 3.18

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Das charakteristische Polynom p_A der Matrix A ist:

$$p_A(x) = -(x-1)^5.$$

Das Minimalpolynom der Matrix A ist:

$$m_A(x) = (x-1)^2.$$

Auf diese Erkenntnisse wird bei der Vorstellung der Aufgabenstellungen immer wieder Bezug genommen. Zusätzlich beschäftige ich mich dort noch einmal mit der Eindeutigkeit der Lösungen, weil dies die Eigenschaft ist, die es erlaubt, dass solche Aufgaben mit OKUSON gestellt werden können und gebe Kontrollalgorithmen an. Falls ich es nicht erwähne, ist eine Aufgabe mit OKUSON kompatibel. Außerdem findet sich zu jeder Aufgabe eine Tabelle, die ihre wichtigsten Eigenschaften zusammenfasst.

Aufgaben zu Eigenwerten und Eigenvektoren

Folgende Aufgabenstellungen leiten sich direkt aus der Definition von Eigenwerten und -vektoren ab.

Aufgabenstellung 3.1

Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ alle Eigenwerte an.

Einstellung im Fenster: „Eigenwerte bestimmen“ [26]

Das Programm realisiert diese Aufgabenstellung, indem es mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform beginnt. Die Eigenwerte von der Matrix B stehen jetzt schon auf der Hauptdiagonalen von A, weil dies die Nullstellen des zugehörigen charakteristischen Polynoms p_A sind und charakteristische Polynome von ähnlichen Matrizen übereinstimmen. Das nutze ich aus. Mein Programm erzeugt nun eine Matrix B, die über $\mathbb{Z}$ ähnlich zu A ist, damit die Eigenwerte von B die gleichen sind wie die von A. Dies geschieht, indem das Programm ein Paar P, P^{-1} von zueinander inversen Matrizen mit ganzzahligen Einträgen findet und dann B über $B = P \circ A \circ P^{-1}$ berechnet. Das Erzeugen einer Jordanschen Normalform ist leicht, wie unten gezeigt wird. Die Schwierigkeit liegt darin, ein Paar P, P^{-1} von über $\mathbb{Z}$ invertierbaren Matrizen zu erzeugen, sodass $B = P \circ A \circ P^{-1}$ gilt. P und P^{-1} müssen über $\mathbb{Z}$ invertierbar sein, damit gewährleistet ist, dass auch B wieder nur Einträge aus $\mathbb{Z}$ enthält. Die Lösung, also die Eigenwerte, sind natürlich eindeutig; zur Kontrolle reicht also ein Abgleich.

Für die Okusonversion dieser Aufgabe habe ich eine Änderung an dieser Aufgabenstellung vorgenommen, weil die Eingabe in ein Textfeld häufig zu Fehlern bei den Studenten führt und man dann von Hand nachkorrigieren müsste. Das Aufstellen von Regeln für die Eingabe, um diese Fehler zu vermeiden, bringt nach Aussage der Anwender relativ wenig. Diese Regeln würden häufig ignoriert werden. Außerdem sollte die Eingabe nicht komplizierter sein als die Aufgabe an sich. Deshalb werden zwei Lösungsmöglichkeiten für die Eigenwerte (später auch für das charakteristische Polynom, das Minimalpolynom und den Nilpotenzgrad) angeboten, eine falsche und eine richtige. Die Aufgabe besteht nun darin, die richtige Lösung auszuwählen. Die Frage, warum es nur eine falsche Lösung gibt, lässt sich damit beantworten, dass mein Programm nur in der Lage ist, Eigenwerte, das charakteristische Polynom, das Minimalpolynom und den Nilpotenzgrad aus Jordanschen Normalformen auszulesen. Zu einer gegebenen Jordanschen Normalform baut es sich in manchen Fällen auf nur eine Weise eine neue Jordansche Normalform für die falsche(n) Lösung(en). Für andere Optionen reicht dies, weil

[26] Dies meint die Option im Startfenster zum Aufgabenbereich der ähnlichen Matrizen.

„Eigenwerte bestimmen“:	
Aufgabenstellung: Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ alle Eigenwerte an.	
Optionen:	
Gegeben:	Matrix $B \in M(n \times n; \mathbb{Z})$
Gesucht:	Eigenwerte von B
Variabel:	n
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	mit Einschränkungen möglich
Parameteraufruf mit:	31

Tabelle 3.1: „Eigenwerte bestimmen“

man diese neue Jordansche Normalform noch mit P, P^{-1} verändern kann, ohne dass die Eigenwerte, das charakteristische Polynom, das Minimalpolynom oder der Nilpotenzgrad verändert werden. Das hilft hier aber nicht. Am Ende hakt es daran, dass es Jordansche Normalformen gibt, bei denen mein Programm nicht einmal mehr einen zufälligen Eigenwert in die Matrix schreibt, sondern nur eine 1 in der Nebendiagonalen in eine 0 umschreibt oder umgekehrt, sodass weitere Durchläufe durch diesen Algorithmus mit den gleichen Startparametern immer die gleiche Ausgabe bewirken werden. Eine Änderung der Startparameter birgt jedoch das Risiko, dass eine als falsch deklarierte Lösung, dann doch richtig ist, weil die Jordanblöcke nur ihre Reihenfolge geändert haben könnten. Also bleibt nur die Variante mit zwei Lösungen, einer richtigen und einer falschen.
Bei dieser Option nutzt mein Programm zur Erzeugung falscher Lösungen den Algorithmus von Aufgabenstellung 3.4. Hier treten noch keine Schwierigkeiten auf. Allerdings greift diese Aufgabe im Programm zur Verringerung der Schreibarbeit und der Übersichtlichkeit halber mit anderen Aufgaben des gleichen Typs auf die gleichen Routinen zurück. Dies zwingt mich auch bei dieser Aufgabe zu der eigentlich unnötigen Einschränkung.

Bleibt die Frage, wie das Programm eine Matrix in Jordanscher Normalform baut. Das Programm beginnt damit, dass es überprüft, welche Größe ihm für einen Jordanblock zur Verfügung steht. Zu Beginn ist dies natürlich, die Zeilenanzahl z der quadratischen Matrix. Jetzt bestimmt es per Zufall die Größe des ersten Jordanblocks und den zugehörigen Eigenwert, wobei dieser im Intervall $[-2z, 2z]$ liegt, und füllt die entsprechenden Bereiche in der Matrix aus. Dann überprüft es wieder, wie groß die Jordanblöcke noch sein

können, wobei die Maximalgröße nicht nur über die verbliebenen auszufüllenden Zeilen, sondern auch über die Größe des letzten Jordanblockes festgelegt ist. Diese zusätzliche Einschränkung sorgt dafür, dass die Blöcke gleich der Größe nach sortiert sind, wenn auch nicht nach den Eigenwerten, denn dieser wird wieder per Zufall aus dem Intervall $[-2z, 2z]$ gewählt. Das Verfahren wird fortgesetzt, bis alle Zeilen angefüllt sind, was schon nach einem Schritt der Fall sein kann.
Alle weiteren Aufgaben, die eine Startmatrix in Jordanscher Normalform benötigen, greifen auf dieses Verfahren zurück.

Aufgabenstellung 3.2

Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ das charakteristische Polynom an.

Einstellung im Fenster: „char. Polynom bestimmen“

Das Programm beginnt mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform, die wie oben bestimmt wird. Damit ist das charakteristische Polynom von B schon festgelegt, weil mein Programm wie oben eine zu A ähnliche Matrix B zusammenbaut, indem es ein Paar von zueinander inversen Matrizen P, P^{-1} mit ganzzahligen Einträgen findet. Es berechnet dann B über $B = P \circ A \circ P^{-1}$. Das charakteristische Polynom von B ist das gleiche wie das von A, weil ähnliche Matrizen das gleiche charakteristische Polynom haben.
Der gleiche Trick wie oben, das Finden eines Paares von zueinander inversen Matrizen P, P^{-1} mit ganzzahligen Einträgen, wird nun immer wieder verwendet werden, um die Eigenschaften von ähnlichen Matrizen einfach ausnutzen zu können. Die Lösung ist eindeutig, weil das charakteristische Polynom so, wie es hier definiert worden ist, eindeutig ist, wodurch auch hier ein Abgleich als Kontrolle reicht.

Auch bei dieser Aufgabe gibt es die oben angesprochene Okusonvariante, bei der zwischen einer richtigen und einer falschen Lösung unterschieden werden muss. Diese Aufgabe greift zur Erzeugung der falschen Lösung auf den Algorithmus von Aufgabenstellung 3.5 zurück. Schaut man sich die zugehörigen Darstellungen an, dann erkennt man, dass gleich der erste Fall, die Möglichkeit von mehr als einer falschen Lösung verhindert.

Aufgabenstellung 3.3

Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ alle Eigenwerte mit den zugehörigen Eigenvektoren $\vec{x} \in \mathbb{C}^n$ an.

Einstellung im Fenster: „Eigenvektoren bestimmen“

„char. Polynom bestimmen“:

Aufgabenstellung:
Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ das charakteristische Polynom an.

Optionen:

Gegeben:	Matrix $B \in M(n \times n; \mathbb{Z})$
Gesucht:	p_B
Variabel:	n
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	mit Einschränkungen möglich
Parameteraufruf mit:	32

Tabelle 3.2: „char. Polynom bestimmen“

Mein Programm ist so aufgebaut, dass eine Matrix A vorgegeben wird, zu der dann nach dem Finden von P, P^{-1} eine ähnliche Matrix B bestimmt wird, sodass $B = P \circ A \circ P^{-1}$ gilt. Die einzige Möglichkeit eine solche Aufgabe zu realisieren besteht darin, diese Gleichung so umzuformen, dass $B \circ P = P \circ A$ gilt. Ist A nun eine Diagonalmatrix, so werden die Spalten von P mit den Einträgen in der zugehörigen Hauptdiagonalen multipliziert. Also kann man mit der Einschränkung, dass A eine Diagonalmatrix ist, solche Aufgaben erzeugen. Die gesuchten Eigenvektoren sind die Spalten von P.
Es bleibt wie in den Aufgaben zuvor die Frage, wie man solche P erzeugt. Dieses Problem wird auch in allen folgenden Aufgaben auftreten und erst in Kapitel 4 gelöst werden.
Probleme gibt es hier mit der Eindeutigkeit der Lösungen, denn im eigentlichen Sinn sind Basen eines Vektorraumes gesucht. Das Programm liefert eine Lösung, aber nicht alle, zumal die Eigenwerte auch noch beliebig in A angeordnet werden können. Beispiel 2.2 zeigte schon einen solchen Fall, wenn A keine Diagonalmatrix ist. Doch auch in diesem Fall ist P nicht eindeutig:

1. Ein Eigenwert λ könnte in A mehrfach vorkommen, dann hätte der Eigenraum zu λ mindestens die Dimension 2, was bedeutet, dass man keine eindeutige Basis und auch kein eindeutiges P mehr finden kann.

2. Selbst wenn das nicht der Fall ist, kann man jeden Eigenvektor - selbst einen mit ganzzahligen Koeffizienten, deren ggt immer 1 ist - mit dem Skalar -1 multiplizieren. Für $P \in GL(n; \mathbb{Z})$ erhält man dann immer noch 2^n Lösungen.

3. Neben P ist auch $-P$ in allen Fällen, bei denen die konjugierende

„Eigenvektoren bestimmen“:	
Aufgabenstellung: Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ alle Eigenwerte mit den zugehörigen Eigenvektoren $\vec{x} \in \mathbb{C}^n$ an.	
Optionen:	
Gegeben:	diagonalisierbare Matrix $B \in M(n \times n; \mathbb{Z})$
Gesucht:	Eigenwerte und -vektoren von B (Spalten von P)
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B = P \circ A \circ P^{-1}$
Verwendung mit OKUSON:	nicht möglich, da P uneindeutig
Parameteraufruf mit:	41

Tabelle 3.3: „Eigenvektoren bestimmen“

> Matrix P als Lösung angegeben werden soll, eine weitere Lösung, was sich sofort aus den Rechenregeln für Matrizen ergibt.[27]

Also muss man einen Algorithmus angeben, mit dem man die Lösung kontrollieren kann. Das ist aber dank der Ähnlichkeit von A und B nicht weiter schwierig. Es muss einfach $B = P \circ A \circ P^{-1}$ gelten. Wichtig ist, dass sich eine solche Aufgabe deshalb für OKUSON nicht verwenden lässt, weil zur Kontrolle die definierende Eigenschaft nachgerechnet werden müsste, was OKUSON nicht kann.[28]

Bis jetzt habe ich Aufgaben gestellt, bei denen Eingangsdaten gegeben sind und der Lösungsweg nach mehrmaligem Üben bekannt sein sollte. Diese Aufgabenstellung kann natürlich, wie im Unterabschnitt 1.2.3 dargestellt, zu Zuordnungsaufgaben verändert werden. Für Mehrfachaufgaben benötige ich jedoch weitere Algorithmen, die sinnvolle falsche Lösungen produzieren. Diese Algorithmen basieren auf dem schon bekannten Prinzip und werden im Zuge der folgenden Aufgabenstellungen vorgestellt.

Aufgabenstellung 3.4

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben den oder die folgenden Eigenwerte $\lambda_1, \lambda_2, ...$?

Einstellung im Fenster: „Ausschluss über Eigenwerte“

[27] Vergleiche auch Bemerkung 3.22.

[28] Allerdings kann man eine solche Uneindeutigkeit ausnutzen, um andere Aufgaben zu bauen, auch wenn ich dies hier nicht getan habe.

Das Programm beginnt mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform. Die Eigenwerte von A stehen damit schon auf der Hauptdiagonalen, weil dies die Nullstellen des zugehörigen charakteristischen Polynoms sind. Das Programm erzeugt nun eine oder mehrere verschiedene Matrizen B, die über $\mathbb{Z}$ ähnlich zu A sind. Die Eigenwerte der Matrizen B sind die gleichen wie die von A, weil ähnliche Matrizen das gleiche charakteristische Polynom haben. Damit gewinnt man also eine bzw. mehrere richtige Lösungen.
Falsche Lösungen werden demzufolge am einfachsten durch eine Abänderung der gewählten Jordanschen Normalform A erreicht. In der Jordanschen Normalform stehen auf der Hauptdiagonalen die Eigenwerte. Mein Programm verändert, wie unten in vereinfachter Form dargestellt, den letzten Eigenwert so, dass die entstehende Matrix zwar immer noch in Jordanscher Normalform ist, diese aber nicht mehr die gleichen Eigenwerte hat wie zu Beginn.

$$\begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_{r-1} & 0 \\ 0 & 0 & \lambda_r \end{pmatrix} \rightarrow \begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_{r-1} & 0 \\ 0 & 0 & \tilde{\lambda}_r \end{pmatrix} \text{ mit } \tilde{\lambda}_r \neq \lambda_i \, \forall i \in \{1, ..., r\}$$

bzw.

$$\begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_r & 1 \\ 0 & 0 & \lambda_r \end{pmatrix} \rightarrow \begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_r & 0 \\ 0 & 0 & \lambda_{r+1} \end{pmatrix} \text{ mit } \lambda_{r+1} \neq \lambda_i \, \forall i \in \{1, ..., r\}$$

Auf diese Weise baut das Programm eine neue Jordansche Normalform als Grundlage für die falschen Lösungen, die zu dieser neuen Jordanschen Normalform ähnlich sind. Diese falschen Lösungen werden auf die gleiche Weise wie die richtigen gewonnen. Man findet eine über $\mathbb{Z}$ invertierbare Matrix P und multipliziert diese von links und ihre Inverse P^{-1} von rechts an die neue Jordansche Normalform. Da diese, wie erkennbar ist, einen neuen Eigenwert enthält und ähnliche Matrizen die gleichen Eigenwerte besitzen, haben die falschen Lösungen genau einen Eigenwert, in dem sie sich von den richtigen unterscheiden. Dass dieser Eigenwert nicht schon im „Rest“ vorkommt, wird dadurch garantiert, dass das Programm zuvor überprüft hat, welche Eigenwerte schon vorhanden waren. Der neue Eigenwert stammt allerdings ebenfalls aus dem Intervall $[-2z, 2z]$. Weil richtige und falsche Lösungen bekannt sind und diese auch die entsprechenden Eigenschaften haben, reicht ein Abgleich zur Kontrolle.

Die gleiche Aufgabenstellung gibt es auch zum charakteristischen Polynom.

„Ausschluss über Eigenwerte“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben den oder die folgenden Eigenwerte $\lambda_1, \lambda_2, ...$?

Optionen:

Gegeben:	Eigenwerte und Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen mit den gegebenen Eigenwerten
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	25

Tabelle 3.4: „Ausschluss über Eigenwerte“

Aufgabenstellung 3.5

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben das folgende charakteristische Polynom?

Einstellung im Fenster: „Ausschluss über char. Polynom“

Das Programm beginnt mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform. Für A selber und für alle zu A ähnlichen Matrizen ist damit das charakteristische Polynom schon festgelegt. Mein Programm erzeugt nun auf gewohnte Weise eine oder mehrere verschiedene Matrizen B, die über $\mathbb{Z}$ ähnlich zu A ist. Das charakteristische Polynom von B ist deshalb das gleiche wie das von A und man gewinnt auf diese Weise eine bzw. mehrere richtige Lösungen.
Falsche Lösungen werden durch eine Abänderung der gewählten Jordanschen Normalform erreicht. In der Jordanschen Normalform stehen auf der Hauptdiagonalen die Eigenwerte, die das charakteristische Polynom festlegen. Mein Programm verändert, wie unten in vereinfachter Form dargestellt, den letzten Eigenwert so, dass die entstehende Matrix immer noch in Jordanscher Normalform ist, aber dem charakteristischen Polynom dieser Matrix entweder ein Linearfaktor fehlt, es einen neuen Linearfaktor enthält und/oder die Exponenten geändert worden sind. Diese neue Matrix bildet die Grundlage

„Ausschluss über char. Polynom“:	
Aufgabenstellung: Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben das folgende charakteristische Polynom?	
Optionen:	
Gegeben:	p_A und Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen mit $p_B = p_A$
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	26

Tabelle 3.5: „Ausschluss über char. Polynom“

für die falschen Lösungen.

$$\begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_{r-1} & 0 \\ 0 & 0 & \lambda_r \end{pmatrix} \rightarrow \begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_{r-1} & 1 \\ 0 & 0 & \lambda_{r-1} \end{pmatrix} \text{ mit } \lambda_{r-1} \neq \lambda_r$$

$$\begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_r & 0 \\ 0 & 0 & \lambda_r \end{pmatrix} \rightarrow \begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_r & 0 \\ 0 & 0 & \lambda_{r+1} \end{pmatrix} \text{ mit } \lambda_r \neq \lambda_{r+1}$$

bzw.

$$\begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_r & 1 \\ 0 & 0 & \lambda_r \end{pmatrix} \rightarrow \begin{pmatrix} \text{Rest} & * & 0 \\ 0 & \lambda_r & 0 \\ 0 & 0 & \lambda_{r+1} \end{pmatrix} \text{ mit } \lambda_r \neq \lambda_{r+1}$$

Auf die gleiche Weise wie in der vorangegangenen Aufgabe baut das Programm mit Hilfe dieser neuen Jordanschen Normalform als Grundlage die falschen Lösungen, die sich von den richtigen Lösungen entweder durch einen neuen bzw. fehlenden Linearfaktor und/oder durch Unterschiede in den Exponenten eines Linearfaktors unterscheiden, weil ähnliche Matrizen das gleiche charakteristische Polynom haben. Zur Kontrolle reicht auch hier ein Abgleich, weil richtige und falsche Lösungen bekannt sind und auch die entsprechenden Eigenschaften besitzen.

Zur Diagonalisierbarkeit von Matrizen habe ich folgende Aufgabenstellungen in das Programm integriert:

„Diagonalisierung“:

Aufgabenstellung:
Zeigen Sie, dass die folgende Matrix $B \in M(n \times n; \mathbb{Z})$ über diagonalisierbar ist, indem Sie eine Matrix $P \in GL(n; \mathbb{Z})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt, wobei A Diagonalgestalt besitzt.

Optionen:

Gegeben:	Matrix $B \in M(n \times n; \mathbb{Z})$
Gesucht:	Diagonalmatrix $A \in M(n \times n; \mathbb{Z})$, $P \in GL(n; \mathbb{Z})$
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B = P \circ A \circ P^{-1}$
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	42

Tabelle 3.6: „Diagonalisierung“

Aufgabenstellung 3.6

Zeigen Sie, dass die folgende Matrix $B \in M(n \times n; \mathbb{Z})$ diagonalisierbar ist, indem Sie eine Matrix $P \in GL(n; \mathbb{Z})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt, wobei A Diagonalgestalt besitzt.

Einstellung im Fenster: „Diagonalisierung“

Diagonalisierbare Matrizen sind ähnlich zu einer Matrix in Diagonalgestalt. Mein Programm beginnt deshalb mit einer Ausgangsmatrix $A \in M(n \times n; \mathbb{Z})$ in Diagonalgestalt. Es findet nun P, P^{-1}, die den gleichen Bedingungen wie in den anderen Aufgabenstellungen genügen, und baut dann mit $B = P \circ A \circ P^{-1}$ eine zu A ähnliche Matrix B zusammen. Die Matrix B ist diagonalisierbar, weil sie ähnlich zur Matrix A ist.
Wie schon in Aufgabenstellung 3.3 dargestellt, ist die Matrix P uneindeutig, eine Anwendung mit OKUSON ist damit nicht möglich, weil man $B = P \circ A \circ P^{-1}$ kontrollieren muss.

Die folgende Aufgabe fragt ebenfalls nach der Diagonalisierbarkeit, erfordert aber bei der Kontrolle nicht das Überprüfen einer definierenden Eigenschaft.

Aufgabenstellung 3.7

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind diagonalisierbar?

Einstellung im Fenster: „Diagonalisierbarkeit“

„Diagonalisierbarkeit“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind diagonalisierbar?

Optionen:

Gegeben:	Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Diagonalisierbare Matrizen B
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	27

Tabelle 3.7: „Diagonalisierbarkeit“

Das Vorgehen zum Erzeugen richtiger Lösungen unterscheidet sich nicht von dem der letzten Aufgabenstellung, wenn man davon absieht, dass hier auch mehrere verschiedene richtige Lösungen erzeugt werden können. Falsche Lösungen erhält man, indem man mit Matrizen aus $M(n \times n; \mathbb{Z})$ startet, die zwar Jordansche Normalform haben, aber keine Diagonalgestalt. Mein Programm sorgt zudem dafür, dass die Mehrzahl der Eigenwerte der richtigen und der falschen Lösungen übereinstimmen, sodass hier kein zu einfaches Ausschlussprinzip gewonnen werden kann, da insbesondere auch der Fall auftreten kann, dass die Eigenwerte inklusive ihrer algebraischen Vielfachheiten übereinstimmen, nicht aber ihre geometrischen Vielfachheiten. Dazu benutzt es den Algorithmus von Aufgabenstellung 3.15. Die Lösungen sind somit eindeutig und entsprechen auch den angegebenen Eigenschaften, richtige Lösungen sind richtig, falsche falsch. Also reicht ein Abgleich zur Kontrolle aus.

Zur Diagonalisierung kann man noch eine Umkehraufgabe stellen.

Aufgabenstellung 3.8

Geben Sie eine diagonalisierbare Matrix $B \in M(n \times n; \mathbb{Z})$ an, die die gegebenen Eigenwerte und -vektoren besitzt. Achten Sie auf die Reihenfolge.

Einstellung im Fenster: „Umkehr der Diagonalisierung“

Die Ausgangsmatrix $A \in M(n \times n; \mathbb{Z})$ des Programms hat Diagonalgestalt. Mein Programm findet nun P, P^{-1} und erzeugt dann damit die diagonalisierbare Matrix B, indem es $B = P \circ A \circ P^{-1}$ ausführt. Die Spalten von P und der zugehörige Eintrag in der Spalte von A werden nun als Eigenvektor

„Umkehr der Diagonalisierung“:

Aufgabenstellung:
Geben Sie eine diagonalisierbare Matrix $B \in M(n \times n; \mathbb{Z})$ an, die die gegebenen Eigenwerte und -vektoren besitzt. Achten Sie auf die Reihenfolge.

Optionen:

Gegeben:	Eigenwerte und Eigenvektoren von B
Gesucht:	$B \in M(n \times n; \mathbb{Z})$
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B \circ \vec{x}_i = \lambda_i \cdot \vec{x}_i \ \forall i \in \{1, ..., n\}$; $\det(\vec{x}_1, ..., \vec{x}_n) \neq 0$
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	51

Tabelle 3.8: „Umkehr der Diagonalisierung“

mit zugehörigem Eigenwert vorgegeben. Die Lösung der Aufgabe ist dann die Matrix B. B ist auch eindeutig, weil die uneindeutige Matrix P vorgegeben ist.
Trotzdem ist die Eingabe einer Matrix bei OKUSON kompliziert. Ich habe deshalb davon abgesehen, diese Aufgabe mit OKUSON kompatibel zu machen. Den Trick, der bei den anderen Aufgaben mit den Eingabeschwierigkeiten funktioniert hat, wollte ich nicht verwenden, weil ich damit nur die Eigenwerte gezielt variieren könnte, aber keine ausreichende Kontrolle über die Eigenvektoren hätte, denn ähnliche Matrizen haben zwar gleiche Eigenwerte, nicht aber gleiche Eigenvektoren. Damit würde die Aufgabe noch schneller zu lösen sein, als sie es jetzt schon ist.
Die Kontrolle der Lösungen geschieht durch eine Überprüfung der definierenden Eigenschaft, nämlich $B \circ \vec{x}_i = \lambda_i \cdot \vec{x}_i$, wobei $\vec{x}_i$ die i-te Spalte von P und λ der zugehörige Eintrag in der i-ten Spalte von A ist. Diese Überprüfung muss dann für alle Eigenwerte und alle zugehörigen Eigenvektoren stattfinden. Zusätzlich muss überprüft werden, ob die n Eigenvektoren eine Basis des $\mathbb{C}^n$ bilden. Dazu kann man sie aneinandergereiht als Matrix auffassen, deren Determinante dann ungleich Null sein muss.

Zum Abschluss des Bereiches der Diagonalisierbarkeit kann man das erste Mal direkt nach einem Beweis der Ähnlichkeit von zwei Matrizen A, B fragen. Dieser Beweis kann ohne die Definition der Jordanschen Normalform gelingen, wenn beide Matrizen diagonalisierbar sind.

„Diagonalmatrizen“:

Aufgabenstellung:
Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt.
Hinweis: Die Eigenwerte von A, B liegen in $\mathbb{Z}$ und A, B sind diagonalisierbar.

Optionen:

Gegeben:	$A, B \in M(n \times n; \mathbb{Z})$, diagonalisierbar
Gesucht:	$P \in GL(n; \mathbb{Z})$
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B = P \circ A \circ P^{-1}$
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	13

Tabelle 3.9: „Diagonalmatrizen“

Aufgabenstellung 3.9

Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt. Hinweis: Die Eigenwerte von A, B liegen in $\mathbb{Z}$ und A, B sind diagonalisierbar.

Einstellung im Fenster: „Diagonalmatrizen“

Ähnlich der vorangegangenen Option wird zu Beginn eine Diagonalmatrix mit Einträgen aus $\mathbb{Z}$ erzeugt. Jetzt findet mein Programm $P_A, {P_A}^{-1}$, mit denen dann A gebaut wird, sodass A schon einmal ähnlich zu der Matrix in Diagonalgestalt ist. Erst dann wird B auf die bekannte Art aus A zusammengebaut, indem P, P^{-1} gefunden werden. A, B sind dann ähnlich zueinander und sie besitzen die gleiche Jordansche Normalform in Form einer Diagonalmatrix.
Auch diese Aufgabe lässt sich nicht direkt für OKUSON verwenden, weil P als Lösung nicht eindeutig ist, wie schon oben herausgearbeitet wurde, sodass die definierende Eigenschaft bei der Kontrolle nachgerechnet werden muss.

Das Programm gibt also die Möglichkeit, an verschiedenen Stellen die Eigenschaften von Eigenwerten, dem charakteristischen Polynom, Eigenvektoren und der Diagonalisierbarkeit abzufragen, wobei seine Algorithmen garantieren, dass es in allen Fällen ganzzahlige Lösungen für die Eigenwerte gibt, zu denen man bei den entsprechenden Aufgabentypen auch Eigenvektoren finden kann, deren Einträge nur aus ganzen Zahlen bestehen.

Das Prinzip der Aufgabenerstellung ist in allen Aufgabentypen bisher das gleiche. Man fängt mit einer Matrix in einer bestimmten Gestalt an, findet ein oder mehrere Paare über $\mathbb{Z}$ invertierbarer Matrizen und erzeugt damit richtige Lösungen. Bei falschen Lösungen sorgt man dafür, dass eine neue Ausgangsmatrix sich auf eine vorgegebene Weise von der Ausgangsmatrix der richtigen Lösungen unterscheidet, aber immer noch eine Jordansche Normalform ist, findet wieder ein oder mehrere Paare über $\mathbb{Z}$ inverser Matrizen und erzeugt damit falsche Lösungen. Dass dieses Verfahren funktioniert, wird durch die Eigenschaften der ähnlichen Matrizen garantiert. Genau dieses Vorgehen wird auch bei der Generierung der folgenden Aufgaben gewählt.

Bisher sind noch die folgenden Probleme ungeklärt: Findet das Programm alle möglichen Lösungen? Wie findet das Programm über $\mathbb{Z}$ invertierbare Matrizen? Beide Probleme hängen eng miteinander zusammen, weshalb auch die erste Fragestellung erst gegen Ende von Kapitel 4 endgültig beantwortet werden kann. An dieser Stelle sei nur Folgendes erwähnt:

Bemerkung 3.19

Aus dem Zusammenhang

$$A, B \in M(n \times n; \mathbb{Z}) \text{ ähnlich über } \mathbb{Z} \Leftrightarrow \exists P \in GL(n; \mathbb{Z}) \text{ mit } B = P \circ A \circ P^{-1}$$

folgt, dass man bei festem A alle B finden kann, die über $\mathbb{Z}$ ähnlich zu A sind, wenn man alle $P \in GL(n; \mathbb{Z})$ finden kann.

Das bedeutet, dass mein Programm in der Lage sein muss, alle $P \in GL(n; \mathbb{Z})$ zu finden, um auch alle Paare von über $\mathbb{Z}$ ähnlichen Matrizen finden zu können.

Aufgaben zu Jordanschen Normalformen

Auch die Erstellung der folgenden Aufgaben funktioniert nach dem gleichen Prinzip.

Aufgabenstellung 3.10

Zeigen Sie, dass die Matrix $B \in M(n \times n; \mathbb{Z})$ nilpotent ist. Geben Sie den Nilpotenzgrad an.

Einstellung im Fenster: „Nilpotenzgrad bestimmen“

Das Programm beginnt mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform und setzt in dieser alle Einträge auf der Diagonalen gleich Null.

Allerdings muss $A = 0$ ausgeschlossen werden, da der Fall sonst relativ häufig bei der Aufgabenerstellung auftritt. Über die Größe der Jordanblöcke in dieser neuen Jordanschen Normalform ist das Minimalpolynom und damit der Nilpotenzgrad schon festgelegt. Das Programm erzeugt nun auf die oben angesprochene Weise eine Matrix B, die über $\mathbb{Z}$ ähnlich zu A ist. Das Minimalpolynom von B ist das gleiche wie das von A, weil ähnliche Matrizen das gleiche Minimalpolynom besitzen; also stimmt auch der Nilpotenzgrad überein. Ein Abgleich kann dann zur Kontrolle der Lösungen dienen.
Der Nilpotenzgrad ist eindeutig, weil das Minimalpolynom eindeutig ist. Also ist hier auch die Lösung eindeutig und die Aufgabe ist an sich mit OKUSON kompatibel. Leider wird bei ihrer Erstellung innerhalb meines Programms auf Routinen zurückgegriffen, die auch Aufgaben erstellen sollen, bei denen das nicht der Fall ist, weil die eindeutige Lösung in diesen Aufgaben eine komplexe Eingabe erfordern würde. Das Problem wird dort gelöst, indem man zwischen der richtigen und einer falschen Lösung wählen kann, sodass keine solche Angabe mehr gemacht werden muss. Deshalb erstellt das Programm eine Ausgangsmatrix für eine falsche Lösung, indem es A wie in Aufgabenstellung 3.13 so verändert, dass sich das Minimalpolynom ändert. Dabei kann der Fall auftreten, dass die neue Jordansche Normalform die Nullmatrix ist. In diesem Fall setzt mein Programm den ersten Eintrag auf der Hauptdiagonalen der Nullmatrix gleich 1 und erzeugt damit eine nicht nilpotente Matrix als Ausgangsmatrix für das Auslesen der falschen Lösung. Die Ausgabe enthält dann einen Strich, um zu zeigen, dass die Matrix nicht nilpotent sein soll. Die schöne Alternative, bei der einfach alle möglichen Nilpotenzgrade aufgezählt werden, scheitert wieder an den anderen Aufgabenstellungen, die die gleichen Routinen nutzen. Wie bei den angesprochenen Aufgaben ist das Erzeugen weiterer falscher Lösungen auch an dieser Stelle sinnlos, da dabei immer nur die gleiche Ausgabe produziert wird.

Aufgabenstellung 3.11

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben den folgenden Nilpotenzgrad?

Einstellung im Fenster: „Ausschluss über Nilpotenzgrad“
Das Programm arbeitet auf die gleiche Weise wie in der obigen Aufgabenstellung, um Ausgangsmatrizen für richtige und falsche Lösungen zu finden. Hier kann man mehrere richtige und falsche Lösungen erzeugen, weil man einfach ähnliche Matrizen zu den Ausgangsmatrizen bildet, die damit das gleiche Minimalpolynom, also auch den gleichen Nilpotenzgrad haben. Damit sind die Lösungen eindeutig und die Aufgabe ist mit OKUSON kompatibel, weil ein simpler Abgleich möglich ist.

„Nilpotenzgrad bestimmen“:

Aufgabenstellung:
Zeigen Sie, dass die Matrix $B \in M(n \times n; \mathbb{Z})$ nilpotent ist. Geben Sie den Nilpotenzgrad an.

Optionen:

Gegeben:	Matrix $B \in M(n \times n; \mathbb{Z})$
Gesucht:	$k \in \mathbb{N}$ mit $B^k = 0, B^{k-1} \neq 0$
Variabel:	n
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	mit Einschränkungen möglich
Parameteraufruf mit:	34

Tabelle 3.10: „Nilpotenzgrad bestimmen“

Analog zu den Aufgaben zum charakteristischen Polynom kann man die folgenden Aufgaben zum Minimalpolynom stellen.

Aufgabenstellung 3.12

Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ das Minimalpolynom an.

Einstellung im Fenster: „Minimalpolynom bestimmen“

Das Programm beginnt mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform. Über die Größe der Jordanblöcke in dieser Jordanschen Normalform ist das Minimalpolynom schon festgelegt. Das Programm erzeugt nun auf die oben angesprochene Weise eine Matrix B, die über $\mathbb{Z}$ ähnlich zu A ist. Das Minimalpolynom von B ist das gleiche wie das von A, weil ähnliche Matrizen das gleiche Minimalpolynom besitzen.
Die Lösung ist damit eindeutig und eigentlich wäre die Aufgabe ohne weiteres mit OKUSON kompatibel, wenn da nicht die komplexe Eingabe eines Polynoms wäre. Wie schon beim charakteristischen Polynom wird das Problem mit der Angabe einer richtigen und einer falschen Lösung umgangen. Und auch hier kann man nur eine falsche Lösung bauen, weil der Algorithmus aus Aufgabenstellung 3.13, der hier schon benutzt wird, nicht flexibel genug ist und das Minimalpolynom nicht ausgerechnet, sondern lediglich aus der Jordanschen Normalform ausgelesen wird.

Aufgabenstellung 3.13

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben das folgende Minimalpolynom?

„Ausschluss über Nilpotenzgrad“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben den folgenden Nilpotenzgrad?

Optionen:

Gegeben:	$k \in \mathbb{N}$ und Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen mit $B^k = 0, B^{k-1} \neq 0$
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	210

Tabelle 3.11: „Ausschluss über Nilpotenzgrad“

„Minimalpolynom bestimmen“:

Aufgabenstellung:
Geben Sie zu der Matrix $B \in M(n \times n; \mathbb{Z})$ das Minimalpolynom an.

Optionen:

Gegeben:	Matrix $B \in M(n \times n; \mathbb{Z})$
Gesucht:	m_B
Variabel:	n
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	mit Einschränkungen möglich
Parameteraufruf mit:	33

Tabelle 3.12: „Minimalpolynom bestimmen“

Einstellung im Fenster: „Ausschluss über Minimalpolynom“

Das Programm beginnt mit einer Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform. Über die Größe der Jordanblöcke in dieser Jordanschen Normalform ist das Minimalpolynom schon festgelegt. Es erzeugt nun auf gewohnte Weise eine Matrix B, die über $\mathbb{Z}$ ähnlich zu A ist. Das Minimalpolynom von B ist das gleiche wie das von A, weil ähnliche Matrizen das gleiche Minimalpolynom besitzen. Also kann man auf diese Weise richtige Lösungen finden. Das Minimalpolynom ist über die Größe der Jordanblöcke festgelegt. Der Exponent eines Linearfaktors im Minimalpolynom ist die Größe des größten Jordanblockes zu diesem im Linearfaktor festgelegten Eigenwert. Mein Programm ordnet bei der Erstellung einer Matrix in Jordanscher Normalform die größten Blöcke oben an. Der Einfachheit halber wird die Größe dieser größten Blöcke um 1 so verringert, dass der letzte Eigenwert seinen eigenen Block bildet. Damit erzeugt man eine andere Matrix in Jordanscher Normalform, die auf die bekannte Weise die Grundlage der falschen Lösungen bildet. Dieses Prinzip funktioniert immer, es sei denn, der erste Block hat schon die Größe 1. In diesem Fall werden die ersten beiden Eigenwerte zu einem Block verschmolzen, wobei der zweite Diagonaleintrag durch den ersten ersetzt wird.

Hier sind einfache Beispiele dargestellt, die zeigen, wie das Programm reagiert:

$$\begin{pmatrix} \lambda_1 & 1 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 1 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix} \rightarrow \begin{pmatrix} \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix}$$

$$\begin{pmatrix} \lambda_1 & 1 & 0 & 0 & 0 \\ 0 & \lambda_1 & 1 & 0 & 0 \\ 0 & 0 & \lambda_1 & 0 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix} \rightarrow \begin{pmatrix} \lambda_1 & 1 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_1 & 0 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix}$$

bzw.

$$\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \rightarrow \begin{pmatrix} \lambda_1 & 1 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

Auf diese Weise baut das Programm eine Jordansche Normalform als Grundlage für die falschen Lösungen, die, wie erkennbar ist, ein anderes Minimalpolynom besitzt, weil ein Linearfaktor fehlt und/oder der Exponent eines oder mehrerer Linearfaktoren im Minimalpolynom verändert worden ist. Die

„Ausschluss über Minimalpolynom“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben das folgende Minimalpolynom?

Optionen:

Gegeben:	m_A und Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen mit $m_B = m_A$
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	29

Tabelle 3.13: „Ausschluss über Minimalpolynom“

Lösung ist aufgrund der weiter oben dargestellten Eigenschaften eindeutig und die Aufgabe damit mit OKUSON kompatibel, weil die Kontrolle über einen einfachen Abgleich erfolgen kann.

Eigentlich hätten an dieser Stelle nun Aufgaben zu Haupträumen stehen können. Doch das Problem solcher Aufgaben liegt darin, dass man hier Basen eines Vektorraums bestimmen soll, die nicht immer eindeutig sind. Mit dieser Begründung fallen solche Aufgaben heraus, was ihr kurzes Erwähnen in der hier gewählten Darstellung erklärt. Allerdings benötigt man im Allgemeinen die Zerlegung in Haupträume, um zu einer Matrix die zugehörige Jordansche Normalform berechnen zu können, was zu den folgenden Aufgaben führt.

Aufgabenstellung 3.14

Geben Sie zu der folgenden Matrix B die zugehörige Jordansche Normalform A und eine Matrix $P \in GL(n; \mathbb{C})$ mit $B = P \circ A \circ P^{-1}$ an.

Einstellung im Fenster: „Jordanform bestimmen“

Bei dieser Einstellung soll zu einer gegebenen Matrix die Jordansche Normalform erzeugt werden. Das Programm arbeitet wie gewohnt umgekehrt. Es fängt mit einer beliebigen Matrix $A \in M(n \times n; \mathbb{Z})$ in Jordanscher Normalform an und erstellt zu dieser dann eine ähnliche Matrix $B \in M(n \times n; \mathbb{Z})$. Weder A noch P sind eindeutig, denn die Jordanblöcke in A können beliebig angeordnet werden, während P als konjugierende Matrix uneindeutig ist. Damit fällt eine solche Aufgabe für OKUSON aus, weil OKUSON nicht in der Lage ist, die definierende Eigenschaft $B = P \circ A \circ P^{-1}$ zu überprüfen. Für OKUSON bietet sich die folgende Alternative an.

„Jordanform bestimmen“:

Aufgabenstellung:
Geben Sie zu der folgenden Matrix B die zugehörige Jordansche Normalform A und eine Matrix $P \in GL(n;\mathbb{C})$ mit $B = P \circ A \circ P^{-1}$ an.

Optionen:

Gegeben:	Matrix $B \in M(n \times n;\mathbb{Z})$
Gesucht:	Jordanform $A \in M(n \times n;\mathbb{Z})$ zu B, $P \in GL(n;\mathbb{Z})$
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B = P \circ A \circ P^{-1}$
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	43

Tabelle 3.14: „Jordanform bestimmen“

Aufgabenstellung 3.15

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n;\mathbb{Z})$ haben die folgende Jordansche Normalform A?

Einstellung im Fenster: „Ausschluss über Jordanform“

Bei dieser Option sollen Matrizen bestimmt werden, die die angegebene Jordansche Normalform besitzen. Die richtigen Lösungen sind zu der angegebenen Form ähnlich und werden auf die gewohnte Weise erzeugt. Die falschen Lösungen werden mit einer anderen Jordanschen Normalform als Grundlage zusammengesetzt, sodass ausgeschlossen wird, dass nicht doch eine dieser falschen Lösungen richtig ist. Entscheidend für Unterschiede bei Jordanschen Normalformen sind die Eigenwerte und die Größe der zugehörigen Jordanblöcke. Die Größe dieser Jordanblöcke ist über die Einsen und Nullen rechts neben bzw. über den Eigenwerten festgelegt. Diese Einsen werden nun per Zufall entweder auf Eins oder Null gesetzt, um eine andere Jordansche Normalform als Grundlage für die falschen Lösungen zusammenzubauen. Dieses Prinzip funktioniert genau dann, wenn man garantieren kann, dass der Fall, dass die gleiche Jordansche Normalform dabei wieder entsteht, ausgeschlossen wird. Dies geschieht, indem eine Eins in der ersten Zeile neben dem Eigenwert immer zur Null wird, während eine Null an dieser Stelle zur Eins wird, wobei der Eigenwert in Zeile 2 dann den Wert des Eigenwertes in Zeile 1 annimmt. Da der größte Block bei der Erstellung einer Jordanschen Normalform durch mein Programm immer oben steht, dieser schon wegen der Null in Zeile 1 die Größe 1 hatte, sind keine weiteren Änderungen mehr notwendig.

„Ausschluss über Jordanform“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ haben die folgende Jordansche Normalform A?

Optionen:

Gegeben:	Matrizen $A, B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen B, die Jordansche Normalform A besitzen
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	28

Tabelle 3.15: „Ausschluss über Jordanform“

Hier sind einfache Beispiele dargestellt, die zeigen, wie das Programm reagiert:

$$\begin{pmatrix} \lambda_1 & 1 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 1 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix} \rightarrow \begin{pmatrix} \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 1 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix}$$

bzw.

$$\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \rightarrow \begin{pmatrix} \lambda_1 & 1 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

Auf diese Weise baut das Programm eine Jordansche Normalform als Grundlage für die falschen Lösungen, die dann auf die bekannte Weise erzeugt werden. Im Gegensatz zur vorangegangenen Aufgabenstellung muss man jetzt nur noch entscheiden, ob eine gegebene Matrix die gegebene Jordansche Normalform besitzt oder nicht, was heißt, dass die Lösung hier eindeutig und die Aufgabe damit kompatibel mit OKUSON ist.

An dieser Stelle gibt das Programm also die Möglichkeit, die einzelnen Schritte bei der Einführung der Jordanschen Normalform zu unterstützen, indem es Aufgaben direkt zu den weiteren Voraussetzungen zur Verfügung stellt. Auch für diese Aufgaben bleiben die Problemstellungen des letzten Unterabschnittes erhalten. Kann man auf diese Weise alle Aufgaben finden? Und wie findet man über $\mathbb{Z}$ invertierbare Matrizen?

Aufgaben direkt zur Ähnlichkeit von Matrizen

Zunächst kann man als Aufgabe geben, die Gültigkeit der definierenden Eigenschaft von ähnlichen Matrizen abzufragen, wobei die Lösung aus der Angabe einer Matrix P besteht, sodass $B = P \circ A \circ P^{-1}$ gilt.

Aufgabenstellung 3.16

Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt.

Einstellung im Fenster: „keine Optionen“
Es wird eine zufällige Matrix $A \in M(n \times n; \mathbb{Z})$ gewählt, zu der auf die übliche Weise eine über $\mathbb{Z}$ ähnliche Matrix B gebaut wird. Das klingt einfach, ist es aber nicht, denn bei einer solchen Aufgabenstellung gibt es nämlich Schwierigkeiten. Die Angabe der Matrix P ist nicht eindeutig wie Satz 3.21 und diesbezüglich die nachfolgende Bemerkung 3.22 zeigen werden. Es existieren in einigen Fällen sogar unendlich viele Varianten, was ausschließt, dass die Aufgabe mit OKUSON kompatibel ist, weil man zur Kontrolle die definierende Eigenschaft überprüfen muss.
Das Problem einer solchen Aufgabe ist zudem die Durchführung der Rechnung, wie das folgende Beispiel zeigt:

Beispiel 3.20

$$A = \begin{pmatrix} 0 & -2 & 0 \\ -2 & -2 & 3 \\ 2 & 0 & -1 \end{pmatrix}, B = \begin{pmatrix} -1 & 2 & 0 \\ 0 & 0 & -2 \\ 3 & -2 & -2 \end{pmatrix}$$

Diese beiden Matrizen sind ähnlich über $\mathbb{Z}$, denn es gilt:

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 0 & -2 & 0 \\ -2 & -2 & 3 \\ 2 & 0 & -1 \end{pmatrix} \circ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -1 & 2 & 0 \\ 0 & 0 & -2 \\ 3 & -2 & -2 \end{pmatrix}$$

$$\text{mit: } \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Die Jordansche Normalform beider Matrizen ist jedoch

$$\begin{pmatrix} -4 & 0 & 0 \\ 0 & \frac{1}{2} + \frac{1}{2} \cdot i \cdot \sqrt{7} & 0 \\ 0 & 0 & \frac{1}{2} + \frac{1}{2} \cdot i \cdot \sqrt{7} \end{pmatrix},$$

die Einträge aus $\mathbb{C}$ besitzt.

„keine Optionen“:

Aufgabenstellung:
Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt.

Optionen:

Gegeben:	Matrizen $A, B \in M(n \times n; \mathbb{Z})$
Gesucht:	$P \in GL(n; \mathbb{Z})$ mit $B = P \circ A \circ P^{-1}$
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B = P \circ A \circ P^{-1}$
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	11

Tabelle 3.16: „keine Optionen“

Die Lösung einer solchen Aufgabe kann im Allgemeinen nur dann gelingen, wenn man die Jordansche Normalform bestimmt und Matrizen findet, sodass A und B ähnlich zu der gleichen Jordanschen Normalform sind. Das Rechenprinzip in solchen Aufgaben mag zwar immer das gleiche sein, allerdings kann es sein, dass es von Hand nur schwer nachprüfbar ist, ob zwei Matrizen zueinander ähnlich sind, weil die Wurzeln, mit denen gearbeitet werden muss, einfach zu verschachtelt sind.

Vielleicht sieht man schon anhand dieses Beispiels ein, dass solche Aufgaben schwer zu handhaben sind, weshalb ich über Jordansche Normalformen eine Sicherung eingebaut habe. Die obige Aufgabenstellung wird dadurch zu verändert, dass man die Eigenwerte der ähnlichen Matrizen aus $\mathbb{Z}$ wählt. Diese Wahl wurde schließlich auch bisher für alle Aufgabenstellungen aus dem Bereich „Ähnliche Matrizen“ getroffen. Es ergibt sich die folgende Variante:

Aufgabenstellung 3.17

Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt. Hinweis: Die Eigenwerte von A, B liegen in $\mathbb{Z}$.

Einstellung im Fenster: „ganzzahlige Eigenwerte“

Anstatt eine zufällige Matrix mit Einträgen aus $\mathbb{Z}$ zu wählen, wird hier nur eine zufällige Matrix mit Einträgen aus $\mathbb{Z}$ in Jordanscher Normalform generiert, aus der dann A gebaut wird, sodass A ähnlich zu der Matrix in Jordanscher Normalform ist. Erst dann wird B auf die bekannte Art zusammengebaut. A, B sind dann ähnlich, weil sie die gleiche Jordansche Normalform besitzen.

„ganzzahlige Eigenwerte“:

Aufgabenstellung:
Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt.
Hinweis: Die Eigenwerte von A, B liegen in $\mathbb{Z}$.

Optionen:

Gegeben:	Matrizen $A, B \in M(n \times n; \mathbb{Z})$
Gesucht:	$P \in GL(n; \mathbb{Z})$ mit $B = P \circ A \circ P^{-1}$
Variabel:	n
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$B = P \circ A \circ P^{-1}$
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	12

Tabelle 3.17: „ganzzahlige Eigenwerte“

Allerdings gelten bezüglich der Eindeutigkeit der Lösung und der Verwendung mit OKUSON immer noch die gleichen Einschränkungen wie in der vorangegangenen Aufgabenstellung.

Alternativ zum direkten Beweis kann man auch Aufgaben erstellen, die abfragen, ob zwei oder mehr Matrizen ähnlich sind. Die Eigenschaften der ähnlichen Matrizen machen es möglich, dass man an verschiedenen Stellen bei der Bearbeitung der Aufgabe auf Widersprüche stoßen kann, wie im Nachfolgenden dargestellt wird, sodass der Beweis lediglich für die richtigen Lösungen geführt werden muss.

Aufgabenstellung 3.18

Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind ähnlich zur Matrix $A \in M(n \times n; \mathbb{Z})$?

Im Programm kann bei dieser Aufgabenstellung ausgewählt werden, an welcher Stelle bei der Überprüfung der Fehler spätestens auftreten soll, bei der Berechnung der Eigenwerte (Einstellung im Fenster: „falsche Eigenwerte“), der charakteristischen Polynome (Einstellung im Fenster: „falsches char. Polynom“) , der Minimalpolynome (Einstellung im Fenster: „falsches Min.-Polynom“) oder der Angabe der Jordanschen Normalformen (Einstellung im Fenster: „falsche Jordanform“).
Dabei garantieren die Eigenschaften von ähnlichen Matrizen, dass die Lösungen stets eindeutig sind, was heißt, dass diese Aufgabenstellungen auch

„falsche Eigenwerte“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind ähnlich zur Matrix $A \in M(n \times n; \mathbb{Z})$?

Optionen:

Gegeben:	Matrizen $A, B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen B, die ähnlich zu A sind
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	21

Tabelle 3.18: „falsche Eigenwerte“

mit OKUSON kompatibel sind, da man hier zur Kontrolle einfach nur einen Abgleich durchführen muss.

Zwei Matrizen A, B sind genau dann nicht ähnlich, wenn sie nicht die gleiche Jordansche Normalform besitzen. Wieder wird mit einer Matrix in Jordanscher Normalform mit Einträgen aus $\mathbb{Z}$ begonnen, aus der auf die bekannte Weise A und die richtigen Lösungen erzeugt werden.

Einstellung im Fenster: „falsche Eigenwerte“

Falsche Lösungen gewinnt man hier auf die gleiche Weise wie in Aufgabenstellung 3.4.

Einstellung im Fenster: „falsches char. Polynom“

Falsche Lösungen gewinnt man bei dieser Einstellung auf die gleiche Weise wie in Aufgabenstellung 3.5.

Einstellung im Fenster: „falsches Minimalpolynom“

Hier gewinnt man falsche Lösungen auf die gleiche Weise wie in Aufgabenstellung 3.13.

Einstellung im Fenster: „falsche Jordanform“

Bei dieser Einstellung werden falsche Lösungen auf die gleiche Weise wie in Aufgabenstellung 3.15 gewonnen.

Nachdem nun alle Aufgabenstellungen zum Bereich der ähnlichen Matrizen vorgestellt worden sind, möchte ich noch eine Anmerkung bezüglich der Grundlage der falschen und der richtigen Lösungen machen. Wie gesehen, ist

„falsches char. Polynom“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind ähnlich zur Matrix $A \in M(n \times n; \mathbb{Z})$?

Optionen:

Gegeben:	Matrizen $A, B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen B, die ähnlich zu A sind
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	22

Tabelle 3.19: „falsches char. Polynom“

„falsches Minimalpolynom“:

Aufgabenstellung:
Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind ähnlich zur Matrix $A \in M(n \times n; \mathbb{Z})$?

Optionen:

Gegeben:	Matrizen $A, B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen B, die ähnlich zu A sind
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	23

Tabelle 3.20: „falsches Minimalpolynom“

„falsche Jordanform“:	
Aufgabenstellung: Welche der folgenden Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$ sind ähnlich zur Matrix $A \in M(n \times n; \mathbb{Z})$?	
Optionen:	
Gegeben:	Matrizen $A, B_1, B_2, ... \in M(n \times n; \mathbb{Z})$
Gesucht:	Matrizen B, die ähnlich zu A sind
Variabel:	n, Anzahl der richtigen und falschen Lösungen
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	24

Tabelle 3.21: „falsche Jordanform“

es so, dass die Grundlage der richtigen Lösungen immer die gleiche Jordansche Normalform ist. Das heißt auch, dass Eigenwerte, charakteristisches Polynom und Minimalpolynom der richtigen Lösungen identisch sind. Dies soll natürlich auch so sein. Problematisch wird es bei den falschen Lösungen, wo meine Algorithmen bei der Erstellung teilweise immer die gleiche Jordansche Normalform auf der Grundlage der Jordanschen Normalform der richtigen Lösungen erzeugen. Das heißt, dass bei einigen Aufgaben, nämlich jenen, bei denen kein zufälliger neuer Eigenwert in die Matrix geschrieben worden ist, Eigenwerte, charakteristisches Polynom und Minimalpolynom der falschen Lösungen ebenfalls übereinstimmen. Sobald die Lernenden dieses Prinzip erst einmal entdeckt haben, könnten Sie es ausnutzen, um schneller eine für sie glaubwürdige Lösung zu finden, die dann am Ende aber nur zu Verwirrung oder gar zu falschen Ergebnissen führt, wenn dieses Prinzip einmal versagt. Aus diesem Grund rate ich dazu, nicht zu viele falsche Lösungen in den Aufgaben vorzugeben.

3.1.5 Die Uneindeutigkeit der konjugierenden Matrix

Es bleibt Frage, ob die Matrix $P \in GL(n; \mathbb{Z})$, die in allen Aufgabenstellungen verwendet und in manchen auch als Lösung eingefordert wird, eindeutig ist. Ich habe schon in der Einleitung mit Beispiel 2.2 gezeigt, dass es einen Fall für $n = 2$ gibt, in dem sie es nicht ist.

Satz 3.21

Seien $A, B \in M(n \times n; K)$, K ein Körper, $n \geq 2$, und gelte $B = P \circ A \circ P^{-1}$

mit $P \in GL(n;K)$, dann existiert mindestens ein Spezialfall von A, B, für den es unendlich viele verschiedene P gibt.

Beweis: Man betrachte Beispiel 2.2 und verallgemeinere es für quadratische Matrizen beliebiger Größe. □

Bemerkung 3.22

Seien $A, B \in M(n \times n; R)$, R ein Ring, $n \geq 2$, und gelte $B = P \circ A \circ P^{-1}$ für ein $P \in GL(n;R)$, dann gibt es mindestens so viele Matrizen $P' \in GL(n;K)$, für die $B = P' \circ A \circ P'^{-1}$ gilt, wie R Einheiten hat, denn sei e eine Einheit von R, dann erfüllt nach den Rechenregeln für Matrizen $P' = e \cdot P$ die definierende Eigenschaft.[29]

Das ist natürlich nicht wirklich zufriedenstellend, denn Aufgaben, die die konjugierende Matrix P als Lösung haben, erfordern damit eine rechnerische Überprüfung der definierenden Eigenschaft. Eine automatische Überprüfung mit OKUSON ist ohne eine solche Routine somit ausgeschlossen, weshalb viele Aufgaben nur über Umwege oder gar nicht mit OKUSON kompatibel sind. Eine solche Routine zu entwickeln, würde den Umfang dieser Arbeit sprengen, wäre meiner Meinung nach aber der nächste Schritt in der Weiterentwicklung von OKUSON.

3.2 Lineare Gleichungssysteme

Mit den nun folgenden Aufgabenstellungen wird der sonst übliche Programmablauf etwas geändert. Gehörten alle bisherigen Aufgabenstellungen mehr oder minder in den Bereich der ähnlichen Matrizen, den ich nur der Übersichtlichkeit halber aufgespalten habe, so ist dies nun nicht mehr der Fall. Der Algorithmus zur Aufgabenerstellung ist ein anderer und muss erst noch vorgestellt werden.

3.2.1 Definition und wichtige Eigenschaften

Definition 3.23

Die Zeilen bzw. Spalten einer Matrix $A \in M(m \times n; K)$ erzeugen jeweils einen Untervektorraum des K^n bzw. des K^m. Die Dimensionen[30] dieser Räume heißen der Zeilen- bzw. der Spaltenrang von A.
Es gilt zudem: Zeilenrang von A = Spaltenrang von $A =: \operatorname{rang}(A)$ [31]

[29] Zum Begriff „Einheit“ siehe Abschnitt 4.1.
[30] Zu den Begriffen „Basis“ und „Dimension“ siehe [Fischer, 2002, Seite 86 ff.].
[31] Vergleiche auch [Wille, 2001, Seite 80].

Beweis: Ein Beweis findet sich in [Beutelspacher, 2001, Seite 101 ff.].

Definition 3.24

(i) Ein lineares Gleichungssystem (LGS) hat das folgende Aussehen:

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &=& b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &=& b_2 \\ \vdots \qquad\qquad\qquad\qquad \vdots &\quad& \vdots \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n &=& b_m \end{array}$$

Dieses System kann man mit Hilfe der Matrix $A = (a_{ij}) \in M(m \times n; K)$ auch in der Form $A \circ \vec{x} = \vec{b}$ schreiben, wobei $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$ gilt. A nennt sich dann die Koeffizientenmatrix des linearen Gleichungssystems.
Erweitert man die Matrix A um eine Spalte, in die in der richtigen Reihenfolge die Einträge von $\vec{b}$ geschrieben werden, so ergibt sich die Matrix $\left(A|\vec{b}\right)$, die erweiterte Matrix des linearen Gleichungssystems.

(ii) Ein lineares Gleichungssystem $A \circ \vec{x} = \vec{b}$ heißt

(a) homogen $\Leftrightarrow \vec{b} = \vec{0}$,

(b) inhomogen $\Leftrightarrow \vec{b} \neq \vec{0}$,

(c) quadratisch $\Leftrightarrow m = n$.

[32]

Eine der wichtigsten Eigenschaften für lineare Gleichungssysteme ist das Lösbarkeitskriterium:

Satz 3.25

(i) Ein lineares Gleichungssystem $A \circ \vec{x} = \vec{b}$ ist genau dann lösbar, wenn der Rang der Koeffizientenmatrix A gleich dem Rang der erweiterten Matrix $\left(A|\vec{b}\right)$ ist. Die Lösungsmenge ist dann ein affiner Unterraum der Dimension $n - \text{rang}(A)$.

(ii) Ein quadratisches lineares Gleichungssystem $A \circ \vec{x} = \vec{b}$ ist genau dann eindeutig lösbar, wenn $A \in GL(n; K)$, also $\text{rang}(A) = n$ gilt.

[33]

[32] Vergleiche auch [Wille, 2001, Seite 91].
[33] Vergleiche auch [Wille, 2001, Seite 91].

Beweis:

(i) Ein Beweis findet sich in [Beutelspacher, 2001, Seite 101 und Seite 138].

(ii) Ein Beweis findet sich in [Beutelspacher, 2001, Seite 105 f.].

Definition 3.26
Seien $A \in M(m \times n; K)$, K ein Körper, $\vec{x} \in V \subset K^n$ ein Spaltenvektor, V ein Vektorraum.

(i) Der Kern der Matrix A ist die Menge der Vektoren $\vec{x} \in V$, die von A auf den Nullvektor abgebildet werden.

$$\ker(A) = \{\vec{x} \in V | A \circ \vec{x} = \vec{0}\}$$

(ii) Das Bild der Matrix A ist die Menge der Vektoren, die bei der Abbildung von V mit A entstehen.

$$\text{bild}(A) = A \circ V$$

Bemerkung 3.27
Sowohl beim Kern als auch beim Bild handelt es sich um Untervektorräume. $\ker(A)$ ist ein Untervektorraum des K^n, $\text{bild}(A)$ ein Untervektorraum des K^m. Demnach lässt sich zu beiden Untervektorräumen eine Basis finden und es gilt der folgende Satz:

Satz 3.28 (Kern-Bild-Satz)
Seien $A \in M(m \times n; K)$, K ein Körper, $V \subset K^n$ ein Vektorraum, dann gilt:

$$\dim(V) = \dim(\text{bild}(A)) + \dim(\ker(A)).$$

Beweis: Ein Beweis für lineare Abbildungen, der sich auf Matrizen übertragen lässt, findet sich in [Fischer, 2002, Seite 117 f.].

Wichtig ist der Zusammenhang zwischen dem Rang und der Invertierbarkeit von quadratischen Matrizen:

Satz 3.29
Für eine Matrix $A \in M(n \times n; K)$, K ein Körper, sind die folgenden Aussagen äquivalent:

(i) A ist invertierbar.

(ii) A^T *ist invertierbar.*

(iii) $\text{rang}(A) = 0$.

[34]

Beweis: Ein Beweis findet sich in [Fischer, 2002, Seite 150].

Wichtige Lernziele aus diesem Bereich, zu denen mein Programm Aufgaben erstellt, die das Erreichen eben dieser überprüfen können, sind die folgenden:

Lernziel 3.11

Die Studenten sollen den Begriff des linearen Gleichungssystems kennen, um seine Eigenschaften (vor allem in Bezug auf Matrizen) wissen, zeigen können, wann ein lineares Gleichungssystem lösbar ist, es lösen und die Eigenschaften dieses Begriffes in verschiedenen Aufgaben ausnutzen können.

Lernziel 3.12

Die Studenten sollen den Begriff des Rangs von Matrizen kennen, um seine Eigenschaften (vor allem in Bezug auf die Invertierbarkeit von Matrizen und die Dimension des Kerns und des Bildes) wissen, zeigen können, welchen Rang eine beliebige Matrix hat, und die Eigenschaften dieses Begriffes in verschiedenen Aufgaben ausnutzen können.

Lernziel 3.13

Die Studenten sollen die Begriffe des Kerns und des Bildes von Matrizen kennen, um ihre Eigenschaften (vor allem in Bezug auf lineare Abbildungen) wissen, die Dimension und eine Basis vom Kern und vom Bild einer Matrix berechnen und die Eigenschaften dieser Begriffe in verschiedenen Aufgaben ausnutzen können.

3.2.2 Programmablauf für LGS-Aufgaben

Ein Teil des Algorithmus, der schon bei den anderen Aufgabenstellungen hilft, kommt auch bei den linearen Gleichungssystemen zum Einsatz. Allerdings besteht nun die Schwierigkeit darin, dass lineare Gleichungssysteme im Allgemeinen keine quadratische Koeffizientenmatrix besitzen. Und da die oben aufgeführten Aufgabenstellungen alle auf quadratische Matrizen zurückgreifen, bleibt nichts anderes übrig, als einen komplett neuen Algorithmus zum Erstellen von Aufgaben zu schreiben, der allerdings auf den früheren Erkenntnissen aufbaut. Der Algorithmus soll Folgendes leisten:

[34] Vergleiche auch [Fischer, 2002, Seite 150].

1. Er soll eine Matrix $A \in M(m \times n; \mathbb{Z})$ erstellen, die eine vorgegebene Zeilenanzahl m, eine vorgegebene Spaltenanzahl n und einen vorgegebenen Rang r besitzt.

2. Er soll eine Basis des Kerns mit Basisvektoren aus $\mathbb{Z}^n$ und des Bildes mit Basisvektoren $\mathbb{Z}^m$ der durch diese Matrix A festgelegten linearen Abbildung liefern.

3. Er soll zu einer vorgegebenen speziellen Lösung $\vec{x}_s \in \mathbb{Z}^n$ einen Vektor $\vec{b} \in \mathbb{Z}^m$ liefern, sodass $A \circ \vec{x}_s = \vec{b}$ gilt.

4. Er soll die Gleichung $A \circ \vec{x} = \vec{b}$ so verändern, dass sie in Bezug auf ihre Lösung keine zu triviale Gestalt hat, wobei der Rang von A nicht verändert werden darf und gewährleistet ist, dass keine der Forderungen bezüglich der Definition über $\mathbb{Z}$ in den vorangegangenen Punkten verletzt wird.

Einen solchen Algorithmus kann man mit Hilfe der folgenden Überlegungen gewinnen. Es gelte:

$$\tilde{A} \circ \vec{\xi} = \vec{\beta} \text{ mit } \tilde{A} \in M(m \times n; \mathbb{Z}),\ \xi \in \mathbb{Z}^n,\ \beta \in \mathbb{Z}^m$$

$\vec{\xi}$ ist im Allgemeinen nicht eindeutig festgelegt, denn man kann immer ein Element des Kerns von $\tilde{A}$ hinzuaddieren.

$$\begin{aligned} &\Leftrightarrow \tilde{A} \circ \vec{\xi} + \vec{0} = \vec{\beta} \\ &\Leftrightarrow \tilde{A} \circ \vec{\xi} + \tilde{A} \circ \vec{\kappa} = \vec{\beta} \text{ mit } \vec{\kappa} \in \ker(\tilde{A}) \\ &\Leftrightarrow \tilde{A} \circ \left(\vec{\xi} + \vec{\kappa}\right) = \vec{\beta} \end{aligned}$$

Einfacher, als dem Programm beizubringen, eine Matrix auf Zeilenstufenform zu bringen, ist es, eine Matrix in einer solchen Form durch elementare Zeilen- und Spaltenumformungen so zu verändern, dass keine Zeilenstufenform mehr vorliegt.

$$\begin{aligned} &\Leftrightarrow \tilde{A} \circ Q \circ \underbrace{Q^{-1} \circ \left(\vec{\xi} + \vec{\kappa}\right)}_{=:\vec{x}} = \vec{\beta} \text{ mit } Q \in GL(n; \mathbb{Z}) \\ &\Leftrightarrow \underbrace{P \circ \tilde{A} \circ Q}_{=:A} \circ \vec{x} = \underbrace{P \circ \vec{\beta}}_{=:\vec{b}} \text{ mit } P \in GL(m; \mathbb{Z}) \\ &\Leftrightarrow A \circ \vec{x} = \vec{b} \end{aligned}$$

Man erhält also eine Matrix A mit dem gleichen Rang wie $\tilde{A}$, zu der sich die Basisvektoren $\vec{k}$ des Kerns von A aus der Formel $\vec{k} = Q^{-1} \circ \vec{\kappa}$ berechnen lassen.

Wenn $\vec{\nu}$ ein Basisvektor des Bildes von $\tilde{A}$ ist, dann ist demzufolge $\vec{v} = P \circ \vec{\nu}$ ein Basisvektor des Bildes von A. Aus $\tilde{A} \circ \vec{\xi} = \vec{\beta}$ folgt mit $\vec{x} = Q^{-1} \circ \vec{\xi}$ und $P \circ \vec{\beta} = \vec{b}$ dann $A \circ \vec{x} = \vec{b}$.

Die Matrizen P, Q findet man mit dem alten Algorithmus, der schon im vorangegangenen Abschnitt benutzt worden ist. Bleibt also nur noch die Frage, wie die Matrix $\tilde{A}$ genau auszusehen hat, um die obigen Forderungen zu erfüllen.

Satz 3.30
Sei $\tilde{A} \in M(m \times n; \mathbb{Z})$ eine Matrix der folgenden Gestalt.

$$\begin{pmatrix} a_{11} & & 0 & \\ & \ddots & & 0 \\ 0 & & a_{rr} & \\ & 0 & & 0 \end{pmatrix}, \text{ mit } r \leq n, m;\ a_{ii} = 1\ \forall i \in \{1, ..., r\}$$

Dann gilt:

(i) $\operatorname{rang}(\tilde{A}) = r$

(ii) $\ker(\tilde{A}) = < e_{r+1}, ..., e_n >$, *wenn* $r < n$, $\ker(A) = \vec{0}$, *wenn* $r = n$

(iii) $\operatorname{bild}(\tilde{A}) = < e_1, e_2, ..., e_r >$

Beweis: Der Beweis ergibt sich direkt aus den Definitionen. □

Abhängig von den Angaben der Zeilenanzahl m, der Spaltenanzahl n und des Rangs r gibt sich das Programm also eine Matrix in der Gestalt von $\tilde{A}$ vor, womit eine Basis des Kerns und eine des Bildes schon festgelegt sind. Dann erzeugt das Programm einen zufälligen Vektor $\vec{\xi} \in \mathbb{Z}^n$, aus dem es mit $\tilde{A} \circ \vec{\xi} = \vec{\beta}$ den Vektor $\vec{\beta} \in \mathbb{Z}^m$ berechnet.
Mit den Matrizen P, Q wird das lineare Gleichungssystem nun verkompliziert. Über die obigen Vorgaben werden dann $\vec{x}$, $\vec{b}$, $\ker(A)$ und $\operatorname{bild}(A)$ berechnet.

Prinzip eines Kontrollalgorithmus $*$ für lineare Gleichungssysteme

Wieder taucht das Problem der Eindeutigkeit auf, denn die Lösungen eines linearen Gleichungssystems basieren auf dem Kern der Koeffizientenmatrix, der zu einer beliebigen speziellen Lösung des linearen Gleichungssystems hinzuaddiert wird. Das heißt, es gilt die Basis eines Untervektorraums zu bestimmen. Im vorangegangenen Abschnitt reichte ein Nachrechnen der definierenden Eigenschaft, hier ist sie zwar ein notwendiges Kriterium, aber kein hinreichendes, denn nun muss auch noch überprüft werden, ob wirklich eine

„Homogene LGS“:	
Aufgabenstellung: Lösen Sie das folgende homogene lineare Gleichungssystem $A \circ \vec{x} = \vec{0}$.	
Optionen:	
Gegeben:	Matrix $A \in M(m \times n; \mathbb{Z})$
Gesucht:	Lösung von $A \cdot \vec{x} = \vec{0}$
Variabel:	m,n, Rang von A
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$*$ (siehe oben)
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	61

Tabelle 3.22: „Homogene LGS“

Basis des Kerns vorliegt. Dazu muss man den Rang der Koeffizientenmatrix kennen, um die Dimension des Kerns ausrechnen zu können, um damit zu wissen, wie viele Vektoren die Basis des Kerns bilden, und zum Schluss überprüfen, ob diese auch linear unabhängig sind. Erst wenn man genauso viele linear unabhängige Vektoren gefunden hat, wie die Dimension des Kerns der Koeffizientenmatrix vorgibt, und jeder einzelne dieser Vektoren im Kern der Koeffizientenmatrix liegt, hat man eine Basis des Kerns der Koeffizientenmatrix gefunden.

3.2.3 Entwicklung der Aufgabenstellung

Aufgabenstellung 3.19

Lösen Sie das folgende homogene lineare Gleichungssystem $A \circ \vec{x} = \vec{0}$.

Einstellung im Fenster: „Homogene LGS“

Das Programm wählt $\vec{\xi} = \vec{0}$ und sorgt damit über die obigen Berechnungsvorschriften dafür, dass auch $\vec{b} = \vec{0}$ gilt. Im Allgemeinen ist die Lösung nicht eindeutig, sodass die oben dargestellte Überprüfung stattfinden muss. Ein Einsatz der Aufgabe mit OKUSON scheidet im allgemeinen Fall also aus.

Aufgabenstellung 3.20

Lösen Sie das folgende (in)homogene lineare Gleichungssystem $A \circ \vec{x} = \vec{b}$.

Einstellung im Fenster: „Inhomogene LGS“

Die Wahl $\vec{\xi} \neq \vec{0}$ sorgt im Allgemeinen dafür, dass auch $\vec{b} \neq \vec{0}$ gilt. Eine Ausnahme besteht dann, wenn $\vec{\xi} \in \ker(A)$ ist. Dann folgt $\vec{b} = \vec{0}$, sodass hier

„Inhomogene LGS“:	
Aufgabenstellung: Lösen Sie das folgende (in)homogene lineare Gleichungssystem $A \circ \vec{x} = \vec{b}$.	
Optionen:	
Gegeben:	Matrix $A \in M(m \times n; \mathbb{Z})$, $\vec{b} \in \mathbb{Z}^m$
Gesucht:	Lösung von $A \cdot \vec{x} = \vec{b}$
Variabel:	m,n, Rang von A
Eindeutigkeit der Lösung:	nicht vorhanden
Kontrollalgorithmus:	$*$ (siehe oben)
Verwendung mit OKUSON:	nicht möglich
Parameteraufruf mit:	62

Tabelle 3.23: „Inhomogene LGS“

auch homogene lineare Gleichungssysteme auftreten können. Es gelten die gleichen Einschränkungen, wie in der vorangegangenen Aufgabenstellung.

Aufgabenstellung 3.21

Welche der folgenden Matrizen $B_1, B_2, ... \in M(m \times n; \mathbb{Z})$ haben den Rang r?

Einstellung im Fenster: „Rang“

Man wählt die gleiche Startmatrix $\tilde{A}$ für die richtigen Lösungen und verändert dann den Wert r, um falsche Lösungen in Form von Matrizen zu bekommen. Da Multiplikationen mit invertierbaren Matrizen den Rang nicht verändern, sind die Lösungen eindeutig; zur Kontrolle reicht also ein simpler Abgleich. Die Aufgabe ist deshalb auch mit OKUSON kompatibel.

Aufgabenstellung 3.22

Für welche der folgenden Matrizen $B_1, B_2, ... \in M(m \times n; \mathbb{Z})$ bilden die folgenden Vektoren eine Basis des Kerns?

Einstellung im Fenster: „Kern-Basis“

Eine Variation von P erzeugt verschiedene richtige Lösungen. Falsche Lösungen bekommt man, indem man r variiert, aber P beibehält, sodass die gegebenen Vektoren zwar teilweise im Kern liegen, aber keine Basis mehr bilden. Eine andere Variante für falsche Lösungen ist die, dass in der Startmatrix $\tilde{A}$ ein $a_{ii} = 1$ seinen Platz mit einer Null auf der Hauptdiagonalen tauscht, sodass der Kern zwar noch die gleiche Dimension hat, aber seine Basis verändert worden ist.
Da Multiplikationen mit invertierbaren Matrizen den Rang nicht verändern,

„Rang“:	
Aufgabenstellung: Welche der folgenden Matrizen $B_1, B_2, ... \in M(m \times n; \mathbb{Z})$ haben den Rang r?	
Optionen:	
Gegeben:	Matrizen $B_1, B_2, ... \in M(m \times n; \mathbb{Z})$, Rang r
Gesucht:	Matrizen B mit Rang r
Variabel:	m,n, Anzahl der richtigen und falschen Lösungen, Rang r
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	72

Tabelle 3.24: „Rang“

sind die Lösungen eindeutig, zur Kontrolle reicht also ein simpler Abgleich. Die Aufgabe ist deshalb auch mit OKUSON kompatibel.

3.3 Andere Optionen

Die Tabellen im vorangegangenen Abschnitt haben schon einige der Optionen vorgestellt, die allen Aufgaben zur Verfügung stehen. Allerdings gibt es noch zwei Optionen für jede Aufgabe, die ich noch nicht vorgestellt habe, was ich hier der Vollständigkeit halber kurz tun möchte.

Es handelt sich dabei um die Anzahl der verschiedenen Aufgaben, die zu diesem Aufgabentyp erstellt werden sollen und einen Parameter, der dafür sorgt, dass Aufgaben mit zu großen ganzen Zahlen aussortiert werden. Dieser Parameter „Maximalbetrag“ legt ein Intervall fest, in dem alle Einträge in den Matrizen liegen dürfen. Damit kann man auf gewisse Weise die Schwierigkeit der Aufgaben verändern. Setzt man zum Beispiel 999 ein, so sortiert das Programm alle Aufgaben aus, in denen in Aufgabenstellung und Lösung eine Zahl größer als 999 oder kleiner als -999 ist.

Bei den anderen Optionen handelt es sich um Ausgabevarianten. Hier kann eingestellt werden, für welches Textverarbeitungsprogramm die Ausgabe geschehen soll. In einigen Fällen sind die Aufgaben nicht mit OKUSON kompatibel. Dort werden die Matrizen aber immerhin in der Schreibweise für OKUSON ausgegeben, damit man immer noch manuell ohne viel Mühe Auf-

„Kern-Basis“:

Aufgabenstellung:
Für welche der folgenden Matrizen $B_1, B_2, ... \in M(m \times n; \mathbb{Z})$ bilden die folgenden Vektoren eine Basis des Kerns?

Optionen:

Gegeben:	Matrizen $B_1, B_2, ... \in M(n \times n; \mathbb{Z})$, Basis des Kerns
Gesucht:	Matrizen B, deren Kern die gegebene Basis hat
Variabel:	m,n, Anzahl der richtigen und falschen Lösungen, Rang r
Eindeutigkeit der Lösung:	vorhanden
Kontrollalgorithmus:	Abgleich
Verwendung mit OKUSON:	möglich
Parameteraufruf mit:	71

Tabelle 3.25: „Kern-Basis“

gaben bauen kann. Mit OKUSON kompatible Aufgaben werden ansonsten in fertiger Form für Übungsblätter ausgegeben.

Kapitel 4

Invertierbarkeit von Matrizen über Integritätsringen

Wie im vorangegangenen Kapitel gesehen, fehlt „nur“ noch ein Kriterium, am besten auch gleich eine Art Algorithmus, mit dem man alle über $\mathbb{Z}$ invertierbaren Matrizen und ihre Inverse finden kann. Die erste Idee wäre natürlich das „Trial-and-Error-Prinzip“, bei dem man sich eine quadratische Matrix mit Einträgen aus $\mathbb{Z}$ vorgibt und das Programm überprüfen lässt, ob diese über $\mathbb{Z}$ invertierbar ist. Dabei tauchen gleich die ersten Probleme auf, denn es ist nicht klar, was die Invertierbarkeit von Matrizen über $\mathbb{Z}$ bedeutet. $\mathbb{Z}$ ist ein Ring, in dem nicht alle Elemente ein multiplikativ inverses Element besitzen. Die Frage ist deshalb, ob man sicherstellen kann, dass bestimmte Matrizen mit Einträgen aus $\mathbb{Z}$ eine inverse Matrix besitzen, die ebenfalls nur Einträge aus $\mathbb{Z}$ besitzt. Der Einfachheit halber verlasse ich $\mathbb{Z}$ und schaue mir zuerst einmal die Bedingungen für die Invertierbarkeit über Körpern an, um dann auf die Invertierbarkeit von Matrizen über Integritätsringen zu schließen. Das ist möglich, weil ich später zeige, dass Integritätsringe immer in einen Körper eingebettet werden können. Ausgehend von den Bedingungen der Invertierbarkeit von Matrizen über Integritätsringen, werde ich einen Algorithmus entwickeln, der in der Lage ist, zwei zueinander inverse Matrizen zu finden, deren Einträge nur aus $\mathbb{Z}$ stammen.

4.1 Voraussetzungen

Definition 4.1

(i) Ein Ring ist eine Menge R mit zwei inneren Verknüpfungen, die Addition $+\colon R \times R \to R$ und die Multiplikation $\cdot : R \times R \to R$, die den

folgenden Bedingungen genügen:[1]

(1) R ist eine kommutative Gruppe bezüglich der Addition.

(2) R ist ein Monoid[2] bezüglich der Multiplikation.

(3) Die Distributivgesetze gelten:

$$(a+b)\cdot c = a\cdot c + b\cdot c$$
$$c\cdot(a+b) = c\cdot a + c\cdot b$$
$$\forall a,b,c \in R$$

(ii) R heißt kommutativer Ring, wenn die Multiplikation zusätzlich noch kommutativ ist.

(iii) Sei $a \in R$ und es existiert ein $b \in R$ mit $a\cdot b = b\cdot a = 1$, dann ist a eine Einheit von R.
R^* ist die Menge der Einheiten von R:

$$R^* = \{a \in R; \exists\, b \in R \text{ mit } a\cdot b = b\cdot a = 1\}$$

Bezüglich der Multiplikation von R ist R^* eine Gruppe, die Einheitengruppe. Denn für jedes Element $e\cdot\tilde{e} \in R^*$ mit $e,\tilde{e} \in R^*$ gibt es ein Element $\tilde{e}^{-1}\cdot e^{-1} \in R$, so dass $e\cdot\tilde{e}\cdot\tilde{e}^{-1}\cdot e^{-1} = 1$ gilt.

(iv) Ein Ring R heißt Schiefkörper, wenn $R \neq 0$ und $R^* = R\backslash\{0\}$.

(v) Ein Ring R heißt Körper, wenn R kommutativ und ein Schiefkörper ist.

(vi) $a \in R$, R ein Ring, heißt Nullteiler, wenn $b \in R$, $b \neq 0$, existiert, sodass $a\cdot b = 0$ oder $b\cdot a = 0$.
Körper und Schiefkörper haben außer der Null keine Nullteiler.

(vii) Ein kommutativer Ring R heißt nullteilerfrei oder Integritätsring, wenn $R \neq 0$ und nur 0 als Nullteiler hat.

[3]

[1] Mit 0 ist nachfolgend das neutrale Element der Addition oder in einigen Fällen der Ring gemeint, der nur das neutrale Element der Addition enthält. Dies ergibt sich jeweils direkt aus dem Zusammenhang. Mit 1 ist das neutrale Element der Multiplikation gemeint.

[2] Die Multiplikation ist assoziativ, außerdem existiert bezüglich der Multiplikation ein neutrales Element in R.

[3] Vergleiche auch [Bosch, 2004, Seite 28 f.].

Definition 4.2

(i) $M(m \times n; R)$, $m, n \in \mathbb{N}$ ist die Menge der Matrizen mit m Zeilen und n Spalten mit Einträgen, die nur aus dem Ring R stammen.
$A = (a_{ij}) \in M(m \times n; R)$ ist eine rechteckige Anordnung der Elemente a_{ij} der folgenden Form:

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} \end{pmatrix}, \text{ wobei alle } a_{ij} \in R.$$

(ii) Sei R ein Ring mit den üblichen Verknüpfungen $+$ und $\cdot$ und seien $A = (a_{ij}), B = (b_{ij}) \in M(m \times n; R)$, $\lambda \in R$, dann sind folgende Verknüpfungen definiert:

(a)

$$+ \; : \; M(m \times n; R) \times M(m \times n; R) \to M(m \times n; R)$$
$$A + B = (c_{ij}) \text{ ,wobei } c_{ij} = a_{ij} + b_{ij}.$$

Diese Addition von Matrizen ist kommutativ, weil sie auf der kommutativen Addition des Ringes R basiert.

(b)

$$\cdot \; : \; R \times M(m \times n; R) \to M(m \times n; R)$$
$$\lambda \cdot A = (c_{ij}) \text{ ,wobei } c_{ij} = \lambda \cdot a_{ij}.$$

Diese Multiplikation einer Matrix mit einem Skalar ist genau dann kommutativ, wenn R ein kommutativer Ring ist, weil sie auf der Multiplikation des Ringes R basiert.

(iii) Sei R ein Ring mit den üblichen Verknüpfungen $+$ und $\cdot$ und seien $A = (a_{ij}) \in M(m \times n; R)$ und $B = (b_{jk}) \in M(n \times p; R)$, $m, n, p \in \mathbb{N}$, dann ist die Matrizenmultiplikation folgendermaßen definiert:

$$\circ \; : \; M(m \times n; R) \times M(n \times p; R) \to M(m \times p; R)$$
$$A \circ B = (c_{ik}) \text{ ,wobei } c_{ik} = \sum_{j=1}^{n} a_{ij} \cdot b_{jk}.$$

Die Assoziativität des Matrizenprodukts folgt aus der Kommutativität der Addition von R und den Distributivgesetzen von R.

(iv) Matrizen der folgenden Art haben spezielle Namen:

(a) $E_n \in M(n \times n; R)$ mit

$$E_n = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

nennt sich die n-te Einheitsmatrix.

(b) $0 \in M(m \times n; R)$ mit

$$0 = \begin{pmatrix} 0 & & 0 \\ & \ddots & \\ 0 & & 0 \end{pmatrix}$$

nennt sich Nullmatrix.

(c) $A = (a_{ij}) \in M(n \times n; R)$ mit

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

nennt sich obere Dreiecksmatrix.

(d) $A = (a_{ij}) \in M(n \times n; R)$ mit

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

nennt sich untere Dreiecksmatrix.

(v) Sei $A = (a_{ij}) \in M(m \times n; K)$. $A^T = (a_{ji}) \in M(n \times m; K)$ mit

$$A^T = (a_{ji}) = \begin{pmatrix} a_{11} & \cdots & a_{i1} & \cdots & a_{m1} \\ \vdots & & \vdots & & \vdots \\ a_{1j} & \cdots & a_{ij} & \cdots & a_{mj} \\ \vdots & & \vdots & & \vdots \\ a_{1n} & \cdots & a_{in} & \cdots & a_{mn} \end{pmatrix}$$

ist die Transponierte von A.

[4]

[4] Vergleiche auch [Wille, 2001, Seite 79 ff.].

Definition 4.3
Sei R ein Ring mit den üblichen Verknüpfungen + und $\cdot$ und sei $A \in M(n \times n; R)$, $n \in \mathbb{N}$. A heißt über dem Ring R invertierbar, wenn es ein $A^{-1} \in M(n \times n; R)$ gibt, sodass gilt:

$$A \circ A^{-1} = E_n$$

Das Ergebnis ist die Einheitsmatrix E_n mit n Zeilen und n Spalten.
Die über R invertierbaren $n \times n$ Matrizen bilden bezüglich der Matrizenmultiplikation die Gruppe $GL(n; R)$, weil für alle $A, B \in GL(n; R)$

$$A \circ B \circ B^{-1} \circ A^{-1} = E_n, \text{ sowie } E_n \circ A = A = A \circ E_n$$

gilt. Also ist $(A \circ B)^{-1} = B^{-1} \circ A^{-1}$. [5]

Lemma 4.4
Seien $A = (a_{ij}) \in M(m \times n; R)$ und $B = (b_{jk}) \in M(n \times p; R)$ und R ein kommutativer Ring mit den üblichen Verknüpfungen + und $\cdot$, dann gilt $(A \circ B)^T = B^T \circ A^T$.

Beweis: Seien $(A \circ B)^T = (c_{ki})$ und $B^T \circ A^T = (d_{ki})$. Es gilt dann

$$c_{ki} = \sum_{j=1}^{n} a_{ij} \cdot b_{jk} \text{ und } d_{ki} = \sum_{j=1}^{n} b_{jk} \cdot a_{ij}.$$

Also $c_{ki} = d_{ki}$ und damit folgt die Behauptung. [6] $\square$

In den folgenden Abschnitten ist mit R immer ein kommutativer Ring mit den üblichen Verknüpfungen + und $\cdot$ gemeint. Sollte dies anders sein, ist es an der betreffenden Stelle vermerkt.

4.2 Die Determinantenfunktion

Die Determinantenfunktion liefert ein Kriterium für die Invertierbarkeit von Matrizen über einem Körper K. Doch um dieses Kriterium zu gewinnen, ist viel Arbeit nötig.
Zuerst wird sich zeigen, dass sich alle über einem Körper K invertierbaren Matrizen aus dem Matrizenprodukt sogenannter Elementarmatrizen zusammensetzen lassen. Dann muss man nur noch ein Kriterium entwickeln, wann dies für eine quadratische Matrix der Fall ist; dieses Kriterium liefert die Determinantenfunktion.

[5] Vergleiche auch [Fischer, 2002, Seite 149 f.].
[6] Vergleiche auch [Wille, 2001, Seite 86 f.].

4.2.1 Elementarmatrizen

Definition 4.5

Sei K ein Körper, $m \in \mathbb{N}$, $1 \leq i, j \leq m$ mit $i \neq j$ und $\lambda \in K$, dann nennt man die folgenden quadratischen Matrizen Elementarmatrizen.

$$S_i(\lambda) := \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & 0 & \\ & & 1 & & & & \\ & & & \lambda & & & \\ & & & & 1 & & \\ & 0 & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

(λ steht in der i-ten Zeile)

$$Q_i^j(\lambda) := \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & 0 & \\ & & 1 & & & & & & \\ & & & 1 & 0 & \cdots & 0 & \lambda & \\ & & & & 1 & & & 0 & \\ & & & & & \ddots & & \vdots & \\ & & & & & & 1 & 0 & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & 0 & & & & & & & 1 \end{pmatrix}$$

(λ steht in der i-ten Zeile und der j-ten Spalte.)

$$P_i^j := \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & 0 & \\ & & 1 & & & & & & \\ & & & 0 & 0 & \cdots & 0 & 1 & \\ & & & 0 & 1 & & & 0 & \\ & & & \vdots & & \ddots & & \vdots & \\ & & & 0 & & & 1 & 0 & \\ & & & 1 & 0 & \cdots & 0 & 0 & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & 0 & & & & & & & 1 \end{pmatrix}$$

(Die Nullen auf der Diagonalen stehen in der i-ten Zeile und der j-ten Zeile.)

[7]

Bemerkung 4.6

(i) Multipliziert man diese Elementarmatrizen von links bzw. von rechts an eine beliebige Matrix $A \in M(n \times m; K)$ bzw. $A \in M(m \times n; K)$, so hat das auf die Matrix A die folgenden Auswirkungen:

(a) $S_i(\lambda)$ bewirkt, von links multipliziert, die Multiplikation der i-ten Zeile von A mit λ, von rechts multipliziert, die Multiplikation der i-ten Spalte von A mit λ.

[7] Vergleiche auch [Fischer, 2002, Seite 163 f.].

(b) $Q_i^j(\lambda)$ bewirkt, von links multipliziert, die Addition des λ-fachen der j-ten Zeile von A zur i-ten Zeile von A, von rechts multipliziert, die Addition des λ-fachen der i-ten Spalte von A zur j-ten Spalte von A.

(c) P_i^j bewirkt, von links multipliziert, die Vertauschung der i-ten Zeile von A mit der j-ten Zeile von A, von rechts multipliziert, die Vertauschung der i-ten Spalte von A mit der j-ten Spalte von A.

(ii) Zur weiteren Vereinfachung kann man einige der Elementarmatrizen durch ein Produkt aus anderen Elementarmatrizen und einfacheren Versionen zusammensetzen:

$$Q_i^j(\lambda) = S_j(\frac{1}{\lambda}) \circ Q_i^j(1) \circ S_j(\lambda)$$
$$P_i^j = Q_j^i(1) \circ Q_i^j(-1) \circ Q_j^i(1) \circ S_j(-1)$$

Mit der obigen Argumentation kann man diese Gleichungen leicht nachvollziehen.

[8]

Lemma 4.7

Die Elementarmatrizen sind invertierbar, ihre Inversen sind:

$$(S_i(\lambda))^{-1} = S_i(\frac{1}{\lambda}),$$
$$(Q_i^j(\lambda))^{-1} = Q_i^j(-\lambda),$$
$$(P_i^j)^{-1} = P_i^j.$$

Beweis: Der Beweis folgt durch Ausrechnen. [9] □

Satz 4.8

Jede invertierbare Matrix $A \in M(n \times n; K)$ lässt sich als ein endliches Produkt von Elementarmatrizen schreiben. Die Gruppe der invertierbaren Matrizen $GL(n; K)$ wird also von den Elementarmatrizen erzeugt.

Beweis: Der Rang r einer invertierbaren $n \times n$ Matrix ist gleich n.[10] Deshalb kann man A durch r Multiplikationen von Elementarmatrizen von links auf

[8] Vergleiche auch [Fischer, 2002, Seite 164 f.].

[9] Vergleiche auch [Fischer, 2002, Seite 166].

[10] Jede lineare Abbildung kann durch eine Matrix definiert werden und umgekehrt. Eine invertierbare Matrix A steht für eine bijektive Abbildung φ von einem Vektorraum V in einen Vektorraum W, wobei $\dim(V) = \dim(W) = n$ gilt. In der Matrix stehen dann in den Spalten die Bilder der kanonischen Einheitsvektoren von V. Damit φ bijektiv ist, müssen diese Bilder der Einheitsvektoren linear unabhängig sein, also ist der Rang von A gleich n. Vergleiche auch [Fischer, 2002, Seite 117 f.].

die Zeilenstufenform B bringen. Diesen Vorgang bezeichnet man als Gaußsches Eliminationsverfahren.[11] Der Rang wird durch Multiplikation mit invertierbaren Matrizen nicht verändert. B ist dann von der folgenden Gestalt:

$$B = \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ & \ddots & \vdots \\ 0 & & b_{nn} \end{pmatrix}$$

Die Diagonalelemente $b_{11}, ..., b_{nn}$ müssen dann von Null verschieden sein, weil A und B sonst nicht den Rang n hätten. Es gilt dann:

$$B = B_r \circ ... \circ B_1 \circ A.$$

Durch weitere Zeilenumformungen wird B zur Einheitsmatrix, indem man alle Einträge von B bis auf die Einträge auf der Hauptdiagonalen mit Hilfe eben dieser Einträge beseitigt. Zum Schluss werden die Einträge in der Hauptdiagonalen noch normiert. Also existieren s Elementarmatrizen, sodass gilt:

$$E_n = B_s \circ ... \circ B_{r+1} \circ B = B_s \circ ... \circ B_1 \circ A.$$

Es folgt

$$A^{-1} = B_s \circ ... \circ B_1 \Leftrightarrow A = B_1^{-1} \circ ... \circ B_s^{-1}$$

und mit dem obigen Lemma die Behauptung. [12] □

Damit ist gezeigt, dass sich alle über einem Körper K invertierbaren Matrizen aus Elementarmatrizen zusammenbauen lassen und alle Produkte aus Elementarmatrizen invertierbar sind. Auf dieses Ergebnis wird in den folgenden Unterabschnitten immer wieder Bezug genommen.

4.2.2 Permutationen

Die Permutationen bilden den ersten Schritt auf dem Weg zur Determinantenfunktion und ihrer Definition über die Leibnizformel.[13]

Definition 4.9

Sei M eine Menge mit genau n verschiedenen Elementen. Eine Permutation ist eine Anordnung der Elemente von M ohne Auslassung und Wiederholung eines der Elemente.[14]

[11] Johann Carl Friedrich Gauß (*30.4.1777 in Braunschweig: †23.2.1855 in Göttingen)

[12] Vergleiche auch [Fischer, 2002, Seite 166].

[13] Die Leibnizformel stammt von Gottfried Wilhelm Leibniz (*1.7.1646 in Leipzig; †14.9.1716 in Hannover). Zu Ehren eines der letzten Universalgelehrten hat sich im Übrigen die Universität Hannover am 1.7.2006 in Gottfried-Wilhelm-Leibniz-Universität Hannover umbenannt.

[14] Vergleiche auch [Anton, 1998, Seite 87].

Beispiel 4.10

Sei $M = \{1, 2, 3\}$, dann existieren 6 verschiedene Permutationen:

$$(1, 2, 3), (3, 1, 2), (2, 3, 1), (2, 1, 3), (3, 2, 1), (1, 3, 2)$$

Wählt man eine Permutation der Menge als Grundanordnung aus, so kann man die Permutationen als Bilder dieser Grundabbildung ansehen.

$$\left(\begin{smallmatrix}1&2&3\\1&2&3\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3\\3&1&2\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3\\2&3&1\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3\\2&1&3\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3\\3&2&1\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3\\1&3&2\end{smallmatrix}\right)$$

Bemerkung 4.11

1. Die Permutationen einer Menge mit genau n verschiedenen Elementen bilden, interpretiert als Abbildungen einer gewählten Grundanordnung, eine Gruppe. Diese Gruppe nennt man die symmetrische Gruppe S_n.[15] Im obigen Beispiel ist also die S_3 dargestellt, die Symmetriegruppe eines gleichseitigen Dreiecks.

2. Permutationen sind, interpretiert als Abbildungen einer Grundanordnung von Elementen, bijektiv, weil im Bild weder ein Element ausgelassen (Surjektivität) noch wiederholt (Injektivität) wird.

Hat man eine Grundanordnung der n verschiedenen Elemente einer beliebigen endlichen Menge gewählt, so kann man sie auch mit ihrer Position in der Grundanordnung identifizieren. Dies wird im Folgenden getan, weil es die weitere Betrachtung von Permutationen erleichtert.

Definition 4.12

(i) Sei $\sigma \in S_n$. Ein Paar (i, j) mit $1 \leq i < j \leq n$ heißt eine Inversion von σ, wenn $\sigma(i) > \sigma(j)$ gilt.
Die Anzahl der Inversionen bezeichnet man mit $\text{inv}(\sigma)$.

(ii) Falls $\text{inv}(\sigma)$ $\left\{ \begin{smallmatrix} gerade \\ ungerade \end{smallmatrix} \right\}$ ist, heißt σ eine $\left\{ \begin{smallmatrix} gerade \\ ungerade \end{smallmatrix} \right\}$ Permutation. Das Signum von σ ist für $n \geq 2$:

$$\text{sgn}(\sigma) = \prod_{i<j} \frac{\sigma(j) - \sigma(i)}{j - i} = \prod_{\substack{\{i,j\} \\ i \neq j}} \frac{\sigma(j) - \sigma(i)}{j - i}$$

Für $n = 1$ gilt:

$$\text{sgn}(\sigma) = 1$$

[16]

[15] Vergleiche auch [Beutelspacher, 2001, Seite 174].
[16] Vergleiche auch [Bessenrodt, 2002, §6].

Bemerkung 4.13
Aus dieser Definition ergibt sich schnell:

$$\sigma \left\{ \begin{smallmatrix} gerade \\ ungerade \end{smallmatrix} \right\} \Leftrightarrow \operatorname{sgn}(\sigma) \left\{ \begin{smallmatrix} \geq \\ < \end{smallmatrix} \right\} 0$$

[17]

Lemma 4.14
Sei $\sigma \in S_n$, dann gilt:

$$\operatorname{sgn}(\sigma) \in \{\pm 1\}$$

Beweis: Für $n = 1$ ist die Behauptung wahr, für $n \geq 2$ gilt: $\sigma \in S_n$ permutiert die 2-elementigen Mengen $\{i, j\}$, weil σ bijektiv ist:

$$\{i, j\} \longmapsto \{\sigma(i), \sigma(j)\}$$

Also:

$$\prod_{i<j} (\sigma(j) - \sigma(i)) = \pm \prod_{i<j} (j - i)$$
$$\Rightarrow \operatorname{sgn}(\sigma) = \pm 1$$

[18] □

Folgerung 4.15

$$\operatorname{sgn}(\sigma) = (-1)^{\operatorname{inv}(\sigma)}$$

[19]

Satz 4.16
Seien $\sigma, \tau \in S_n$, dann gilt:

$$\operatorname{sgn}(\sigma \cdot \tau) = \operatorname{sgn}(\sigma) \cdot \operatorname{sgn}(\tau)$$

Beweis: Für $n = 1$ folgt die Behauptung sofort, da $\sigma = \tau = \mathrm{id}$ gilt.
Für $n \geq 2$ gilt:

$$\operatorname{sgn}(\sigma \cdot \tau) = \prod_{\substack{\{i,j\} \\ i \neq j}} \frac{\sigma\tau(j) - \sigma\tau(i)}{\tau(j) - \tau(i)} \cdot \frac{\tau(j) - \tau(i)}{j - i}$$

[17] Vergleiche auch [Bessenrodt, 2002, §6].
[18] Vergleiche auch [Bessenrodt, 2002, §6].
[19] Vergleiche auch [Bessenrodt, 2002, §6].

Da τ die 2-elementigen Teilmengen permutiert, gilt:

$$\prod_{\substack{\{i,j\}\\ i\neq j}} \frac{\sigma\tau(j)-\sigma\tau(i)}{\tau(j)-\tau(i)} = \prod_{\substack{\{i,j\}\\ i\neq j}} \frac{\sigma(j)-\sigma(i)}{j-i} = \operatorname{sgn}(\sigma)$$

Es folgt die Behauptung. [20] $\square$

Bemerkung 4.17

(i) $A_n := \{\sigma \in S_n; \operatorname{sgn}(\sigma) = 1\} = \ker(\operatorname{sgn})$ ist damit ein Normalteiler[21] von S_n. A_n nennt sich alternierende Gruppe.

(ii) Für $n \geq 2$ ist sgn surjektiv. Also gilt für $n \geq 2$ nach dem Homomorphiesatz[22] $S_n/A_n \cong \mathbb{Z}_2$. Es folgt daher $|A_n| = \frac{|A_n|}{2} = \frac{n!}{2}$.
[23]

Definition 4.18

Sei $\sigma \in S_n$. $\exists i,j \in \{1,...,n\}$, $i \neq j$, sodass $\sigma(i) = j$, $\sigma(j) = i$, $\sigma(k) = k$, $\forall k \in \{1,...,n\}\backslash\{i,j\}$, dann heißt σ eine Transposition. [24]

Lemma 4.19

(i) Sei σ eine Transposition, dann gilt $\operatorname{sgn}(\sigma) = -1$.

(ii) Sei σ ein Produkt aus k Transpositionen, dann gilt $\operatorname{sgn}(\sigma) = (-1)^k$.
[25]

Beweis:

(i) Vertauscht σ die Zahlen k, l, dann kann man ohne Einschränkung der Allgemeinheit $k < l$ annehmen und man erhält:

$$\operatorname{sgn}(\sigma) = \prod_{i<j} \frac{\sigma(j)-\sigma(i)}{j-i}$$

[20] Vergleiche auch [Bessenrodt, 2002, §6].
[21] Zur Definition eines Normalteilers siehe [Bosch, 2004, Seite 18].
[22] [Bosch, 2004, Seite 17, Satz 6, Korollar 7].
[23] Vergleiche auch [Bessenrodt, 2002, §6].
[24] Vergleiche auch [Bessenrodt, 2002, §6].
[25] Vergleiche auch [Bessenrodt, 2002, §6].

$$= \frac{\sigma(l)-\sigma(k)}{l-k} \cdot \prod_{\substack{i<j \\ i\neq k; j\neq l}} \frac{\sigma(j)-\sigma(i)}{j-i} \cdot \prod_{j<k} \frac{\sigma(k)-\sigma(j)}{k-j}$$
$$\cdot \prod_{\substack{k<j \\ j\neq l}} \frac{\sigma(j)-\sigma(k)}{j-k} \cdot \prod_{\substack{i<l \\ i\neq k}} \frac{\sigma(l)-\sigma(i)}{l-i} \cdot \prod_{l<i} \frac{\sigma(i)-\sigma(l)}{i-l}$$
$$= -1 \cdot 1 \cdot \prod_{j<k} \frac{l-j}{k-j} \cdot \prod_{\substack{k<j \\ j\neq l}} \frac{j-l}{j-k} \cdot \prod_{\substack{i<l \\ i\neq k}} \frac{k-i}{l-i} \cdot \prod_{l<i} \frac{i-k}{i-l}$$
$$= -1$$

[26]

(ii) Diese Behauptung folgt mit (i) direkt aus Satz 4.16.

□

Satz 4.20

Sei $n \geq 2$, dann wird die S_n von Transpositionen erzeugt. Jedes $\sigma \in S_n$ ist ein Produkt von höchstens $n-1$ Transpositionen.

Beweis: Ist $\sigma(1) = 1$, so setze $\tau_1 := \mathrm{id}$, andernfalls definiere τ_1 als Transposition von 1 und $\sigma(1)$. Für $\sigma_1 := \tau_1\sigma$ gilt dann $\sigma_1(1) = 1$.
Ist $\sigma(2) = 2$, so setze $\tau_2 := \mathrm{id}$, andernfalls definiere τ_2 als Transposition von 2 und $\sigma(2)$. Für $\sigma_2 := \tau_2\tau_1\sigma$ gilt dann $\sigma_2(i) = i$, für $i = 1, 2$.
Dieses Verfahren wird sukzessive fortgeführt. Im $(n-1)$-ten Schritt gilt mit $\sigma_{n-1} := \tau_{n-1} \cdots \tau_1\sigma$, $\tau_1, ..., \tau_{n-1}$ wie oben:

$$\sigma_{n-1}(i) = i, \text{ für } i = 1, ..., n-1$$

Da σ als Permutation bijektiv ist, folgt $\sigma_{n-1}(n) = n$. Es gilt dann:

$$\sigma_{n-1} = \tau_{n-1} \cdots \tau_1\sigma = \mathrm{id} \Rightarrow \sigma = \tau_1 \cdots \tau_{n-1}$$

Also folgt, dass jedes $\sigma \in S_n$ ein Produkt aus höchstens $n-1$ Transpositionen ist. [27] □

[26] Vergleiche auch [Pohlers, 1999, Seite 114].
[27] Vergleiche auch [Bessenrodt, 2002, §6].

4.2.3 Eigenschaften der Determinantenfunktion

Mit diesen Voraussetzungen lässt sich nun die Determinantenfunktion definieren sowie ihre Existenz und ihre Eindeutigkeit beweisen:

Definition 4.21

Sei K ein Körper und $n \in \mathbb{N}$, $A = (a_{ij})$. Eine Abbildung det: $M(n \times n; K) \to K$, $A \mapsto \det(A)$, heißt Determinante, wenn die folgenden Bedingungen gelten:

(i) det ist linear in jeder Zeile, für jeden Index $i \in \{1, .., n\}$ gilt also:
Sind die a_j, $j \in \{1, .., n\}$, die Zeilenvektoren, die in A stehen und ist $a_i = \tilde{a}_i + \bar{a}_i$, so gilt:

$$\det \begin{pmatrix} \vdots \\ a_{i-1} \\ a_i \\ a_{i+1} \\ \vdots \end{pmatrix} = \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \tilde{a}_i + \bar{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix} = \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \tilde{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix} + \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \bar{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix}.$$

Sind die a_j, $j \in \{1, .., n\}$, die Zeilenvektoren, die in A stehen und ist $a_i = \lambda \tilde{a}_i$, so gilt:

$$\det \begin{pmatrix} \vdots \\ a_{i-1} \\ a_i \\ a_{i+1} \\ \vdots \end{pmatrix} = \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \lambda \tilde{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix} = \lambda \cdot \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \tilde{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix}.$$

(ii) det ist alternierend. Es gilt $\det(A) = 0$, wenn A mindestens zwei gleiche Zeilen hat.

(iii) det ist normiert. Es gilt $\det(E_n) = 1$.

[28]

Satz 4.22

Seien $A, B \in M(n \times n; K)$, K ein Körper. Die Determinantenfunktion det *hat die folgenden Eigenschaften:*

(i) $\forall \lambda \in K$ *gilt* $\det(\lambda \cdot A) = \lambda^n \cdot A$.

(ii) Enthält A eine Nullzeile, gilt $\det(A) = 0$.

(iii) Sei B eine Matrix, die durch genau eine Zeilenvertauschung aus A entsteht, so ist $\det(B) = -\det(A)$.

[28] Vergleiche auch [Fischer, 2002, Seite 178 f.].

(iv) *Sei $\lambda \in K$ und B eine Matrix, die aus A durch die Addition des λ-fachen der j-ten Zeile von A zur i-ten Zeile von A $(i \neq j)$ entsteht, so ist* $\det(B) = \det(A)$.

(v) *Ist A eine obere Dreiecksmatrix,*

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ & \ddots & \vdots \\ 0 & & a_{nn} \end{pmatrix},$$

so ist $\det(A) = a_{11} \cdots a_{nn}$.

Die Behauptung gilt natürlich auch für untere Dreiecksmatrizen, der Beweis verläuft dann analog.

(vi) *Für $n \geq 2$ und $A \in M(n \times n; K)$ von der Gestalt*

$$A = \begin{pmatrix} A_1 & R \\ 0 & A_2 \end{pmatrix},$$

mit A_1 und A_2 quadratisch gilt $\det(A) = \det(A_1) \cdot \det(A_2)$.

(vii)

$$\det(A) = 0 \Leftrightarrow \operatorname{rang}(A) < n$$

(viii)

$$\det(A \circ B) = \det(A) \cdot \det(B)$$

Für $A \in GL(n; K)$ gilt insbesondere:

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

Beweis:

(i) Die Behauptung folgt direkt aus der ersten Eigenschaft der Definition.

(ii) Die Behauptung folgt direkt aus der ersten Eigenschaft der Definition.

(iii) Sind a_j, $j \in \{1, .., n\}$, die Zeilenvektoren, die in A stehen und angenommen die i-te und j-te Zeile seien bei A und B jeweils vertauscht, dann gilt:

$$\det(A) + \det(B) = \det \begin{pmatrix} a_1 \\ \vdots \\ a_i \\ \vdots \\ a_j \\ \vdots \\ a_n \end{pmatrix} + \det \begin{pmatrix} a_1 \\ \vdots \\ a_j \\ \vdots \\ a_i \\ \vdots \\ a_n \end{pmatrix}$$

$$= \det \begin{pmatrix} a_1 \\ \vdots \\ a_i \\ \vdots \\ a_i \\ \vdots \\ a_n \end{pmatrix} + \det \begin{pmatrix} a_1 \\ \vdots \\ a_i \\ \vdots \\ a_j \\ \vdots \\ a_n \end{pmatrix} + \det \begin{pmatrix} a_1 \\ \vdots \\ a_j \\ \vdots \\ a_i \\ \vdots \\ a_n \end{pmatrix} + \det \begin{pmatrix} a_1 \\ \vdots \\ a_j \\ \vdots \\ a_j \\ \vdots \\ a_n \end{pmatrix}$$

$$= \det \begin{pmatrix} a_1 \\ \vdots \\ a_i+a_j \\ \vdots \\ a_i+a_j \\ \vdots \\ a_n \end{pmatrix} = 0.$$

(iv) Sind a_j, $j \in \{1, .., n\}$, die Zeilenvektoren, die in A stehen, dann gilt:

$$\det(B) = \det \begin{pmatrix} a_1 \\ \vdots \\ a_i+\lambda \cdot a_j \\ \vdots \\ a_j \\ \vdots \\ a_n \end{pmatrix} = \det(A) + \lambda \cdot \det \begin{pmatrix} a_1 \\ \vdots \\ a_j \\ \vdots \\ a_j \\ \vdots \\ a_n \end{pmatrix} = \det(A).$$

(v) Für den Fall $a_{ii} \neq 0 \, \forall i \in \{1, ..., n\}$ kann man die obere Dreiecksmatrix über k-fache Zeilenadditionen auf Diagonalgestalt bringen und es folgt:

$$\det(A) = \det \begin{pmatrix} a_{11} & & 0 \\ & \ddots & \\ 0 & & a_{nn} \end{pmatrix} = a_{11} \cdot ... \cdot a_{nn} \cdot \det(E_n) = a_{11} \cdot ... \cdot a_{nn}.$$

Im Fall $a_{ii} = 0$ für mindestens ein i, wird das größte solche i gewählt. Sollte die i-te Zeile noch keine Nullzeile sein, so wird mit k-fachen Zeilenadditionen der Zeilen unterhalb der i-ten Zeile die i-te Zeile ausgeräumt. Es folgt $\det(A) = 0$.

(vi) Durch k-fache Zeilenadditionen und Zeilenvertauschungen kann man A_1 in eine obere Dreiecksmatrix B_1 umwandeln. A_2 wird dabei nicht verändert, aber R wird zu $\tilde{R}$. Sei i die Anzahl der Zeilenvertauschungen, dann gilt:

$$\det(A_1) = (-1)^i \cdot \det(B_1).$$

Auf die gleiche Art wird A_2 in die obere Dreiecksmatrix B_2 umgewandelt. Sei hier j die Anzahl der Zeilenoperationen, dann gilt:

$$\det(A_2) = (-1)^j \cdot \det(B_2).$$

Man erhält die obere Dreiecksmatrix

$$B := \begin{pmatrix} B_1 & \tilde{R} \\ 0 & B_2 \end{pmatrix}.$$

Es folgt $\det(B) = \det(B_1) \cdot \det(B_2)$ und aus $\det(A) = (-1)^{i+j} \cdot \det(B)$ die Behauptung.

(vii) Durch k-fache Zeilenadditionen und Zeilenvertauschungen kann man A auf Zeilenstufenform bringen, also in eine obere Dreiecksmatrix B umwandeln. Es gilt $\det(A) = \pm \det(B)$ und $\text{rang}(A) = \text{rang}(B)$. Damit folgt:

$$\text{rang}(B) = n \Leftrightarrow \det(B) = b_{11} \cdot ... \cdot b_{nn} \neq 0.$$

(viii) Für den Fall $\text{rang}(A) < n$, ist auch $\text{rang}(A \circ B) < n$ und die zu zeigende Gleichung lautet $0 = 0$.
Also können wir annehmen, dass $A \in GL(n; K)$. Damit können wir A als Produkt von s Elementarmatrizen schreiben: $A = C_1 \circ ... \circ A_s$. Also genügt es zu zeigen, dass für jede Elementarmatrix C vom Typ $S_i(\lambda)$ oder $Q_i^j(1)$

$$\det(C \circ B) = \det(C) \cdot \det(B)$$

gilt. Mit den schon gezeigten Eigenschaften folgt:

$$\det(S_i(\lambda)) = \lambda,$$
$$\det(Q_i^j(1)) = 1.$$

Nach Bemerkung 4.6 bewirkt eine Multiplikation von $S_i(\lambda)$ von links an B die Multiplikation der i-ten Zeile von B mit λ. Also gilt:

$$\det(S_i(\lambda) \circ B) = \lambda \cdot \det(B).$$

Eine Multiplikation von $Q_i^j(1)$ von links an B hat die Auswirkung, dass die j-te Zeile von B zur i-ten addiert wird. Also gilt:

$$\det(Q_i^j(1) \circ B) = 1 \cdot \det(B).$$

Es folgt die Behauptung.

Damit sind die Eigenschaften der Determinantenfunktion gezeigt.[29] □

Allerdings ist noch nicht klar, ob eine Funktion mit diesen Eigenschaften überhaupt existiert und ob sie dann auch eindeutig ist.

[29] Vergleiche auch [Fischer, 2002, Seite 179 ff.].

4.2.4 Existenz und Eindeutigkeit der Determinantenfunktion

Lemma 4.23

Seien e_i für $i \in \{1, ..., n\}$ die Einheitsvektoren in Zeilenform. Für jede Permutation $\sigma \in S_n$ gilt dann:

$$\det \begin{pmatrix} e_{\sigma(1)} \\ \vdots \\ e_{\sigma(n)} \end{pmatrix} = \operatorname{sgn}(\sigma).$$

Beweis: $\sigma \in S_n$ lässt sich durch k Transpositionen zusammenbauen, $\sigma = \tau_1 \cdots \tau_k$. Dementsprechend kann man E_n durch k Zeilenvertauschungen in die obige Matrix überführen. Es folgt die Behauptung.[30] $\square$

Satz 4.24

Ist K ein Körper und $n \geq 1$, so gibt es genau eine Determinantenfunktion

$$\det : M(n \times n; K) \to K.$$

Sei $A = (a_{ij}) \in M(n \times n; K)$, dann ist die Determinante von A:

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma(i)}.$$

Diese Formel nennt man auch Leibnizformel.

Beweis: Zuerst zeige ich die Eindeutigkeit, indem ich die Formel mit Hilfe ihrer Eigenschaften entwickle:
Seien e_i für $i \in \{1, ..., n\}$ die Einheitsvektoren in Zeilenform, $A = (a_{ij})$ wie oben und a_i die Zeilenvektoren, die in A stehen, dann gilt:

$$\begin{aligned} \det(A) = \det(a_{ij}) &= \sum_{i_1=1}^{n} a_{1i_1} \cdot \det \begin{pmatrix} e_{i_1} \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \\ &= \sum_{i_1=1}^{n} a_{1i_1} \cdot \sum_{i_2=1}^{n} a_{2i_2} \cdot \det \begin{pmatrix} e_{i_1} \\ e_{i_2} \\ a_3 \\ \vdots \\ a_n \end{pmatrix} = ... \\ = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_n=1}^{n} a_{1i_1} \cdot a_{2i_2} \cdot ... \cdot a_{ni_n} &\cdot \det \begin{pmatrix} e_{i_1} \\ \vdots \\ e_{i_n} \end{pmatrix}. \end{aligned}$$

[30] Vergleiche auch [Fischer, 2002, Seite 192].

Aus dem vorangegangenen Lemma ergibt sich:

$$\det \begin{pmatrix} e_{i_1} \\ \vdots \\ e_{i_n} \end{pmatrix} = \begin{cases} \operatorname{sgn}(\sigma), & \text{falls es ein } \sigma \in S_n \text{ gibt mit } \sigma(\nu) = i_\nu \text{ für alle } \nu, \\ 0, & \text{sonst.} \end{cases}$$

Von den n^n Summanden verbleiben also nur $n!$. Damit ist die Eindeutigkeit gezeigt.

Um die Existenz zu zeigen, müssen die drei Axiome der Determinantenfunktion überprüft werden:

(i)

$$\det \begin{pmatrix} \vdots \\ a_{i-1} \\ \tilde{a}_i + \bar{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix}$$

$$= \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot a_{1\sigma(1)} \cdots a_{i-1,\sigma(i-1)} \cdot (\tilde{a}_{i\sigma(i)} + \bar{a}_{i\sigma(i)}) \cdot a_{i+1,\sigma(i+1)} \cdots a_{n\sigma(n)}$$

$$= \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot a_{1\sigma(1)} \cdots a_{i-1,\sigma(i-1)} \cdot \tilde{a}_{i\sigma(i)} \cdot a_{i+1,\sigma(i+1)} \cdots a_{n\sigma(n)}$$

$$+ \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot a_{1\sigma(1)} \cdots a_{i-1,\sigma(i-1)} \cdot \bar{a}_{i\sigma(i)} \cdot a_{i+1,\sigma(i+1)} \cdots a_{n\sigma(n)}$$

$$= \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \tilde{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix} + \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \bar{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix}$$

$$\det \begin{pmatrix} \vdots \\ a_{i-1} \\ \lambda \tilde{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix}$$

$$= \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot a_{1\sigma(1)} \cdots a_{i-1,\sigma(i-1)} \cdot \lambda \tilde{a}_{i\sigma(i)} \cdot a_{i+1,\sigma(i+1)} \cdots a_{n\sigma(n)}$$

$$= \lambda \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot a_{1\sigma(1)} \cdots a_{i-1,\sigma(i-1)} \cdot \tilde{a}_{i\sigma(i)} \cdot a_{i+1,\sigma(i+1)} \cdots a_{n\sigma(n)}$$

$$= \lambda \cdot \det \begin{pmatrix} \vdots \\ a_{i-1} \\ \tilde{a}_i \\ a_{i+1} \\ \vdots \end{pmatrix}$$

(ii) Angenommen die k-te und l-te Zeile seien gleich, wobei aber $k < l$ gelten soll. Nun sei τ genau die Transposition, die k und l vertauscht. Weil A_n ein Normalteiler von S_n ist und $S_n/A_n = \mathbb{Z}_2$ gilt, gilt $S_n = A_n \cup A_n\tau$. Diese Vereinigung ist aufgrund der Normalteilereigenschaft von A_n disjunkt. Es gilt nun:

$$\det(A) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma(i)} = \sum_{\sigma\in A_n} \prod_{i=1}^{n} a_{i\sigma(i)} - \sum_{\sigma\in A_n} \prod_{i=1}^{n} a_{i\sigma(\tau(i))}$$

Aufgrund der Definition von τ und der Kommutativität der Multiplikation von K gilt nun

$$\begin{aligned}\prod_{i=1}^{n} a_{i\sigma(\tau(i))} &= a_{1\sigma(\tau(1))} \cdots a_{k\sigma(\tau(k))} \cdots a_{l\sigma(\tau(l))} \cdots a_{n\sigma(\tau(n))} \\ &= a_{1\sigma(1)} \cdots a_{k\sigma(l)} \cdots a_{l\sigma(k)} \cdots a_{n\sigma(n)} \\ &= a_{1\sigma(1)} \cdots a_{k\sigma(k)} \cdots a_{l\sigma(l)} \cdots a_{n\sigma(n)} \\ &= \prod_{i=1}^{n} a_{i\sigma(i)},\end{aligned}$$

weil die k-te und l-te Zeile gleich sind.
Damit heben sich alle Summanden der obigen Summe auf und es folgt $\det(A) = 0$.

(iii)

$$\det(E_n) = \det((\delta_{ij})) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) \cdot \delta_{1\sigma(1)} \cdots \delta_{n\sigma(n)} = \operatorname{sgn}(\mathrm{id}) = 1$$

δ_{ij} ist das Kroneckersymbol[31] und $(\delta_{ij}) = \begin{pmatrix} \delta_{11} & \cdots & \delta_{1j} & \cdots & \delta_{1n} \\ \vdots & & \vdots & & \vdots \\ \delta_{i1} & \cdots & \delta_{ij} & \cdots & \delta_{in} \\ \vdots & & \vdots & & \vdots \\ \delta_{n1} & \cdots & \delta_{nj} & \cdots & \delta_{nn} \end{pmatrix}$.

[32] □

Vorerst bleibt also festzuhalten, dass $A \in GL(n;K) \Leftrightarrow \det(A) \neq 0$ gilt.

Korollar 4.25
Sei $A \in M(n \times n; K)$ eine Matrix über dem Körper K, dann gilt

$$\det(A) = \det(A^T).$$

[31] Es gilt $\delta_{ij} = 1$, wenn $i = j$, und $\delta_{ij} = 0$, wenn $i \neq j$.
[32] Vergleiche auch [Fischer, 2002, Seite 192 ff.].

Beweis: Setze $\sigma(i) = j$, dann gilt mit $i = \sigma^{-1}(j)$ aufgrund der Kommutativität der beiden Verknüpfungen in K:

$$\begin{aligned}\det(A) &= \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i\sigma(i)} \\ &= \operatorname{sgn}^2(\sigma^{-1}) \cdot \sum_{\sigma^{-1} \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{\sigma^{-1}(j)j} \\ &= \sum_{\sigma^{-1} \in S_n} \operatorname{sgn}^2(\sigma^{-1}) \cdot \operatorname{sgn}(\sigma) \prod_{j=1}^{n} a_{\sigma^{-1}(j)j} \\ &= \sum_{\sigma^{-1} \in S_n} \operatorname{sgn}(\sigma^{-1}) \prod_{j=1}^{n} a_{\sigma^{-1}(j)j} = \det(A^T).\end{aligned}$$

□

Damit gelten die besagten Eigenschaften der Determinantenfunktion nicht nur für die Zeilen, sondern auch für die Spalten von Matrizen.

4.3 Die Invertierungsformel

Mit den Ergebnissen des letzten Abschnittes kann man zwar schon Einiges anfangen, allerdings reichen sie noch nicht aus, um ein Kriterium zu entwickeln, wann eine Matrix über dem Ring der ganzen Zahlen invertierbar ist. Dazu muss noch eine Formel zur Invertierung von Matrizen über einem Körper K angegeben werden.

4.3.1 Minoren, Kofaktoren und die Adjunkte einer Matrix

Definition 4.26

(i) Sei $A = (a_{ij}) \in M(n \times n; K)$, dann ist der Minor M_{ij} des Elements a_{ij} folgendermaßen definiert:

$$M_{ij} = \det \begin{pmatrix} a_{11} & \cdots & a_{1,j-1} & a_{1,j+1} & \cdots & a_{1n} \\ \vdots & & & & & \vdots \\ a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & & & & & \vdots \\ a_{n1} & \cdots & a_{n,j-1} & a_{n,j+1} & \cdots & a_{nn} \end{pmatrix}.$$

Die i-te Zeile und die j-te Spalte von A wurden gestrichen.

(ii) Sei $A = (a_{ij}) \in M(n \times n; K)$, dann ist der Kofaktor C_{ij} des Elements a_{ij}:

$$C_{ij} = (-1)^{i+j} \cdot M_{ij}.$$

[33]

Satz 4.27 (Kofaktorentwicklung, Laplacescher Entwicklungssatz)
Sei $A = (a_{ij}) \in M(n \times n; K)$, dann gilt:

$$\det(A) = a_{i1}C_{i1} + a_{i2}C_{i2} + ... + a_{in}C_{in}.$$

[34]

Beweis: Als erstes wird die i-te Zeile so nach oben getauscht, dass alle anderen Zeilen ihre Reihenfolge beibehalten. Dazu sind $i-1$ Zeilenvertauschungen notwendig, die sich in einem Faktor $(-1)^{i-1}$ in der Determinante von A bemerkbar machen.

$$\det \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = a_{i1} \cdot (-1)^{i-1} \cdot \det \begin{pmatrix} 1 & 0 & \cdots & 0 \\ a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{i-1,1} & a_{i-1,2} & \cdots & a_{i-1,n} \\ a_{i+1,1} & a_{i+1,2} & \cdots & a_{i+1,n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

$$+a_{i2} \cdot (-1)^{i-1} \cdot \det \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{i-1,1} & a_{i-1,2} & a_{i-1,3} & \cdots & a_{i-1,n} \\ a_{i+1,1} & a_{i+1,2} & a_{i-1,3} & \cdots & a_{i+1,n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} + ...$$

$$+a_{in} \cdot (-1)^{i-1} \cdot \det \begin{pmatrix} 0 & \cdots & 0 & 1 \\ a_{11} & \cdots & a_{1,n-1} & a_{1n} \\ \vdots & & \vdots & \vdots \\ a_{i-1,1} & \cdots & a_{i-1,n-1} & a_{i-1,n} \\ a_{i+1,1} & \cdots & a_{i+1,n-1} & a_{i+1,n} \\ \vdots & & \vdots & \vdots \\ a_{n1} & \cdots & a_{n,n-1} & a_{nn} \end{pmatrix}$$

Man kann das Verfahren hier schon abbrechen, indem man von Satz 4.22 die Eigenschaft (vi) zusammen mit Korollar 4.25 benutzt, allerdings ist dann die folgende Bemerkung nicht so leicht nachzuvollziehen.
Stellvertretend für alle wird deshalb der j-te Summand betrachtet: Auch in dieser Determinante tauschen wir die j-te Spalte so in die erste Spalte, dass alle anderen ihre ursprüngliche Reihenfolge beibehalten. Dazu sind $j - 1$

[33] Vergleiche auch [Anton, 1998, Seite 112].
[34] Pierre-Simon (Marquis de) Laplace (*28.3.1749 Beaumont-en-Auge; †5.3.1827 in Paris)

Spaltenvertauschungen notwendig, die sich in einem Faktor $(-1)^{j-1}$ in der Determinante bemerkbar machen. Dass Spalten auf diese Weise vertauscht werden dürfen folgt aus Korollar 4.25, denn Spaltenvertauschungen sind eigentlich Zeilenvertauschungen in der transponierten Matrix. Es gilt dann:

$$
\begin{aligned}
& a_{ij} \cdot (-1)^{i-1} \cdot \det \begin{pmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ a_{11} & \cdots & a_{1,j-1} & a_{1j} & a_{1,j+1} & \cdots & a_{1n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & \cdots & a_{n,j-1} & a_{nj} & a_{n,j+1} & \cdots & a_{nn} \end{pmatrix} \\
= & a_{ij} \cdot (-1)^{i-1+j-1} \cdot \det \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ a_{1j} & a_{11} & \cdots & a_{1,j-1} & a_{1,j+1} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{i-1,j} & a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ a_{i+1,j} & a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{nj} & a_{n1} & \cdots & a_{n,j-1} & a_{n,j+1} & \cdots & a_{nn} \end{pmatrix} \\
= & a_{ij} \cdot (-1)^{i+j} \det \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & a_{11} & \cdots & a_{1,j-1} & a_{1,j+1} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ 0 & a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & a_{n1} & \cdots & a_{n,j-1} & a_{n,j+1} & \cdots & a_{nn} \end{pmatrix}
\end{aligned}
$$

Die Null in der ersten Spalte und der k-ten Zeile, $k \in \{2, ..., n\}$, ist folgendermaßen entstanden: Wenn $a_{kj} \neq 0$ ist, dann wurde die erste Zeile mit $-a_{kj}$ multipliziert und zur k-ten Zeile addiert. Anschließend wurde die erste Zeile wieder durch $-a_{kj}$ geteilt. Bei $a_{kj} = 0$ ist nichts zu tun. Dieses Verfahren ändert den Wert der Determinante nicht.
Es gilt nun aufgrund der Eigenschaften der Determinantenfunktion:

$$
\begin{aligned}
& a_{ij} \cdot (-1)^{i+j} \det \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & a_{11} & \cdots & a_{1,j-1} & a_{1,j+1} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ 0 & a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & a_{n1} & \cdots & a_{n,j-1} & a_{n,j+1} & \cdots & a_{nn} \end{pmatrix} \\
= & a_{ij} \cdot (-1)^{i+j} \cdot M_{ij} = a_{ij} \cdot C_{ij}.
\end{aligned}
$$

Und hieraus folgt die Behauptung. □

Bemerkung 4.28

Mit Hilfe der im obigen Beweis vorgestellten Methode ist es leicht einzusehen, dass für $i \neq j$

$$a_{i1}C_{j1} + a_{i2}C_{j2} + ... + a_{in}C_{jn} = 0$$

gilt, denn werden die Schritte aus dem obigen Beweis umgekehrt durchgeführt, dann baut man in der j-ten Zeile der entstehenden Matrix ein weiteres Mal die i-te Zeile zusammen. Berechnet man dann die Determinante der entstandenen Matrix auf die übliche Weise, so hat man zwei gleiche Zeilen und die Determinante ist Null.

Definition 4.29
Sei $A = (a_{ij}) \in M(n \times n; K)$, K ein Körper, und seien die C_{ij} die Kofaktoren der Elemente a_{ij}, dann ist

$$\operatorname{adj}(A) = \begin{pmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ C_{12} & C_{22} & \cdots & C_{n2} \\ \vdots & \vdots & & \vdots \\ C_{1n} & C_{2n} & \cdots & C_{nn} \end{pmatrix}$$

die Adjunkte der Matrix A. [35]

4.3.2 Die Invertierungsformel für Matrizen

Satz 4.30
Sei $A = (a_{ij}) \in GL(n; K)$, dann gilt:

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

Beweis:

$$A \circ \operatorname{adj}(A) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \circ \begin{pmatrix} C_{11} & C_{21} & \cdots & C_{j1} & \cdots & C_{n1} \\ C_{12} & C_{22} & \cdots & C_{j2} & \cdots & C_{n2} \\ \vdots & \vdots & & \vdots & & \vdots \\ C_{1n} & C_{2n} & \cdots & C_{jn} & \cdots & C_{nn} \end{pmatrix}$$

Nun wird das Element p_{ij} in der i-ten Zeile und der j-ten Spalte des Produktes betrachtet:

$$p_{ij} = a_{i1}C_{j1} + a_{i2}C_{j2} + ... + a_{in}C_{jn} = \begin{cases} \det(A), & \text{falls } i = j, \\ 0, & \text{sonst.} \end{cases}$$

[35] Vergleiche auch [Anton, 1998, Seite 116].

Es gilt also:

$$A \circ \operatorname{adj}(A) = \begin{pmatrix} \det(A) & & 0 \\ & \ddots & \\ 0 & & \det(A) \end{pmatrix} = \det(A) \cdot E_n$$
$$\Leftrightarrow A \circ \left(\frac{1}{\det(A)} \operatorname{adj}(A) \right) = E_n.$$

Und damit folgt die Behauptung. [36] □

Nachdem nun ein Kriterium gefunden wurde, mit dem man zeigen kann, unter welchen Bedingungen eine quadratische Matrix über einem Körper invertierbar ist, und eine Vorschrift, um diese inverse Matrix auch zu berechnen, kann man schließlich ein Kriterium zur Invertierbarkeit von Matrizen über dem Ring der ganzen Zahlen, $\mathbb{Z}$, erarbeiten.

4.4 Invertierbarkeit über Integritätsringen

4.4.1 Vorüberlegungen

Folgerung 4.31

(i) Die Determinantenfunktion ist als eine Funktion definiert worden, die Matrizen mit Einträgen aus einem Körper K auf Elemente eines Körpers K abbildet. Diese Voraussetzungen werden nun auch nicht mehr geändert; sie haben dementsprechend Auswirkungen auf die Ringe, die ich noch verwenden kann. Diese Ringe können also höchstens Teilmengen des Körpers K sein, mit den gleichen Verknüpfungen $+$ und $\cdot$, wie sie auch auf dem Körper definiert sind. Die Ringe, die demnach nur in Frage kommen, sind Integritätsringe, da sie wie Körper kommutative Ringe sind und keine Nullteiler besitzen.
Die Frage ist nun noch, ob man aus jedem Integritätsring R auch einen Körper, den Quotientenkörper $Q(R)$, konstruieren kann, der diesen Integritätsring enthält. Man kann, wie nachfolgend gezeigt wird.

(ii) Für $A = (a_{ij}) \in GL(n; K)$ gilt $A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A)$, wenn K ein Körper ist, doch was passiert, wenn K lediglich ein Integritätsring R ist, aus dem man einen Körper bauen kann?
Seien also $A, A^{-1} \in GL(n; Q(R))$. $Q(R)$ ist der Quotientenkörper von

[36] Vergleiche auch [Anton, 1998, Seite 117 f.].

dem Integritätsring R, also der Körper, der sich aus R bauen lässt. Alle Einträge in A, A^{-1} seien Elemente von R. Es gilt also:

$$1 = \det(E_n) = \underbrace{\det(A)}_{\in R} \cdot \underbrace{\det(A^{-1})}_{\in R}$$

Die Determinanten von A, A^{-1} sind nach Satz 4.24 Elemente von R. Demzufolge müssen die Determinanten von A, A^{-1} Einheiten von R sein. Also muss gelten:

$$A = (a_{ij}) \in GL(n; R) \Rightarrow \det(A) \in R^*$$

Sei nun $A \in M(n \times n; R)$, R ein Integritätsring, mit $\det(A) \in R^*$ dann gilt

$$A = (a_{ij}) \in GL(n; R) \Leftarrow \det(A) \in R^*,$$

weil Satz 4.30 über $Q(R)$ gilt und mit Satz 4.24 auch alle Elemente von $\mathrm{adj}(A)$ in R liegen. Es folgt:

$$A = (a_{ij}) \in GL(n; R) \Leftrightarrow \det(A) \in R^*.$$

(iii) Das Matrizenprodukt zweier über einem Integritätsring R invertierbarer Matrizen A, B ist ebenfalls über R invertierbar, denn die Einträge des Produktes $A \circ B$ stammen ebenfalls aus R und die Determinante $\det(A \circ B)$ ist ein Element der multiplikativen Einheitengruppe.

(iv) Ich habe zu einem früheren Zeitpunkt gezeigt, dass die $GL(n; K)$, K ein Körper, von den Elementarmatrizen über K erzeugt wird. Doch nicht für alle diese Elementarmatrizen A mit Einträgen eines Integritätsringes R, der eine Teilmenge von K ist, gilt $\det(A) \in R^*$. Demzufolge wird in diesem Abschnitt gezeigt, wie man die Menge der Elementarmatrizen über K verändern muss, um eine Gruppe $GL(n; R)$ der über R invertierbaren Matrizen zu erzeugen, und dass die Elemente dieser veränderten Menge die $GL(n; R)$ erzeugt.

4.4.2 Konstruktion des Quotientenkörpers

Satz 4.32

Jeder Integritätsring R mit den Verknüpfungen $+$ und $\cdot$ ist ein Unterring eines Körpers, nämlich des Quotientenkörpers $Q(R)$.[37]

[37] Jeder Schüler kennt mit den Bruchzahlen schon einen Quotientenkörper, nämlich den Quotientenkörper des Rings der ganzen Zahlen. Man beachte einfach die Analogien im folgenden Satz.

Beweis: M sei die folgende Menge von Paaren aus Elementen des Integritätsrings R:

$$M = \{(a,b); a \in R, b \in R\backslash\{0\}\}.$$

Auf der Menge M kann man eine Äquivalenzrelation $\sim$ definieren:

$$(a,b) \sim (\tilde{a},\tilde{b}) \Leftrightarrow a \cdot \tilde{b} = \tilde{a} \cdot b \text{ für } (a,b),(\tilde{a},\tilde{b}) \in M$$

Die Eigenschaften einer Äquivalenzrelation müssen noch nachgewiesen werden:

(i) Reflexivität:

$$(a,b) \sim (a,b)\ \forall (a,b) \in M, \text{ denn: } a \cdot b = a \cdot b.$$

(ii) Symmetrie:

$$(a,b) \sim (\tilde{a},\tilde{b}) \Rightarrow (\tilde{a},\tilde{b}) \sim (a,b)\ \forall (a,b),(\tilde{a},\tilde{b}) \in M,$$
$$\text{denn: } a \cdot \tilde{b} = \tilde{a} \cdot b = a \cdot \tilde{b}.$$

(iii) Transitivität:

$$(a,b) \sim (\tilde{a},\tilde{b}), (\tilde{a},\tilde{b}) \sim (\bar{a},\bar{b})$$
$$\Rightarrow (a,b) \sim (\bar{a},\bar{b})\ \forall (a,b),(\tilde{a},\tilde{b}),(\bar{a},\bar{b}) \in M, \text{denn:}$$
$$a \cdot \tilde{b} = \tilde{a} \cdot b \Rightarrow a \cdot \tilde{b} \cdot \bar{b} = \tilde{a} \cdot b \cdot \bar{b}$$
$$\tilde{a} \cdot \bar{b} = \bar{a} \cdot \tilde{b} \Rightarrow \tilde{a} \cdot b \cdot \bar{b} = \bar{a} \cdot b \cdot \tilde{b}$$
$$\Rightarrow a \cdot \tilde{b} \cdot \bar{b} = \bar{a} \cdot b \cdot \tilde{b}$$
$$\Rightarrow a \cdot \bar{b} = \bar{a} \cdot b, \text{ weil } R \text{ ein Integritätsring ist und } b \neq 0.$$

Jede Äquivalenzrelation zerlegt die Menge, auf der sie definiert wird, in disjunkte Äquivalenzklassen. In diesen Äquivalenzklassen sind alle Elemente enthalten, die äquivalent zueinander sind. Hier ist $Q(R) = M/\sim$ die Menge der Äquivalenzklassen. Die zu $(a,b) \in M$ gehörige Äquivalenzklasse wird mit $\left[\frac{a}{b}\right] \in Q(R)$ bezeichnet. Ihre Elemente $\frac{a}{b} \in \left[\frac{a}{b}\right]$ sind die Repräsentanten der Äquivalenzklassen, mit denen man ersatzweise rechnen kann:

$$\frac{a}{b} \sim \frac{\tilde{a}}{\tilde{b}} \Leftrightarrow a \cdot \tilde{b} = \tilde{a} \cdot b$$
$$\text{bzw.}$$
$$\left[\frac{a}{b}\right] = \left[\frac{\tilde{a}}{\tilde{b}}\right] \Leftrightarrow a \cdot \tilde{b} = \tilde{a} \cdot b$$

Mit den folgenden Verknüpfungen $\oplus$ und $\odot$ folgt, dass $Q(R)$ ein Körper ist:

$$\oplus : Q(R) \times Q(R) \to Q(R),\ \left[\frac{a}{b}\right] \oplus \left[\frac{\tilde{a}}{\tilde{b}}\right] = \left[\frac{a \cdot \tilde{b} + \tilde{a} \cdot b}{b \cdot \tilde{b}}\right]$$

$$\odot : Q(R) \times Q(R) \to Q(R),\ \left[\frac{a}{b}\right] \odot \left[\frac{\tilde{a}}{\tilde{b}}\right] = \left[\frac{a \cdot \tilde{a}}{b \cdot \tilde{b}}\right]$$

Es muss noch gezeigt werden, dass diese Definitionen unabhängig von der Wahl der Repräsentanten sind. Also ist zu zeigen, dass für $\frac{a}{b}, \frac{\tilde{a}}{\tilde{b}} \in \left[\frac{a}{b}\right]$ und für $\frac{c}{d}, \frac{\tilde{c}}{\tilde{d}} \in \left[\frac{c}{d}\right]$

$$\frac{a}{b} \oplus \frac{c}{d} = \frac{\tilde{a}}{\tilde{b}} \oplus \frac{\tilde{c}}{\tilde{d}}\ \forall \left[\frac{a}{b}\right], \left[\frac{c}{d}\right] \in Q(R)$$

und

$$\frac{a}{b} \odot \frac{c}{d} = \frac{\tilde{a}}{\tilde{b}} \odot \frac{\tilde{c}}{\tilde{d}}\ \forall \left[\frac{a}{b}\right], \left[\frac{c}{d}\right] \in Q(R)$$

gelten und diese Definitionen die Körperaxiome erfüllen:

(i) Wohldefiniertheit von $\oplus$ und $\odot$:

$\oplus$:

$$\begin{aligned}
&\frac{a \cdot d + b \cdot c}{b \cdot d} \sim \frac{\tilde{a} \cdot \tilde{d} + \tilde{b} \cdot \tilde{c}}{\tilde{b} \cdot \tilde{d}} \\
\Leftrightarrow\ & a \cdot d + b \cdot c \sim \frac{\tilde{a} \cdot \tilde{d} + \tilde{b} \cdot \tilde{c}}{\tilde{b} \cdot \tilde{d}} \cdot b \cdot d \\
\Leftrightarrow\ & a \cdot d + b \cdot c \sim \frac{a \cdot \tilde{b} \cdot \tilde{d} \cdot d + \tilde{b} \cdot c \cdot \tilde{d} \cdot b}{\tilde{b} \cdot \tilde{d}} \\
\Leftrightarrow\ & a \cdot d + b \cdot c \sim a \cdot d + b \cdot c
\end{aligned}$$

$\odot$:

$$\begin{aligned}
&\frac{a \cdot c}{b \cdot d} \sim \frac{\tilde{a} \cdot \tilde{c}}{\tilde{b} \cdot \tilde{d}} \\
\Leftrightarrow\ & a \cdot c \sim \frac{\tilde{a} \cdot \tilde{c} \cdot b \cdot d}{\tilde{b} \cdot \tilde{d}} \\
\Leftrightarrow\ & a \cdot c \sim \frac{a \cdot \tilde{b} \cdot c \cdot \tilde{d}}{\tilde{b} \cdot \tilde{d}} \\
\Leftrightarrow\ & a \cdot c \sim a \cdot c
\end{aligned}$$

(ii) Die Assoziativität und Kommutativität der beiden Verknüpfungen folgt aus der Assoziativität bzw. der Kommutativität der Verknüpfungen von R.

(iii) Das Distributivgesetz folgt ebenfalls aus dem Distributivgesetz von R.

(iv) Das neutrale Element bezüglich $\oplus$ ist $\left[\frac{0}{a}\right], a \in R\backslash\{0\}$.
Das neutrale Element bezüglich $\odot$ ist $\left[\frac{a}{a}\right], a \in R\backslash\{0\}$.

(v) Das inverse Element von $\left[\frac{a}{b}\right] \in Q(R)$ bezüglich $\oplus$ ist $\left[\frac{-a}{b}\right] \in Q(R)$.
Das inverse Element von $\left[\frac{a}{b}\right] \in Q(R)$, $a, b \in R\backslash\{0\}$, bezüglich $\odot$ ist $\left[\frac{b}{a}\right] \in Q(R)$.

Mit der Angabe eines injektiven Ringhomomorphismus, der zeigt, dass R ein Unterring von $Q(R)$ ist, folgt die Behauptung.

$$R \to Q(R), a \mapsto \left[\frac{a}{1}\right]$$

[38] □

4.4.3 Elementarmatrizen über einem Integritätsring

Analog zu den Elementarmatrizen über einem Körper K sollte es auch Matrizen über einem Integritätsring R, geben die die Gruppe der über R invertierbaren Matrizen, $GL(n; R)$, erzeugen.
Die Elementarmatrizen über einem Körper K sind sicherlich ein guter Anfang, um Elementarmatrizen von R zu finden. Allerdings müssen sie teilweise erst eingeschränkt und dann um einen neuen Typ erweitert werden.

Lemma 4.33

Sei R ein Integritätsring und $S_i(\lambda) \in M(n \times n; R)$ so definiert wie im Unterabschnitt über Elementarmatrizen über einem Körper K, dann ist $S_i(\lambda)$ nur für den Fall $\lambda \in R^*$ über R invertierbar.

Beweis:

$$S_i(\lambda) = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & 0 & \\ & & 1 & & & \\ & & & \lambda & & \\ & & & & 1 & \\ & 0 & & & & \ddots \\ & & & & & & 1 \end{pmatrix}$$

(λ steht in der i-ten Zeile)

$$\Rightarrow \det(S_i(\lambda)) = \lambda$$

$$\Rightarrow S_i(\lambda) \in GL(n; R)\ \forall \lambda \in R^* \text{ nach Folgerung 4.31 } (ii).$$

□

[38] Vergleiche auch [Bosch, 2004, Seite 61 f.] und [Bothmer, 2004, Kapitel 2, Satz 4.5].

Lemma 4.34
$Q_i^j(\lambda)$, $P_i^j \in M(n \times n; R)$ sind so, wie sie im Anschnitt über Elementarmatrizen definiert worden sind, über einem Integritätsring R invertierbar.

Beweis:

$Q_i^j(\lambda)$:

$$Q_i^j(\lambda) = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & 0 & \\ & & 1 & & & & & & & \\ & & & 1 & 0 & \cdots & 0 & \lambda & & \\ & & & & 1 & & & 0 & & \\ & & & & & \ddots & & \vdots & & \\ & & & & & & 1 & 0 & & \\ & & & & & & & 1 & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & 0 & & & & & & & & 1 \end{pmatrix}$$

(λ steht in der i-ten Zeile und der j-ten Spalte.)

$$\Rightarrow \det(Q_i^j(\lambda)) = 1 \in R^*$$

P_i^j:

$$P_i^j = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & 0 & \\ & & 1 & & & & & & & \\ & & & 0 & 0 & \cdots & 0 & 1 & & \\ & & & 0 & 1 & & & 0 & & \\ & & & \vdots & & \ddots & & \vdots & & \\ & & & 0 & & & 1 & 0 & & \\ & & & 1 & 0 & \cdots & 0 & 0 & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & 0 & & & & & & & & 1 \end{pmatrix}$$

(Die Nullen auf der Diagonalen stehen in der i-ten Zeile und der j-ten Zeile.)

$$\Rightarrow \det(P_i^j) = -1 \in R^*,$$
$$\text{wegen } 0 = -1(1-1) = -1 + (-1)^2 \Leftrightarrow 1 = (-1)^2,$$
$$\text{weil } R \text{ ein Integritätsring ist.}$$

□

Mit Hilfe der Elementarmatrizen konnte man eine Matrix über einem Körper K in eine obere Dreiecksmatrix umformen, ohne ihren Rang zu verändern. Doch mit der Einschränkung auf einen Integritätsring R mussten auch die Elementarmatrizen eingeschränkt werden, weil nicht mehr alle über R invertierbar sind. Die Frage ist nun, mit welcher über R invertierbaren Matrix man das gleiche über R schafft.

Bemerkung 4.35

Angenommen der Zeilenvektor $(u, v), u, v \in R$, soll mit Hilfe einer von rechts multiplizierten und über dem Integritätsring R invertierbaren Matrix M so verändert werden, dass seine zweite Komponente zu Null wird. Dies ist notwendig, um eine Matrix über R in eine obere Dreiecksmatrix transformieren zu können, ohne ihren Rang zu verändern. Über einem Körper K ist das leicht; hier ist im Fall $n = 2$

$$M = \tilde{m} \cdot \begin{pmatrix} m_{11} & -\frac{v}{u} \\ m_{21} & 1 \end{pmatrix}, \tilde{m}, m_{11}, m_{21} \in K.$$

Für $\tilde{m} = m_{11} = 1, m_{21} = 0$ gilt insbesondere $M = Q_1^2(-\frac{v}{u})$. Zwar gilt $\det(Q_1^2(-\frac{v}{u})) = 1$, aber im Allgemeinen ist $Q_1^2(-\frac{v}{u}) \notin GL(2; R)$, weil nicht immer $-\frac{v}{u} \in R$ gilt.
Es gilt also eine solche Matrix M zu finden, wenn man erreichen möchte, dass eine Matrix über R nur mit Spalten- bzw. nur mit Zeilenoperationen in eine obere Dreiecksmatrix transformiert werden kann, ohne ihren Rang zu verändern.

Definition 4.36

Sei R ein Integritätsring, $n \in \mathbb{N}$, $1 \leq i, j \leq n$ mit $i \neq j$ und $\lambda \in K$, dann definiere man $G_i^j(a, b, c, d)$ durch:

$$G_i^j(a, b, c, d) := \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & 0 & \\ & & 1 & & & & & & & \\ & & & d & 0 & \cdots & 0 & -b & & \\ & & & 0 & 1 & & & 0 & & \\ & & & \vdots & & \ddots & & \vdots & & \\ & & & 0 & & & 1 & 0 & & \\ & & & -c & 0 & \cdots & 0 & a & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & 0 & & & & & & & & & 1 \end{pmatrix},$$

$$(g_{ii} = d, g_{ij} = -b, g_{ji} = -c, g_{jj} = a)$$

$$a, b, c, d \in R \text{ mit } a \cdot d - b \cdot c = 1.$$

Bemerkung 4.37

(i) $\det(G_i^j(a, b, c, d)) = a \cdot d - b \cdot c = 1$

(ii) $G_i^j(a,b,c,d) \in R$, R ein Integritätsring, ist über R invertierbar.

$$(G_i^j(a,b,c,d))^{-1} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & 0 & \\ & & 1 & & & & & & \\ & & & a & 0 & \cdots & 0 & b & \\ & & & 0 & 1 & & & 0 & \\ & & & \vdots & & \ddots & & \vdots & \\ & & & 0 & & & 1 & 0 & \\ & & & c & 0 & \cdots & 0 & d & \\ & & & & & & & & 1 \\ & & & & & & & & & \ddots \\ & 0 & & & & & & & & & 1 \end{pmatrix},$$

$$(g_{ii} = a, g_{ij} = b, g_{ji} = c, g_{jj} = d)$$

$$a, b, c, d \in R \text{ mit } a \cdot d - b \cdot c = 1.$$

Es stellt sich die Frage, wie dieses $G_i^j(a,b,c,d)$ dabei helfen soll, eine Matrix zu diagonalisieren. Dazu müssen die Integritätsringe noch weiter eingeschränkt werden.

4.4.4 Ideale und Hauptidealringe

Definition 4.38

(i) Es sei R ein Ring. Eine Teilmenge $\iota \subset R$ heißt ein Ideal in R, wenn gilt:

(a) ι ist eine additive Untergruppe von R.

(b) $r \in R, i \in \iota \Rightarrow r \cdot i \in \iota$

(ii) Eigenschaften von Idealen:

(a) Sei ι ein Ideal eines Ringes R. Eine Familie (i_j), $j \in J$, J eine Indexmenge, von Elementen aus ι heißt ein Erzeugendensystem von ι, wenn $\iota = \sum_{j \in J} R \cdot i_j$ gilt, also ι mit dem von der Familie erzeugten Ideal übereinstimmt.

(b) ι heißt endlich erzeugt, wenn ι ein endliches Erzeugendensystem besitzt.

(c) ι heißt Hauptideal, wenn ι von einem einzigen Element erzeugt wird: $\iota = (i), i \in \iota$.

(d) Wenn R ein Integritätsring ist und jedes Ideal in R ein Hauptideal ist, dann heißt R Hauptidealring.

(e) Die Addition zweier Ideale ι_1, ι_2 des Ringes R erzeugt, wie man leicht nachrechnet, ein neues Ideal ι_{12}:

$$\iota_{12} := \iota_1 + \iota_2 := \{i_1 + i_2; i_1 \in \iota_1, i_2 \in \iota_2\}.$$

Seien R ein Hauptidealring und i_1, i_2 die Erzeuger von ι_1 bzw. ι_2, dann ist auch ι_{12} ein Hauptideal. Es gilt dann:

$$\iota_{12} := \iota_1 + \iota_2 := (i_1) + (i_2) := \{r \cdot i_1 + \tilde{r} \cdot i_2; r, \tilde{r} \in R\}.$$

[39]

Lemma 4.39

Sei ι ein Hauptideal eines Ringes R, $\iota = (i) = (j)$, $i, j \in R$, aber $i \neq j$ und $i, j \neq 0$, dann gilt $i = e \cdot j$ mit $e \in R^*$.

Beweis: Da ι Hauptideal ist, gilt $i = a \cdot j$ und $b \cdot i = j$ mit $a, b \in R$. Es folgt: $i = a \cdot b \cdot i \Leftrightarrow a \cdot b = 1$, weil R als Hauptidealring auch ein Integritätsring ist. Demnach ist $a = b^{-1}$ und damit $a, b \in R^*$. $\square$

Lemma 4.40

Sei R ein Hauptidealring und ι_1, ι_2 und ι_{12} Ideale mit $\iota_1 = (i_1)$, $\iota_2 = (i_2)$ und $\iota_1 + \iota_2 = \iota_{12}$, dann ist ι_{12} das kleinste Ideal, das i_1 und i_2 enthält.

Beweis: Wegen $\iota_{12} = \{r \cdot i_1 + \tilde{r} \cdot i_2; r, \tilde{r} \in R\}$ gilt $i_1 \in \iota_{12}$ und $i_2 \in \iota_{12}$.
Sei nun $(\mathrm{ggt}(i_1, i_2))$ das kleinste Ideal, das i_1 und i_2 enthält.[40] Definiere nun $g \in \iota_{12}$ mit $g = r \cdot i_1 + \tilde{r} \cdot i_2$ mit $r, \tilde{r} \in R$. Weil $i_1, i_2 \in (\mathrm{ggt}(i_1, i_2))$ und $(\mathrm{ggt}(i_1, i_2))$ ein Ideal ist, sind auch $r \cdot i_1, \tilde{r} \cdot i_2 \in (\mathrm{ggt}(i_1, i_2))$. Damit ist auch $g \in (\mathrm{ggt}(i_1, i_2))$, denn jedes Ideal ist eine additive Untergruppe. Also gilt $\iota_{12} \subset (\mathrm{ggt}(i_1, i_2))$. Da aber $i_1, i_2 \in \iota_{12}$ und $(\mathrm{ggt}(i_1, i_2))$ das kleinste Ideal mit dieser Eigenschaft ist, folgt $\iota_{12} = (\mathrm{ggt}(i_1, i_2))$. $\square$

Definition 4.41

Sei R ein Hauptidealring.

(i) $r \in R$ teilt $\tilde{r} \in R$ genau dann, wenn $(r) \supset (\tilde{r})$ gilt. Man schreibt auch $r \mid \tilde{r}$.
Deshalb gibt es dann auch ein $a \in R$, sodass $a \cdot r = \tilde{r}$ gilt.

(ii) Die Menge der größten gemeinsamen Teiler zweier Elemente $r, \tilde{r}$ ist die Menge $\mathrm{GGT}(r, \tilde{r})$ der Erzeuger des kleinsten Ideals, das r und $\tilde{r}$ enthält. Ein solcher Erzeuger lässt sich durch die Funktion ggt aus dem obigen Lemma bestimmen, also ist $\mathrm{GGT}(r, \tilde{r}) = \{e \cdot \mathrm{ggt}(r, \tilde{r}); e \in R^*\}$ und

[39] Vergleiche auch [Bosch, 2004, Seite 34 ff.].

[40] $(\mathrm{ggt}(i_1, i_2))$ bezeichnet hier im Moment nur die Funktion, die aus den Erzeugern i_1, i_2 zweier Hauptideale ι_1, ι_2 einen Erzeuger des kleinsten Hauptideals ι_{12} zusammensetzt, das i_1, i_2 enthält.

damit folgt auch $\text{ggt}(r, \tilde{r}) \mid r$, sowie $\text{ggt}(r, \tilde{r}) \mid \tilde{r}$.
Deshalb gibt es dann auch ein $a \in R$, sodass $a \cdot \text{ggt}(r, \tilde{r}) = r$ gilt, und ein $\tilde{a} \in R$, sodass $\tilde{a} \cdot \text{ggt}(r, \tilde{r}) = \tilde{r}$ gilt.

Aus dem obigen Lemma und dieser Definition folgt nun:

Korollar 4.42

Sei R ein Hauptidealring und seien $i_1, i_2, i_3 \in R$, dann gilt $\text{ggt}(i_1, \text{ggt}(i_2, i_3)) = \text{ggt}(i_3, \text{ggt}(i_1, i_2))$.

Beweis: Es gilt der folgende Zusammenhang:

$$\begin{aligned}(\text{ggt}(i_1, \text{ggt}(i_2, i_3))) &= (i_1) + (\text{ggt}(i_2, i_3)) = \{r \cdot i_1 + \tilde{r} \cdot \text{ggt}(i_2, i_3); r, \tilde{r} \in R\} \\ &= \{r \cdot i_1 + \tilde{r} \cdot i_2 + \bar{r} \cdot i_3; r, \tilde{r}, \bar{r} \in R\} = \{r \cdot \text{ggt}(i_1, i_2) + \bar{r} \cdot i_3; r, \bar{r} \in R\} \\ &= (\text{ggt}(i_3, \text{ggt}(i_1, i_2)))\,.\end{aligned}$$

□

Lemma 4.43

Jeder Hauptidealring R ist noethersch. Das heißt, dass jede aufsteigende Kette von Idealen $\iota_1 \subset \iota_2 \subset ... \subset R$ stationär wird. Es gibt also ein $n \in \mathbb{N}$ mit $\iota_j = \iota_n \ \forall j \geq n$.

Beweis: Da die Vereinigung einer aufsteigenden Kette von Idealen wieder Ideale ergibt, kann man ein Ideal $\iota := \cup_{j \geq 1} \iota_j$ zusammensetzen. Weil R ein Hauptidealring ist, gibt es ein i, sodass gilt $(i) = \iota$. $i \in \iota$, also muss es ein n und damit ein ι_n geben, sodass $i \in \iota_n$. Es folgt:

$$(i) \subset \iota_n \subset \iota = (i).$$

Damit wird die Idealkette $\iota_1 \subset \iota_2 \subset ...$ bei ι_n stationär.[41] □

4.4.5 Hauptsatz über die Invertierbarkeit von Matrizen über einem Hauptidealring

Satz 4.44

Sei R ein Hauptidealring, $A \in M(m \times n; R)$, dann existieren invertierbare Matrizen $P \in GL(n; R)$ und $Q \in GL(m; R)$, sodass

$$P \circ A \circ Q = \begin{pmatrix} d_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & d_r & \\ & 0 & & 0 \end{pmatrix},$$

[41] Vergleiche auch [Bosch, 2004, Seite 47].

wobei $d_1 \mid d_2 \mid ... \mid d_{r-1} \mid d_r$.

Beweis: A wird durch sukzessive Zeilen- und Spaltenoperationen vereinfacht werden. Dabei kommen folgende Spaltenoperationen vor:

(1) Das Vertauschen zweier Spalten i und j: P_i^j

(2) Die Multiplikation der i-ten Spalte mit einer Einheit von R: $S_i(\lambda)$, wobei $\lambda \in R^*$

(3) Die Addition des λ-fachen der j-ten Spalte zur i-ten Spalte: $Q_i^j(\lambda)$, wobei $\lambda \in R$

(4) Die Veränderung zweier Spalten i, j auf die Weise, dass die beiden Einträge in der ersten Zeile und in der i-ten bzw. j-ten Spalte, u, v, in der i-ten Spalte durch ein Element der Menge ihrer größten gemeinsamen Teiler und in der j-ten Spalte durch Null ersetzt werden: $G_i^j(u, v)$, wobei $u, v \in R, u \neq 0$

Die Vorarbeit macht sich jetzt bezahlt, allein Operation (4) ist noch unbekannt und muss genauer untersucht werden:
Sei $A = (u, v), u, v \in R$ mit $u \neq 0$, eine 1×2 Matrix, $t \in \mathrm{GGT}(u, v)$. Gesucht ist eine Matrix $G(u, v)$, sodass $(u, v) \circ G(u, v) = (t, 0)$ gilt.
Weil R ein Hauptidealring ist, gilt $(t) = (\mathrm{ggt}(u, v)) = (u) + (v)$ und es existieren a, b, c, d mit $t = u \cdot d - v \cdot c, u = t \cdot a, v = t \cdot b$. Also gilt auch $t \neq 0$.
Es gilt dann:

$$t = t \cdot a \cdot d - t \cdot b \cdot c = t \cdot (a \cdot d - b \cdot c).$$

Und weil R als Hauptidealring auch ein Integritätsring ist, folgt mit $t \neq 0$:

$$1 = a \cdot d - b \cdot c.$$

Die Matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in M(2 \times 2, R)$ ist also über R invertierbar; ihre Inverse ist:

$$\tilde{G}_1^2(u, v) := \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Diese leistet, wie gewünscht:

$$(u, v) \circ \tilde{G}_1^2(u, v) = (u \cdot d - v \cdot c, -u \cdot b + v \cdot a) = (t, 0).$$

Bei beliebigen Matrizen $A = (a_{ij}) \in M(m \times n; R)$ mit $a_{11} = u, a_{12} = v$ kann man A mit der Matrix

$$G_1^2(u, v) = \begin{pmatrix} \tilde{G}_1^2(u, v) & 0 \\ 0 & E_{n-2} \end{pmatrix}$$

von rechts multiplizieren und man erhält den gewünschten Effekt in den ersten beiden Spalten. Sobald Spalten vertauscht werden, erhält man die Matrix für Operation (4) für zwei beliebige Spalten i, j: $G_i^j(u, v)$. Allerdings ist es nicht so einfach zu kontrollieren, was in den anderen Zeilen, nämlich jene, in denen u, v nicht stehen, passiert.

Die Operationen (1) bis (4) werden nun angewendet, um die Matrix A auf die gewünschte Gestalt zu bringen:
Im Nachfolgenden verhält es sich so, dass alle Operationen, die mit von links multiplizierten Matrizen vorgenommen werden, zusammen die Matrix P bilden. Analog bilden alle Operationen, die mit von rechts multiplizierten Matrizen vorgenommen werden, zusammen die Matrix Q. Da die Matrizen der Operationen (1) bis (4) invertierbar über R sind, sind auch ihre Produkte über R invertierbar, denn die Elemente in den Produktmatrizen sind ebenfalls aus R und nach den Eigenschaften der Determinantenfunktion ist die Determinante der Produktmatrix eine Einheit von R.

Ist $A = 0$, so ist nichts zu zeigen.
Ist $A \neq 0$, so kann man nach Zeilen- und Spaltenvertauschungen $a_{11} \neq 0$ annehmen. a_{11} und a_{12} werden nun durch $\text{ggt}(a_{11}, a_{12}) =: \tilde{a}_{11}$ und Null ersetzt, indem Operation (4) angewendet wird.
Dann werden $\tilde{a}_{11}$ und a_{13} durch $\text{ggt}(\tilde{a}_{11}, a_{13})$ und Null ersetzt, indem wieder Operation (4) angewendet wird. So erhalten wir nach endlich vielen Schritten, dass, abgesehen vom ersten Eintrag, nur noch Nullen in der ersten Zeile stehen:

$$\begin{pmatrix} a'_{11} & 0 & \cdots & 0 \\ * & & & \\ \vdots & & A' & \\ * & & & \end{pmatrix}$$

Auf diese Matrix werden nun in analoger Art und Weise Zeilenoperationen angewandt. Sie liefern demnach die Matrix

$$\begin{pmatrix} a''_{11} & * & \cdots & * \\ 0 & & & \\ \vdots & & A'' & \\ 0 & & & \end{pmatrix}.$$

Das Verfahren wird wiederholt, die Einträge $a_{11}, a'_{11}, a''_{11}, ...$ erfüllen dabei:

$$(a_{11}) \subset (a'_{11}) \subset (a''_{11}) \subset ... \ .$$

Da R als Hauptidealring auch ein noetherscher Ring ist, wird diese Kette schließlich stationär, das heißt, alle Elemente in der ersten Zeile bzw. der

ersten Spalte sind Vielfache von $a^{(i)}{}_{11}$.
Mit Operationen vom Typ (3) erhält man dann eine Matrix der folgenden Gestalt:

$$\tilde{A} = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \bar{A} & \\ 0 & & & \end{pmatrix}.$$

Auf $\bar{A}$ kann man Induktion anwenden und erhält $\bar{P}, \bar{Q}$ mit:

$$\bar{P} \circ \bar{A} \circ \bar{Q} = \begin{pmatrix} d_2 & & 0 & \\ & \ddots & & 0 \\ 0 & & d_r & \\ & 0 & & 0 \end{pmatrix}.$$

$P = \left(\begin{smallmatrix} 1 & 0 \\ 0 & \bar{P} \end{smallmatrix}\right)$, $Q = \left(\begin{smallmatrix} 1 & 0 \\ 0 & \bar{Q} \end{smallmatrix}\right)$ liefern dann:

$$P \circ \tilde{A} \circ Q = \begin{pmatrix} d_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & d_r & \\ & 0 & & 0 \end{pmatrix}.$$

Nun muss noch $d_1 \mid d_2 \mid ... \mid d_{r-1} \mid d_r$ erreicht werden:
Im 2×2 Fall nimmt man die folgenden Umformungen vor:

$$\begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix} \overset{(3)}{\to} \begin{pmatrix} d_1 & d_2 \\ 0 & d_2 \end{pmatrix} \overset{(4)}{\to} \begin{pmatrix} \tilde{d} & 0 \\ \alpha \cdot d_2 & \beta \cdot d_2 \end{pmatrix} \overset{(3)}{\to} \begin{pmatrix} \tilde{d} & 0 \\ 0 & \beta \cdot d_2 \end{pmatrix},$$
$$\text{wobei } \tilde{d} \in \mathrm{GGT}(d_1, d_2).$$

Mit ähnlichen Operationen kann man erreichen, dass d_1 zuerst alle Elemente $d_2, ..., d_r$ teilt, denn mit den Abkürzungen $g_{12} := \mathrm{ggt}(d_i, d_j)$, $g_{ijk} := \mathrm{ggt}(\mathrm{ggt}(d_i, d_j), d_k)$, ... gilt:

$$\begin{pmatrix} d_1 & 0 & 0 & \cdots & 0 & \\ 0 & d_2 & 0 & & 0 & \\ 0 & 0 & d_3 & & 0 & \\ \vdots & & & \ddots & \vdots & 0 \\ 0 & 0 & 0 & \cdots & d_r & \\ & & 0 & & & 0 \end{pmatrix}$$

$$\overset{(4)}{\to} \begin{pmatrix} g_{12} & 0 & 0 & \cdots & 0 & \\ 0 & \frac{d_1}{g_{12}} \cdot d_2 & 0 & & 0 & \\ 0 & 0 & d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & d_r & \\ & & 0 & & & 0 \end{pmatrix} = \begin{pmatrix} g_{12} & 0 & 0 & \cdots & 0 & \\ 0 & x_2 \cdot d_2 & 0 & & 0 & \\ 0 & 0 & d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & d_r & \\ & & 0 & & & 0 \end{pmatrix}$$

$$\xrightarrow{(4)} \begin{pmatrix} g_{123} & 0 & 0 & \cdots & 0 & \\ 0 & x_2 \cdot d_2 & 0 & & 0 & \\ 0 & 0 & \frac{g_{12}}{g_{123}} \cdot d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & d_r & \\ & & 0 & & & 0 \end{pmatrix} = \begin{pmatrix} g_{123} & 0 & 0 & \cdots & 0 & \\ 0 & x_2 \cdot d_2 & 0 & & 0 & \\ 0 & 0 & x_3 \cdot d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & d_r & \\ & & 0 & & & 0 \end{pmatrix}$$

$$\xrightarrow{(4)} \begin{pmatrix} g_{12\ldots r} & 0 & 0 & \cdots & 0 & \\ 0 & x_2 \cdot d_2 & 0 & & 0 & \\ 0 & 0 & x_3 \cdot d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & x_r \cdot d_r & \\ & & 0 & & & 0 \end{pmatrix}$$

mit $x_i \in R \, \forall i \in \{2, ..., n\}$.

Nun wird das gleiche Verfahren mit den Diagonalelementen von Zeile 2 bis Zeile r durchgeführt: Dabei erhält man zuerst eine Matrix der folgenden Gestalt

$$\begin{pmatrix} g_{12\ldots r} & 0 & 0 & \cdots & 0 & \\ 0 & f_{23} \cdot g_{23} & 0 & & 0 & \\ 0 & 0 & \frac{x_2 \cdot d_2}{f_{23} \cdot g_{23}} \cdot x_3 \cdot d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & x_r \cdot d_r & \\ & & 0 & & & 0 \end{pmatrix},$$

wobei $f_{23} \cdot g_{23} := \text{ggt}(x_2 \cdot d_2, x_3 \cdot d_3)$. Dabei ist $f_{23} \in R$, weil das Element g_{23} die Elemente $x_2 \cdot d_2$ und $x_3 \cdot d_3$ teilt. Führt man diese Schritte bis zum r-ten Eintrag durch, dann sieht die Matrix folgendermaßen aus

$$\begin{pmatrix} g_{12\ldots r} & 0 & 0 & \cdots & 0 & \\ 0 & f_{23\ldots r} \cdot g_{23\ldots r} & 0 & & 0 & \\ 0 & 0 & \frac{x_2 \cdot d_2}{f_{23} \cdot g_{23}} \cdot x_3 \cdot d_3 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & \frac{f_{23\ldots(r-1)} \cdot g_{23\ldots(r-1)}}{f_{23\ldots r} \cdot g_{23\ldots r}} \cdot x_r \cdot d_r & \\ & & 0 & & & 0 \end{pmatrix},$$

wobei $f_{23\ldots i} \cdot g_{23\ldots i} := \text{ggt}(f_{23\ldots(i-1)} \cdot g_{23\ldots(i-1)}, x_i \cdot d_i)$ gilt. Damit teilt $g_{12\ldots r}$ $f_{23\ldots r} \cdot g_{23\ldots r}$ und man erhält man bei Fortführung des Verfahrens induktiv eine Matrix der gewünschten Gestalt.[42] $\square$

Bemerkung 4.45

Es gilt übrigens, wie im Beweis festgestellt:

$$G_i^j(u, v) = G_i^j(a, b, c, d),$$

wobei $u, v \in R$ mit $u \neq 0$ und $a, b, c, d \in R$ mit $a \cdot d - b \cdot c = 1$.

[42] Vergleiche auch [Schreyer, 1992].

Schön wäre nun noch ein Algorithmus zur Bestimmung von a, b, c, d direkt aus u, v. Diesen Algorithmus stelle ich im nächsten Unterabschnitt vor, doch zunächst sichere ich mit dem folgenden Satz das wichtigste Ergebnis dieses Kapitels.

Satz 4.46

Sei A eine über einem Hauptidealring R invertierbare Matrix, dann kann man sie mit Hilfe der obigen vier Operationen (1) bis (4) zusammensetzen. A ist dann ein Produkt der Matrizen P_i^j, $S_i(\lambda), \lambda \in R^*$, $Q_i^j(\lambda), \lambda \in R$ und $G_i^j(u, v), u, v \in R\backslash\{0\}$.
Das heißt, diese Matrizen erzeugen die $GL(n; R)$, sie sind also Elementarmatrizen über dem Hauptidealring R.

Beweis: Da A eine invertierbare Matrix mit Einträgen aus R ist, gilt nach dem vorangegangenen Satz:

$$D = P \circ A \circ Q \text{ mit } D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & & \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & d_n \end{pmatrix},$$

wobei $d_1 \mid d_2 \mid ... \mid d_{n-1} \mid d_n$ und P, Q über R invertierbar. Damit folgt also:

(i)
$$A = P^{-1} \circ D \circ Q^{-1},$$

(ii)
$$\det(D) \in R^* \Rightarrow D \text{ ist über } R \text{ invertierbar.}$$

Weil die Einheiten eines Rings eine multiplikative Gruppe bilden, $d_1 \mid d_2 \mid ... \mid d_{n-1} \mid d_n$ und R ein Hauptidealring ist, gilt:

$$\det(D) = \prod_{i=1}^{n} d_i \in R^* \Rightarrow d_i \in R^* \ \forall i \in \{1, ..., n\}.$$

Die Multiplikation einer Zeile bzw. Spalte einer Matrix mit Einträgen aus R mit einer Einheit von R geschieht allerdings ebenfalls mit einer über R invertierbaren Matrix, nämlich $S_i(\lambda), \lambda \in R^*$. Deshalb sind nach spätestens n weiteren Zeilen- oder Spaltenoperationen alle d_i gleich 1 und es gilt:

$$A = P^{-1} \circ D \circ Q^{-1} = \tilde{P}^{-1} \circ E_n \circ \tilde{Q}^{-1} = \tilde{P}^{-1} \circ \tilde{Q}^{-1}.$$

Mit P, Q über R invertierbar sind natürlich auch P^{-1}, Q^{-1} und damit auch A über R invertierbar und es gilt:

$$A^{-1} = \tilde{Q} \circ \tilde{P}.$$

Also kann man jede über einem Hauptidealring R invertierbare Matrix mit diesen vier Typen von Matrizen zusammenbauen und jede mit diesen über eine Matrixmultiplikation zusammengebaute Matrix ist über R invertierbar. P_i^j, $S_i(\lambda), \lambda \in R^*$, $Q_i^j(\lambda), \lambda \in R$ und $G_i^j(u,v), u, v \in R$ mit $u \neq 0$ sind also Elementarmatrizen des Hauptidealringes R. □

4.4.6 Der euklidische Algorithmus

Definition 4.47

Ein Integritätsring R mit der Abbildung $\delta : R\backslash\{0\} \to \mathbb{N}$ heißt ein euklidischer Ring, wenn es zu beliebigen Elementen $f, g \in R, g \neq 0$, Elemente $q, r \in R$ gibt mit $f = q \cdot g + r$, wobei $\delta(r) < \delta(g)$ oder $r = 0$.
Die Abbildung δ nennt sich Grad- oder Normabbildung des euklidischen Ringes R.[43]

Lemma 4.48

Jeder euklidische Ring R ist ein Hauptidealring.

Beweis: Sei ι ein Ideal und $i \in \iota$, dann gilt schon einmal $(i) \subset \iota$. Unter den Elementen von $\iota\backslash\{0\}$ wird eines gewählt, das unter der Normabbildung von R minimal wird. Sei i dieses Element. Sei weiterhin $f \in \iota$, dann gibt es nach den Eigenschaften eines euklidischen Ringes $q, r \in R$, sodass $f = qi + r$, wobei $\delta(r) < \delta(i)$ oder $r = 0$. Dann ist $r = f - qi \in \iota$. Allerdings war i ein Element aus ι, das unter δ minimal werden sollte, was nur die Folgerung $r = 0$ zulässt. Es gilt also $f = qi$ und somit $\iota \subset (i)$. Damit ist R ein Hauptidealring.[44] □

Bemerkung 4.49

Sei R ein euklidischer Ring, seien $f, q, r_1, r_2 \in R, g \neq 0$, und gelte $f =$

[43] Vergleiche auch [Bosch, 2004, Seite 44].
[44] Vergleiche auch [Bosch, 2004, Seite 44].

$r_1 \cdot g + r_2$, dann existieren $r_3, ..., r_{n+1}, q_1, ...q_n \in R$ und es gilt:

$$\begin{aligned} f &= q \cdot r_1 + r_2 \\ r_1 &= q_1 \cdot r_2 + r_3 \\ r_2 &= q_2 \cdot r_3 + r_4 \\ &\vdots \\ r_{n-1} &= q_{n-1} \cdot r_n + r_{n+1} \\ r_n &= q_n \cdot r_{n+1} \end{aligned}$$

Wegen $\delta(r_1) > \delta(r_2) > ... > \delta(r_n + 1)$ und $\delta(r_i) \in \mathbb{N}\, \forall i \in \{1, ..., n+1\}$ bricht das Verfahren nach endlich vielen Schritten ab.
Dieses Verfahren heißt euklidischer Algorithmus.

Satz 4.50

(i) Seien $u, v \in R$, $u \neq 0$, R ein euklidischer Ring mit der Normabbildung δ, dann bestimmt der euklidische Algorithmus ein $t \in \mathrm{GGT}(u, v)$ und gibt Werte $a, b, c, d \in R$ an, sodass $t = u \cdot d - v \cdot c$, $u = t \cdot a$ und $v = t \cdot b$ gilt.

(ii) $\mathbb{Z}$ ist ein euklidischer Ring.

Beweis:

(i) Seien $u, v \in R$, dann gibt es nach Voraussetzung $q, v_2 \in R$ mit $u = q \cdot v + v_2$ und $\delta(v_2) < \delta(v_1)$. Man führt nun den euklidischen Algorithmus bis zum Ende durch und erhält die folgenden Zeilen:

$$\begin{aligned} u &= q \cdot v + v_2 \\ v &= q_1 \cdot v_2 + v_3 \\ v_2 &= q_2 \cdot v_3 + v_4 \\ &\vdots \\ v_{n-1} &= q_{n-1} \cdot v_n + v_{n+1} \\ v_n &= q_n \cdot v_{n+1} \end{aligned}$$

Nun gilt:

$$\begin{gathered} (v_{n+1}) \supset (v_n) \Rightarrow (v_{n+1}) \supset (v_{n-1}) \Rightarrow ... \\ \Rightarrow (v_{n+1}) \supset (v_2) \Rightarrow (v_{n+1}) \supset (v) \Rightarrow (v_{n+1}) \supset (u) \\ \Rightarrow u, v \in (v_{n+1}) \Rightarrow (u) + (v) \subset (v_{n+1}) \end{gathered}$$

Es bleibt nur noch die umgekehrte Inklusion zu zeigen, also

$$(u) + (v) \supset (v_{n+1}).$$

Aus den Gleichungen des euklidischen Algorithmus gewinnt man nach längerer Rechnung:

$$\begin{gathered} v_2 = u - q \cdot v \\ v_3 = v - q_1 \cdot v_2 = (1 - q \cdot q_1) \cdot v - q_1 \cdot u = \alpha_3 \cdot u + \beta_3 \cdot v \\ v_4 = v_2 - q_2 \cdot v_3 = ... = \alpha_4 \cdot u + \beta_4 \cdot v \\ \vdots \\ v_{n+1} = v_{n-1} - q_{n-1} \cdot v_n = \alpha_{n+1} \cdot u + \beta_{n+1} \cdot v, \\ \alpha_i, \beta_i \in R \, \forall i \in \{3, ..., n+1\} \\ \Rightarrow (v_{n+1}) \subset (u) + (v) \end{gathered}$$

Damit folgt $(v_{n+1}) = (u) + (v)$, also $v_{n+1} \in \mathrm{GGT}(u, v)$.
Identifiziere nun t mit v_{n+1}, d mit α_{n+1}, $-c$ mit β_{n+1}, a mit $\frac{u}{v_{n+1}}$ und b mit $\frac{v}{v_{n+1}}$, denn wegen $(v_{n+1}) \supset (v)$ und $(v_{n+1}) \supset (u)$ gibt es a, b, sodass $u = a \cdot v_{n+1}, v = b \cdot v_{n+1}$.

(ii) Die Multiplikation in $\mathbb{Z}$ ist kommutativ und der einzige Nullteiler von $\mathbb{Z}$ ist Null selber, also ist $\mathbb{Z}$ ein Integritätsring. Die Normabbildung δ wird mit der Betragsfunktion identifiziert und aufgrund der Division mit Rest über $\mathbb{Z}$ findet man mit den folgenden Vorschriften zu jedem Paar $f, g \in R$, $g \neq 0$, Elemente $q, r \in R$, sodass $f = q \cdot g + r$ gilt, sowie $|r| < |g|$ oder $r = 0$.

$$\begin{gathered} q := \left[\frac{f}{g}\right] \in R, \; r := \left(\frac{f}{g} - q\right) \cdot g \in R \\ \Rightarrow |r| < |g| \text{ oder } r = 0 \end{gathered}$$

[45]

□

Bemerkung 4.51

Damit gelten die Ergebnisse dieses Abschnittes insbesondere für $\mathbb{Z}$, den Ring der ganzen Zahlen:

[45] Hier bezeichnen die eckigen Klammern die Gaußklammern.

(i) Eine beliebige Matrix $A \in M(n \times n; \mathbb{Z})$ ist über $\mathbb{Z}$ invertierbar, wenn $\det(A) = \pm 1$, denn $\mathbb{Z}^* = \{\pm 1\}$.

(ii) Die folgenden Matrizen erzeugen $GL(n; \mathbb{Z})$, die Gruppe der über $\mathbb{Z}$ invertierbaren Matrizen:

$$P_i^j,$$
$$S_i(\lambda), \lambda \in \{\pm 1\},$$
$$Q_i^j(\lambda), \lambda \in \mathbb{Z},$$
$$G_i^j(u, v), u, v \in \mathbb{Z}, u \neq 0.$$

Sie sind also Elementarmatrizen über $\mathbb{Z}$.

Die wichtigste Erkenntnis ist also, dass man die Gruppe der über $\mathbb{Z}$ invertierbaren Matrizen mit Hilfe von vier Elementarmatrizen erzeugen kann, wobei man die Einträge für $G_i^j(u, v)$ aus dem euklidischen Algorithmus gewinnt. Und genau auf diese Weise arbeitet auch mein Programm, indem es diese Eigenschaft ausnutzt, eine über $\mathbb{Z}$ invertierbare Matrix und ihre Inverse zu finden.

4.5 Programmablauf

Das Prinzip, eine über $\mathbb{Z}$ invertierbare Matrix zu finden, ist einfach. Das Programm beginnt mit zwei gleich großen quadratischen Matrizen L und R. Zuerst sind dies die Einheitsmatrizen der vorgegebenen Größe. Per Zufall wird eine der vier Elementarmatrizen der passenden Größe über $\mathbb{Z}$ gewählt und über eine Berechnungsvorschrift sofort deren Inverse berechnet. Die betroffenen Zeilen bzw. Spalten und alle anderen variablen Einträge in den Elementarmatrizen werden ebenfalls per Zufall gewählt. Die Grenzen der Intervalle, aus denen die Zufallszahlen gewählt werden, sind, wenn sie nicht von der Größe der Matrizen direkt vorgegeben sind, die Zeilenanzahl z als obere bzw. $-z$ als untere Grenze oder sind anderweitig von z abhängig.[46] Die gewählte Elementarmatrix wird nun von links an L heranmultipliziert, ihre Inverse von rechts an R. Dieses Verfahren wiederholt sich für jede Aufgabe so oft, wie ebenfalls durch die Zeilenanzahl vorgegeben, sodass die dann entstandenen L und R im Allgemeinen keine allzu triviale Gestalt mehr haben. Bei vielen Aufgabentypen geschieht mit L und R nun Folgendes. Im Anschluss an ihre Bildung wird L von links an eine Startmatrix A und R von rechts an das erhaltene Matrizenprodukt heranmultipliziert. Das Ergebnis

[46] Eine genauere Betrachtung erlaubt der Quelltext des Programms im Anhang.

ist die Ergebnismatrix B. Start- und Ergebnismatrix sind in diesem Fall aufgrund der Definition von ähnlichen Matrizen ähnlich über $\mathbb{Z}$.
Das Programm überprüft nun in einigen Fällen noch, ob die Beträge aller Einträge in L, R und der Ergebnismatrix kleiner sind als der angegebene Maximalbetrag und ob es diese Aufgabe nicht schon einmal produziert hat. Sollte einer der Fälle eintreten, dann sortiert es die Aufgabe aus, ansonsten schreibt sie sie in den Zwischenspeicher und erzeugt die nächste Aufgabe, sofern noch mehr erzeugt werden sollen. Zum Schluss werden die Aufgaben mit den gewählten Methoden ausgegeben.

Nun ist noch zu prüfen, ob die gewählte Vorschrift zur Erzeugung der Inversen der Elementarmatrizen Matrix stimmt:

Lemma 4.52
Sei $A_i^j \in M(n;K)$, K ein Körper, $i,j \in \{1,...,n\}$, $i \neq j$, $a_{ii}, a_{ij}, a_{ji}, a_{jj} \in K$ mit $a_{ii} \cdot a_{ij} - a_{ij} \cdot a_{ji} = 1$, eine Matrix der folgenden Gestalt.

$$A_i^j := \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & 0 & \\ & & 1 & & & & & & & \\ & & & a_{ii} & 0 & \cdots & 0 & a_{ij} & & \\ & & & 0 & 1 & & & 0 & & \\ & & & \vdots & & \ddots & & \vdots & & \\ & & & 0 & & & 1 & 0 & & \\ & & & a_{ji} & 0 & \cdots & 0 & a_{jj} & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & 0 & & & & & & & & & 1 \end{pmatrix}$$

Die inverse Matrix zu A_i^j ist dann:

$$\left(A_i^j\right)^{-1} := \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & 0 & \\ & & 1 & & & & & & & \\ & & & a_{jj} & 0 & \cdots & 0 & -a_{ij} & & \\ & & & 0 & 1 & & & 0 & & \\ & & & \vdots & & \ddots & & \vdots & & \\ & & & 0 & & & 1 & 0 & & \\ & & & -a_{ji} & 0 & \cdots & 0 & a_{ii} & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & 0 & & & & & & & & & 1 \end{pmatrix}$$

Beweis: Der Beweis erfolgt durch Nachrechnen. □

Bemerkung 4.53
Mit dieser Formel, die das Programm verwendet, kann man demnach die Inverse jeder Elementarmatrix über $\mathbb{Z}$ berechnen.

Bleibt noch die Frage, ob solche zusammengesetzten Matrizen L und R invers zueinander sind. Sie sind es, wie das folgende Lemma zeigt.

Lemma 4.54
Seien $L_1, L_2, R_1, R_2 \in GL(n;\mathbb{Z})$, $L_1 = (R_1)^{-1}$, $L_2 = (R_2)^{-1}$, dann gilt $(L_2 \circ L_1)^{-1} = R_1 \circ R_2$.

Beweis: Der Beweis folgt direkt aus den Regeln der Matrizenrechung, denn $(L_2 \circ L_1)^{-1} = (L_1)^{-1} \circ (L_2)^{-1} = R_1 \circ R_2$. $\square$

Zu guter letzt fehlt von vielen Aufgaben aus Kapitel 3 noch der Beweis, dass das Programm auch alle möglichen Aufgaben finden kann.

Satz 4.55
Zu einem festen $A \in M(n \times n;\mathbb{Z})$ lässt sich jede zu A über $\mathbb{Z}$ ähnliche Matrix finden.

Beweis: Nach Bemerkung 3.19 reicht es zu zeigen, dass man jedes Paar von zueinander invertierbaren Matrizen, die nur Einträge aus $\mathbb{Z}$ haben, finden kann.
Der Algorithmus setzt $P \in GL(n;\mathbb{Z})$ als Produkt von Elementarmatrizen über $\mathbb{Z}$ zusammen. Die $GL(n;\mathbb{Z})$ wird von diesen Elementarmatrizen erzeugt, also findet man jedes $P \in GL(n;\mathbb{Z})$ und wegen des vorangegangenen Lemmas auch jedes Paar von zueinander inversen Matrizen aus $GL(n;\mathbb{Z})$. $\square$

Da mein Programm immer mit dem Setzen einer Ausgangsmatrix beginnt und ein dazu unabhängiges Paar von über $\mathbb{Z}$ zueinander inversen Matrizen erzeugt, ist klar, dass dabei alle Paare dieser Matrizen erzeugt werden können, wie oben gezeigt. Damit ist auch gezeigt, dass das Programm zu jeder Startmatrix alle möglichen Ergebnismatrizen finden kann.

Kapitel 5

Kurzanleitung

5.1 Kurzbeschreibung des Programmablaufs

Mein Aufgabengenerator Agla (**A**ufgaben**g**enerator zur **l**inearen **A**lgebra) kann 25 verschiedene Aufgaben zu verschiedenen Bereichen der linearen Algebra erzeugen, deren Erstellung zu großen Teilen auf den Eigenschaften von ähnlichen Matrizen beruht. Er ist in der Lage jede quadratische Matrix mit ganzzahligen Einträgen zu finden, deren Inverse ebenfalls nur ganzzahlige Einträge hat. Diese Matrizen sind Matrizenprodukte von vier sogenannten Elementarmatrizen, die in Agla integriert sind und mit denen er arbeitet. Indem man diese invertierbaren Matrizen erst aus einem Produkt von Elementarmatrizen berechnet, kann man auch gleich die jeweils zugehörige inverse Matrix bestimmen. Diese Eigenschaft wird bei der Aufgabenerstellung konsequent ausgenutzt.

Die Aufgabenerstellung läuft immer nach dem gleichen Prinzip ab. Agla erzeugt abhängig von der gewählten Option eine oder mehrere Ausgangsmatrizen und baut dann ein zufälliges Paar von zueinander inversen Matrizen der passenden Größe zusammen. Diese zueinander inversen Matrizen werden nun von links beziehungsweise von rechts zu der oder zu den Ausgangsmatrizen multipliziert. Aus den erhaltenen Matrizenprodukten setzt sich dann die Aufgabe zusammen, wobei die Lösung gerade mitberechnet worden ist. Auf diesem Prinzip basieren alle Aufgaben.

Beispiel 5.1
Agla beginnt mit der Matrix A, zum Beispiel

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

Per Zufall wählt er sich nun eine Elementarmatrix L aus. Hier soll

$$L = \begin{pmatrix} 1 & -3 \\ 0 & 1 \end{pmatrix}$$

betrachtet werden. Die Inverse R zu L wäre dann

$$R = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}.$$

Das Produkt wäre dann

$$B = \begin{pmatrix} 1 & -3 \\ 0 & 1 \end{pmatrix} \circ \begin{pmatrix} 4 & -3 \\ 2 & -1 \end{pmatrix} \circ \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -2 & -6 \\ 2 & 5 \end{pmatrix}.$$

Damit gilt, dass A ähnlich zu B ist.

Agla könnte nun hierzu die folgende Aufgabe stellen:
Zeigen Sie, dass die folgenden Matrizen $A, B \in M(n \times n; \mathbb{Z})$ ähnlich sind, indem Sie eine Matrix $P \in GL(n; \mathbb{C})$ angeben, sodass $B = P \circ A \circ P^{-1}$ gilt.

Eine mögliche Lösung wäre hier die Matrix L. Eine mögliche Lösung deshalb, weil auch $-L$ eine Lösung ist und Agla Aufgaben generieren kann, zu denen es unendlich viele verschiedene Lösungen gibt. Die Kontrolle einer solchen Aufgabe muss also durch Überprüfung von $B = P \circ A \circ P^{-1}$ geschehen, genaueres dazu findet man in den vorangegangenen Kapiteln.

5.2 Das Hauptfenster

Im Hauptfenster kann man zwischen den Aufgabenbereichen wählen und die Ausgabeart des Programms festlegen. Ich habe dabei verschiedene Formate berücksichtigt, wobei ich zur Gestaltung von Übungsblättern mit TEX und OKUSON noch die Option auf große bzw. kleine Matrizen gegeben habe. Die Ausgabe geschieht im Textfeld im unteren Bereich.

Das Hauptfenster enthält noch drei Buttons. Der Button „Textfeldinhalt löschen“ tut genau das mit dem Inhalt des unteren Textfeldes. Der Inhalt kann aber glücklicherweise mit dem Button „Textfeldinhalt wieder herstellen“ zurückgeholt werden. Ein weiteres Drücken des Buttons „Textfeldinhalt wiederherstellen“ holt den gerade überschriebenen Inhalt zurück. Allgemein stellt „Textfeldinhalt wieder herstellen“ den alten Zustand des Textfeldes vor der letzten Schreibaktion von Agla, aber nicht vom Anwender im Textfeld wieder her. Trotzdem sollte der Gebrauch beider Buttons nur nach reiflicher Überlegung geschehen, denn trotz dieser Sicherung kann durch hastiges Drücken immer noch der Inhalt, den man behalten wollte, verloren gehen.

Ein Schließen des Hauptfensters beendet auch das Programm.

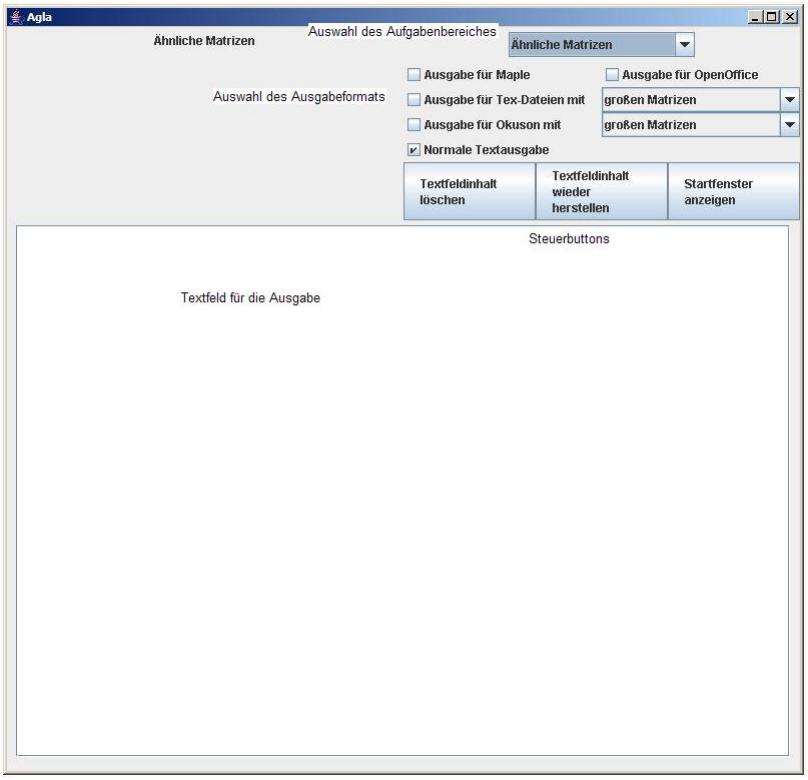

Abbildung 5.1: Hauptfenster von Agla

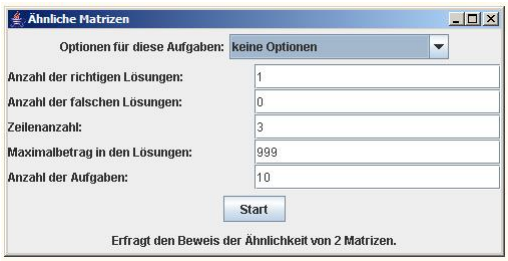

Abbildung 5.2: Startfenster zum Bereich 'Ähnliche Matrizen'

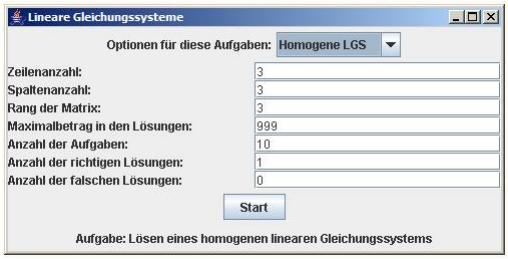

Abbildung 5.3: Startfenster zum Bereich 'Lineare Gleichungssysteme'

5.3 Die Startfenster

Das erste Drücken des Buttons 'Startfenster anzeigen' hat das Öffnen eines Startfensters für den gewählten Aufgabenbereich zur Folge. Erst in diesem Fenster können die genauen Aufgaben angegeben und mit weiteren Optionen versehen werden. Auftretende Fehlermeldungen geschehen im Textfeld des Hauptfensters. Drückt man den Button 'Startfenster anzeigen' ein weiteres Mal, wird das schon offene Fenster wieder in den Vordergrund geholt.
Das Drücken des Buttons 'Start' in den Startfenstern bewirkt, dass Agla versucht, die gegebenen Werte einzulesen. Scheitert er dabei, macht er eine Meldung im Hauptfenster und gibt auch die Fehlerquelle an. In manchen Fällen bewirkt eine Änderung des Aufgabentyps eine Änderung der eingegebenen Werte, in anderen Fällen sieht Agla nur das Erstellen einer richtigen Lösung vor und ignoriert dabei die Angaben in den Feldern. Erst dann erstellt Agla im Hintergrund die Aufgaben, die er, sobald er fertig ist, im Textfeld des Hauptfensters ausgibt. In dieser Zeit ignoriert Agla alle weiteren Befehle. Einzig und allein der Schließbefehl des Hauptfensters funktioniert. Sollte sich Agla in einer Endlosschleife verfangen, was bei unsinnigen Eingaben[1] des

[1]Man könnte die Maximalgrenze ziemlich klein und die Aufgabenanzahl sehr groß wählen. Das hat dann zur Folge, dass Agla ziemlich lange nach Aufgaben suchen wird, wenn es überhaupt welche findet.

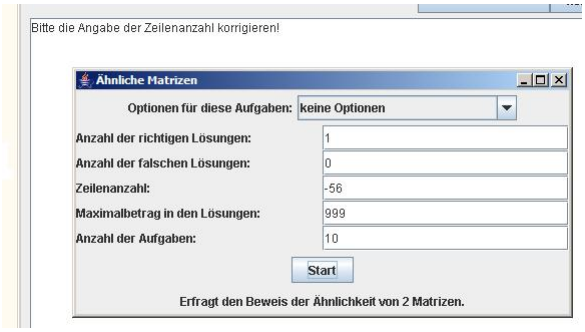

Abbildung 5.4: Beispiel für eine Fehlermeldung

Anwenders der Fall sein kann, ist dies die einzige Möglichkeit zum Abbruch, was im Übrigen den Verlust aller zuvor nicht gesicherten Aufgaben nach sich zieht. Vorsicht ist also angebracht.

Das Aussehen der Fenster von Agla hängt ein wenig vom Betriebssystem ab, sie können in der dort üblichen Art geschlossen werden.

5.4 Die Aufgaben

Die genaue Aufgabenbeschreibung findet man in Kapitel 3 ab Seite 37 und ab Seite 71. Dort kann man auch genau nachlesen, wie die Aufgaben im Einzelnen generiert werden. Zudem findet man zu jeder Aufgabenstellung eine Tabelle, die einen schnellen Überblick verschafft. Mit Hilfe des Tabellenverzeichnisses dieser Arbeit auf Seite 181 kann man diese Überblicktabellen schnell nachschlagen. Zudem enthalten diese auch den Wert, mit dem die Aufgabe beim Aufruf mit Parametern gestartet wird.

5.5 Die Ausgabe

Das Ausgabeformat kann festgelegt werden, das ist besonders im Zusammenhang mit dem Erstellen von Übungsblättern hilfreich. Agla ist auch in der Lage, mehrere Ausgabevarianten zu verarbeiten. Die Aufgaben werden zuerst komplett auf die eine, dann komplett auf die andere Art ausgegeben.

In Abbildung 5.5 ist eine Ausgabe für Maple zu sehen.[2] Man kann ohne große

[2]Maple ist ein Computeralgebraprogramm. Ich benutze zum Zeitpunkt der Erstellung dieser Arbeit die Version Maple 9.01. Weitere Informationen finden sich bei der Auflistung meiner technischen Hilfsmittel.

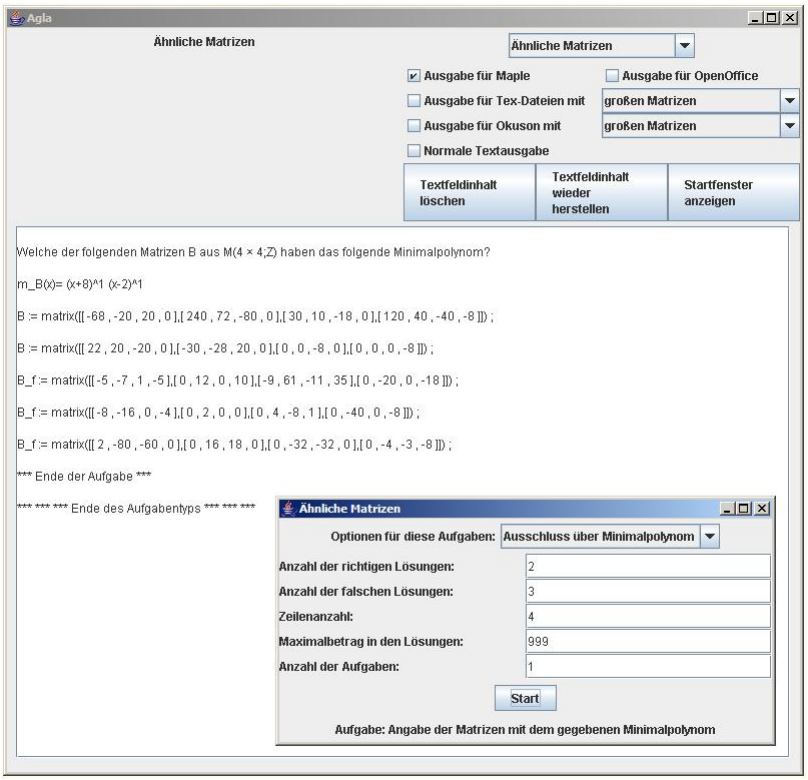

Abbildung 5.5: Beispielausgabe für Maple

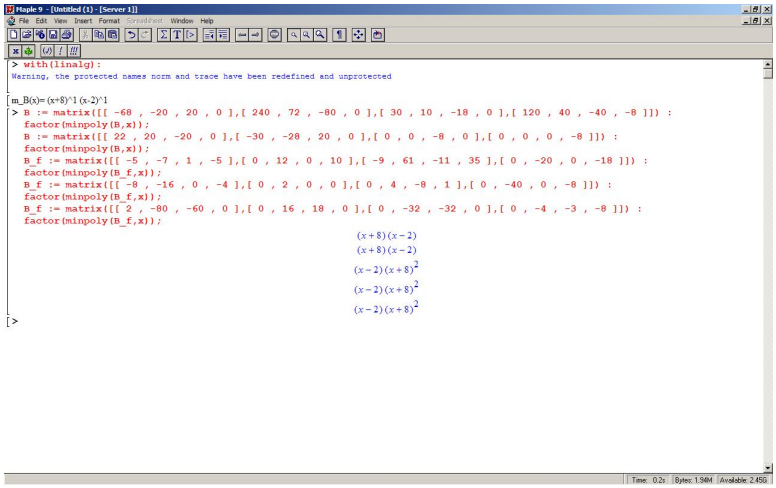

Abbildung 5.6: Kontrolle mit Maple

Probleme eine solche Ausgabe kopieren und die Rechenkünste von Agla mit Maple überprüfen.

Ein Ziel meiner Staatsexamensarbeit war es, Agla mit OKUSON kompatibel zu machen. Bei einigen Aufgabenstellungen scheitert man allerdings daran, dass die Lösungen nicht eindeutig sind. OKUSON müsste zur Überprüfung von Lösungen, Rechenalgorithmen durchführen können. Dazu ist es allerdings zum Zeitpunkt dieser Arbeit nicht in der Lage, sodass diese Kompatibilität eingeschränkt ist. Bei der Mehrzahl der Aufgabenstellungen, die auch in den obigen Tabellen als solche gekennzeichnet sind, kann man aber trotzdem komplette Aufgaben zur Verwendung mit OKUSON erzeugen.

Eine Ausgabe, die nicht unbedingt an ein bestimmtes Programm angepasst ist, ist bei Agla voreingestellt. Man muss das Häkchen im Startfenster entfernen, um diese Ausgabevariante abzustellen.

5.6 Aufruf von Agla

Agla ist mit Java geschrieben worden. Das bedeutet, dass man Agla nur dann starten kann, wenn auf dem Rechner ein möglichst aktueller Interpreter für Javaprogramme installiert ist.

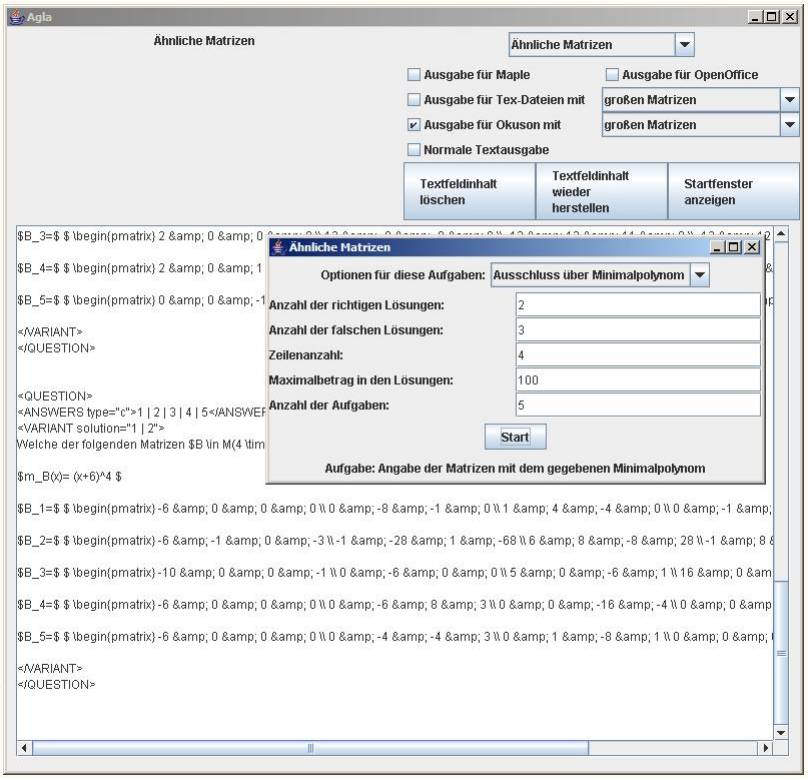

Abbildung 5.7: Ausgabe für OKUSON

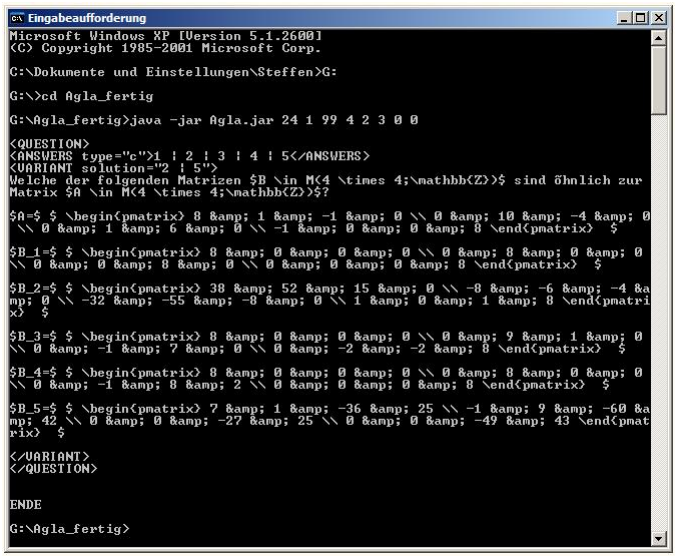

Abbildung 5.8: Ausgabe bei Parameteraufruf

5.6.1 Der Aufruf von Agla ohne Parameter

Der Aufruf selbst hängt vom Betriebssystem ab. Unter Windows kann man die Datei „Agla.jar“ einfach per Doppelklick ausführen, allerdings ohne ihr dabei Parameter zu übergeben. In dem Verzeichnis, in dem die Datei „Agla.jar“ steht, kann man Agla auch mit dem Befehl „java -jar Agla.jar“ starten. Dateien im jar-Format sind eigentlich komprimierte class-Dateien. Hat man diese class-Dateien entpackt, findet sich die Datei „Agla.class“. Agla kann man dann auch in dem Verzeichnis mit allen entpackten class-Dateien mit dem Befehl „java Agla“ starten.
Zudem funktioniert Agla auch als Applet.

5.6.2 Der Parameteraufruf

Agla enthält eine Alternative zum Aufruf der Aufgabengenerierung mit der Ausgabe und den Einstellungen in den Fenstern. Man kann es ebenso mit Parametern aufrufen und damit die Fenster ignorieren. Die Ausgabe geschieht dann in dem Systemfenster, aus dem man Agla auch gerade gestartet hat.
Dieser Parameteraufruf kann damit von anderen Programmen ausgenutzt werden. Allerdings ist er nicht ungefährlich, denn eine falsche Eingabe, die beim Aufruf ohne Parameter von Agla abgefangen wird, bewirkt in diesem Fall entweder eine Endlosschleife oder alternativ einen normalen Start des Programms.
Die Parameter sind an den Aufruf „java -jar Agla.jar“ beziehungsweise „java

Agla“ angehängt, wobei die Parameter mit einem Leerzeichen vom Aufruf von Agla und untereinander zu trennen sind.

Die einzelnen Parameter stehen in genau dieser Reihenfolge für:

- Aufgabenstellung
- Aufgabenanzahl
- Maximalbetrag
- Zeilenanzahl
- Anzahl der richtigen Lösungen
- Anzahl der falschen Lösungen
- Spaltenanzahl
- Rang

Die Tabellen in Kapitel 3 geben Auskunft darüber, welche Angaben wo benötigt werden. Bei jedem Aufruf müssen immer diese acht Parameter angegeben werden, ansonsten funktioniert der Aufruf über Parameter nicht und Agla startet sich normal. Am besten setzt man die nicht benötigten Parameter gleich Null.

Beispiel 5.2

Der Aufruf „java -jar Agla.jar 24 1 99 4 2 3 0 0“ erzeugt eine Ausgabe wie in Abbildung 5.8.

Mit diesem Aufruf erzeugt Agla eine Aufgabe, bei der die zu einer gegebenen Matrix ähnlichen Matrizen angegeben werden sollen. Alle Matrizen sind deshalb quadratisch und haben vier Zeilen. In den Aufgaben kommt keine Zahl mit einem größeren Betrag als 99 vor. Agla erstellt zwei richtige und drei falsche Lösungen. Er ignoriert zudem die Angabe für die Spaltenanzahl und den Rang, weil sie nicht benötigt werden.

Auch der alternative Aufruf von oben kann genutzt werden, wie man in Abbildung 5.9 erkennt. Die Reihenfolge der anzugebenden Parameter ändert sich dabei nicht.

In den Tabellen in Kapitel 3 kann man ablesen, welcher Wert für welche Aufgabenstellung beim Parameteraufruf einzugeben ist. Im Tabellenverzeichnis auf Seite 181 findet man die Seitenzahlen, auf denen die Tabellen zu finden sind.

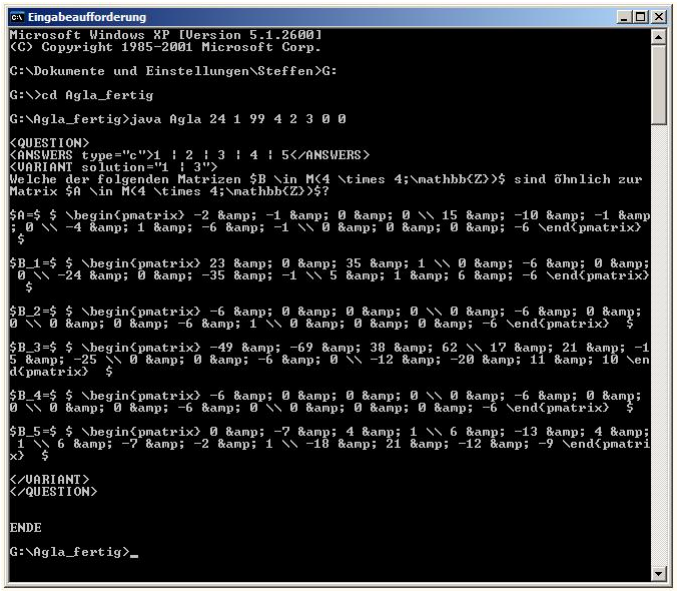

Abbildung 5.9: Ausgabe bei Parameteraufruf

Zwei Warnungen sind hier noch angebracht:

1. Das Eingabefenster des Systems hat in manchen Fällen eine bestimmte Zeilenlänge. Das heißt, dass eine zu lange Ausgabe dann einfach in Teilen verschwindet und so nicht mehr nachprüfbar ist. Der Parameteraufruf war eigentlich für kurze Demonstrationsausgaben vorgesehen; viele Aufgaben lässt man besser im Hauptfenster erzeugen, dessen Zeilenanzahl nicht begrenzt ist.

2. Sollte sich Agla beim Aufruf mit Parametern in einer Endlosschleife verfangen, muss die Umgebung, aus der es gestartet wurde, beendet werden, um das Erstellen von Aufgaben abzubrechen. Die Parameter für einen Aufruf mit Parametern sollten am besten immer vorher über das Hauptfenster getestet werden.

5.7 Probleme

Agla läuft keinesfalls problemfrei. Die Probleme treten besonders dann auf, wenn der Anwender Agla überfordert oder eine Angabe vergessen hat. Hier mal eine Auswahl von Problemen, auf die ich auch eine Antwort parat habe.

Problem 5.1

Agla nimmt die Standardwerte, obwohl nur Zahlen in den Eingabefeldern

stehen.

Antwort: Die kompletten Felder löschen, wahrscheinlich steht irgendwo noch ein Zeichen. Leerzeichen sind hier besonders tückisch, da man sie nicht sieht.

Problem 5.2

Ich hätte die letzte Ausgabe gerne in einem anderen Format. Wie geht das?

Antwort: Das geht im Moment leider noch nicht. Hier kann erst eine spätere Version Abhilfe schaffen. Momentan muss man sich leider noch vorab entscheiden.

Problem 5.3

Agla schreibt einen kurzen einleitenden Text und ansonsten nichts oder einfach gar nichts ins Textfeld.

Antwort: Hier gibt es zwei Möglichkeiten. Entweder ist keine Ausgabevariante aktiviert oder Agla ist vielleicht in einer Endlosschleife gefangen. In diesem Fall muss Agla dann manuell beendet werden. Kontrollieren Sie deshalb immer alle Eingaben vor dem Start der Aufgabengenerierung.
Die Eingaben für Agla musste ich flexibel halten, es kann also sein, dass das Intervall für die Matrizeneinträge zu klein gewählt wurde, sodass das Programm keine oder nicht genügend Aufgaben finden kann. Alternativ könnte es auch weniger mögliche Aufgaben geben, als man erzeugen möchte. Wenn man Agla nicht abbrechen möchte, dann holt man sich am besten etwas zu Trinken und setzt sich vor den Rechner, um zu sehen, ob Agla nicht doch in endlicher Zeit fertig wird.

Problem 5.4

Der Parameteraufruf funktioniert nicht, Agla baut das Hauptfenster auf, anstatt Aufgaben ins Systemfenster zu schreiben.

Antwort: Es ist generell so, dass Agla im Fall des Parameteraufrufs bei einem Fehler gleich welcher Art, das Hauptfenster öffnet. Also gibt es zwei Möglichkeiten.

1. Die Parameterangabe war falsch. Am besten lesen Sie noch einmal nach, wie sie genau funktioniert. Schauen Sie sich insbesondere die obigen Beispiele an.

2. Sie sind das Opfer eines internen Fehlers von Agla geworden. Versuchen Sie mal die gleichen Einstellungen im Hauptfenster. Wenn Agla auch dort eine unsinnige Fehlermeldung ausgibt, dann ist es sogar sehr wahrscheinlich, dass Sie einen Fehler entdeckt haben. Bitte schicken Sie die Einstellungen und die Fehlerbeschreibung an mich, damit ich den Fehler in Agla schnellstens korrigieren kann.

Kapitel 6

Schluss

Nachdem mein Programm mit „Agla“ auch einen Namen hat, komme ich nun auf die Forderungen am Anfang meiner Arbeit zurück.
Es ist natürlich Aufgabe des Anwenders von Agla Lernziele zu definieren und zu entscheiden, wie er deren Erreichen überprüfen möchte. Vorgaben mache ich hier an keiner Stelle. Für mich war wesentlich, dass ich mir überlege, welche Lernziele ein Anwender auf alle Fälle definiert, sodass ich zu diesen Aufgaben erstellen kann. Es ist zudem klar, dass es natürlich mehr Lernziele gibt, als hier dargestellt worden sind. Genauso klar ist, dass man mehr Aufgaben als die vorgestellten zu den in dieser Arbeit definierten Lernzielen finden kann. Ich denke, dass diese Arbeit als Anhaltspunkt dienen kann, wie man weitere Aufgaben besonders mit Agla bauen kann.

Ebenso muss die Gestaltung von Übungen Aufgabe des Anwender sein. Wie gefordert, habe ich mich im Rahmen meiner Möglichkeiten darum bemüht, zu möglichst vielen Bereichen mehrere Aufgabenstellungen anzubieten. Dies sind meistens zwei, eine, in der etwas direkt bestimmt werden soll, eine, bei der man Lösungen ausschließen soll und die richtigen angeben muss. Weiterhin habe ich an genau dieser Stelle für den Ausschluss von Matrizen, die zu einer gegebenen Matrix nicht ähnlich sind, eine Möglichkeit eingebaut, dass man als Aufgabensteller entscheiden kann, an welcher Stelle sich Lösungen als falsch erweisen.
Für eine Berücksichtigung von Standardfehlern müsste man zuerst herausfinden, welche Fehler in Bezug auf welche Aufgabenstellung Standardfehler sind. Das kann ich im Rahmen dieser Arbeit nicht leisten, zumal die Fehlerquellen bei Aufgaben aus den betrachteten Bereichen so komplex sind, dass man nicht mehr unbedingt Standardfehler finden kann. Eine Betrachtung von Standardfehlern kann mit Agla also nur eingeschränkt stattfinden.

Auch wenn die Bearbeitung vielleicht einen anderen Eindruck macht, so be-

stand doch das Hauptproblem darin, einen Algorithmus zu finden, der dafür sorgt, dass man alle Paare von quadratischen Matrizen in einer beliebigen Größe finden kann. Die Aufarbeitung der nötigen Theorie war aus meiner Sicht zwar aufwendig, aber nötig, um darstellen zu können, wie man einen möglichst einfachen Algorithmus bekommt, der den gestellten Anforderungen genügt.
Der gefundene Algorithmus erfüllt meiner Meinung nach die gestellten Anforderungen. Dadurch, dass alle Probleme auf das Finden einer oder mehrerer Startmatrizen und Paaren von invertierbaren Matrizen mit Einträgen in $\mathbb{Z}$ zurückgeführt werden, ergibt sich ein immer gleiches Vorgehen, wobei jedoch die ausgegebenen Aufgaben durchaus unterschiedlich aussehen.

Das Problem der Eindeutigkeit kann ich für viele Aufgaben nur ungenügend lösen, indem ich einen Lösungsalgorithmus fordere. Allerdings bleibt mir nichts anderes übrig, wenn die allgemeine Lösung eines Problems eigentlich ein Vektorraum oder ein affiner Unterraum beziehungsweise eine unendliche Teilmenge davon ist. Eine Alternative wäre, den Studenten, die die Aufgaben lösen sollen, Vorgaben zu machen, wie die Lösung auszusehen hat. Jetzt ist allerdings die Frage, ob die Studenten diese Vorgaben verstehen und in der Lage sind, sie umzusetzen. Vielleicht ist es auch komplizierter eine passende Lösungsangabe zu finden als die Lösung selbst? Dieses Problem umgeht allein ein Nachrechnen der Lösungen als Kontrollalgorithmus, sodass die Eingabe möglichst gleich der errechneten Lösung der Studenten ist.

Erwähnenswert ist noch, dass in manchen Aufgabenstellungen die Ausgangsmatrizen der falschen und in allen Aufgabenstellungen die Ausgangsmatrizen der richtigen Lösungen immer die gleichen sind. Es ist deshalb nicht unbedingt sinnvoll zu viele falsche und richtige Lösungen anzugeben. Dieses Prinzip wird sonst mit Sicherheit schnell erkannt werden, was zur Folge hat, dass man einfach die zur Verfügung gestellten Lösungen auf die schon errechneten Lösungen hin überprüft, um so Arbeit zu sparen. Dies ist aus meiner Sicht eine Schwäche von Agla, kann jedoch auch dazu dienen, die Studenten mit Absicht zu animieren, genau auf solche Ideen zu kommen, wobei man allerdings Wert darauf legen sollte, dass ein solches Prinzip nicht zu häufig anwendbar ist.

Ob die Aufgaben aus Agla nun in der Lage sind, das Erreichen von Lernzielen beobachtbar zu machen oder Übungen flexibel zu gestalten, muss jeder Anwender für sich selbst entscheiden. Ich wünsche jedenfalls viel Erfolg bei der Arbeit mit Agla.

Agla ist sicherlich nicht fertig. Mit dieser Arbeit ist aus meiner Sicht nur ein

Zwischenschritt gemacht. Ich denke, dass ich Agla erweitern werde. Allerdings muss ich dazu noch einmal seine Grundstrukturen verändern, weil diese im Moment nur schwerlich Erweiterungen zulassen und derart kompliziert programmiert sind, dass erfahrene Programmierer daran verzweifeln würden. Mit dieser Erweiterung könnte man auch gleichzeitig einige der dargestellten Probleme bezüglich der Kompatibilität mit OKUSON und den Auswirkungen der einen Aufgabenstellung auf eine andere verhindern. Dafür brauche ich allerdings wesentlich mehr Zeit.
Mit dem in den vier Monaten Erreichten bin ich für meine Begriffe zufrieden. Ich hätte zu Beginn nicht gedacht, dass ich in dieser Zeit so weit kommen würde.

Kapitel 7

Anhang

7.1 Kongruente Matrizen

Mit Aufgaben zu kongruenten Matrizen betritt man den Bereich der Vektorräume mit Skalarprodukt. Ich habe diese Aufgaben im Anhang aufgenommen[1], weil der Generierungsalgorithmus sich nicht so sehr von dem der ähnlichen Matrizen unterscheidet. Anstatt die Matrizen zu invertieren, werden sie hier nur transponiert, mit einem einzigen Befehl erschließen sich komplett neue Aufgabenstellungen, die aber auf die alten Algorithmen zurückgreifen.

7.1.1 Definition und wichtige Eigenschaften

Definition 7.1

(i) Sei V ein Vektorraum über dem Körper K.
Eine Abbildung $\beta : V \times V \to K$ heißt eine Bilinearform, falls sie die folgenden Eigenschaften hat:

(a) $\beta(\vec{x}_1 + \vec{x}_2, \vec{y}_1 + \vec{y}_2) = \beta(\vec{x}_1, \vec{y}_1) + \beta(\vec{x}_1, \vec{y}_2) + \beta(\vec{x}_2, \vec{y}_1) + \beta(\vec{x}_2, \vec{y}_2)$
$\forall \vec{x}_1, \vec{x}_2, \vec{y}_1, \vec{y}_2 \in V$

(b) $\beta(a \cdot \vec{x}, b \cdot \vec{y}) = a \cdot b \cdot \beta(\vec{x}, \vec{y})$
$\forall \vec{x}, \vec{y} \in V, \forall a, b \in K$

β ist also über jedem Argument linear.

(ii) Sei $\beta : V \times V \to K$ eine Bilinearform.

(a) β heißt symmetrisch, falls $\beta(\vec{x}, \vec{y}) = \beta(\vec{y}, \vec{x})$ $\forall \vec{x}, \vec{y} \in V$ gilt.
Eine symmetrische Bilinearform β heißt auch Skalarprodukt auf

[1] Sie sind nicht in Agla integriert. Vielleicht wird Agla im Anschluss an diese Arbeit an genau dieser Stelle weiter ergänzt werden.

V.
Insbesondere ist das kanonische Skalarprodukt, definiert durch $\beta(\vec{x},\vec{y}) = \sum_{i=1}^{n} x_i \cdot y_i$, symmetrisch.

(b) β heißt alternierend, falls $\beta(\vec{x},\vec{y}) = -\beta(\vec{y},\vec{x})\ \forall \vec{x},\vec{y} \in V$ gilt.

(iii) Ist $C = (\vec{c}_1, ..., \vec{c}_n)$ eine Basis von V und $\beta : V \times V \to K$ eine Bilinearform, so heißt die Matrix $B_C(\beta) := (\beta(\vec{c}_i, \vec{c}_j)))$ die Matrix der Bilinearform β bezüglich der Basis C. Wenn $V = K^n$ und C die kanonische Basis ist, wird die Matrix der Bilinearform β mit $B(\beta)$ bezeichnet.

(iv) Ist $B \in M(n \times n; K)$ eine Matrix, dann kann man immer durch $\beta_B : K^n \times K^n \to K; \beta(\vec{x},\vec{y}) = \vec{x}^T \circ B \circ \vec{y} = \sum_{i,j=1}^{n} b_{ij} \cdot x_i \cdot y_j$ eine Bilinearform definieren.

(v) Seien $A_1, A_2 \in M(n \times n; K)$, so heißen A_1, A_2 kongruent, wenn sie bezüglich geeigneter Basen die gleiche Bilinearform β darstellen, es also einen Vektorraum V über K gibt, eine Bilinearform β und Basen C_1, C_2 von V, sodass $A_1 = B_{C_1}(\beta)$ und $A_2 = B_{C_2}(\beta)$

[2]

Bis jetzt ist noch nicht ganz ersichtlich, an welcher Stelle der schon bekannte Algorithmus des Programms hilfreich werden könnte. Abhilfe schaffen folgende wichtige Eigenschaften:

Satz 7.2
Es sei $\dim(V) = n, \beta : V \times V \to K$ *eine Bilinearform,* C *eine Basis von* V*. Seien zudem* $A_1, A_2 \in M(n \times n; K)$*. Unter diesen Voraussetzungen gilt Folgendes:*

(i) β *symmetrisch* $\Leftrightarrow$ $B_C(\beta)$ *symmetrisch*

(ii) β *alternierend* $\Leftrightarrow$ $B_C(\beta)$ *schiefsymmetrisch,* $B_C(\beta) = -(B_C(\beta))^T$

(iii) Die folgenden Aussagen sind äquivalent:

(a) A_1 *ist kongruent zu* A_2

(b) Es existiert $P \in GL(n; K)$ *mit* $A_2 = P \circ A_1 \circ P^T$.

(c) Es gibt eine Basis $\tilde{C}$ *des* K^n *mit* $A_2 = B_{\tilde{C}}(\beta_{A_1})$

(iv) Kongruente Matrizen haben den gleichen Rang.

[2] Vergleiche auch [Holz & Wille, 2002, Seite 198 f.].

[3]

Beweis:

(i)-(iii) Der Beweis folgt jeweils aus der Bijektivität der Abbildung zwischen den Bilinearformen und den Matrizen, wie sie in Punkt (iii) von Definition 7.1 dargestellt ist.

(iv) Die Behauptung folgt direkt aus (*iii*).

□

Die Eigenschaft (*iii*) ist die für den Algorithmus entscheidende. Sie ist analog zur Eigenschaft der ähnlichen Matrizen definiert, die Matrix P muss invertierbar sein, allerdings steht nicht die Inverse, sondern die Transponierte in der Gleichung. Wie schon in der Einleitung dieses Abschnittes angedeutet, reicht also ein Befehl, die Transponierung einer Matrix, aus, um neue Aufgabentypen aus dem gleichen Algorithmus zu gewinnen.
Es bleibt noch zu erwähnen, dass die Kongruenz genau wie die Ähnlichkeit von Matrizen eine Äquivalenzrelation ist, die $M(n \times n; K)$ in Äquivalenzklassen einteilt. In dieser Arbeit will ich nur noch auf die einfachsten Matrizen in diesen Äquivalenzklassen verweisen.

Satz 7.3

Sei K ein Körper mit $1 + 1 \neq 0$, $A \in M(n \times n; K)$ eine symmetrische Matrix, dann ist A kongruent zu einer Matrix in Diagonalgestalt. Diese Diagonalgestalt nennt sich dann eine Normalform der symmetrischen bilinearen Abbildung zu A.

Beweis: Ein Beweis findet sich in [Holz & Wille, 2002, Seite 207 f.].

Satz 7.4

Sei $A \in M(n \times n; K)$ eine schiefsymmetrische Matrix, dann ist A kongruent zu einer Matrix B in der folgenden Gestalt:

$$B = \begin{pmatrix} \boxed{\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}} & & & & \\ & \ddots & & & \\ & & \boxed{\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}} & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{pmatrix}$$

[3] Vergleiche auch [Holz & Wille, 2002, Seite 200].

Für die Anzahl m der Matrizen $\left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$ gilt dabei $m = \frac{1}{2}\operatorname{rang}(A)$.
Eine Matrix in dieser Gestalt nennt sich dann eine Normalform der alternierenden bilinearen Abbildung zu A.[4]

Beweis: Ein Beweis findet sich in [Holz & Wille, 2002, Seite 212 f.].

Die wichtigsten Lernziele, die sich aus dieser knappen Darstellung ergeben, sind die folgenden:

Lernziel 7.1

Die Studenten sollen den Begriff der Kongruenz von Matrizen kennen, um seine Eigenschaften (vor allem in Bezug auf Bilinearformen) wissen, zeigen können, wann zwei Matrizen kongruent sind, und die Eigenschaften dieses Begriffes in verschiedenen Aufgaben ausnutzen können.

Lernziel 7.2

Die Studenten sollen den Begriff der Normalform einer bilinearen Abbildung in Bezug auf symmetrische und alternierende Bilinearformen kennen, um seine Eigenschaften wissen, die Normalform symmetrischer und alternierende bilinearer Abbildungen bestimmen und ihre Eigenschaften in verschiedenen Aufgaben ausnutzen können.

7.1.2 Aufgaben zu kongruenten Matrizen

Aufgabenstellung 7.1

Zeigen Sie, dass die symmetrischen Matrizen $A, B \in M(n \times n; \mathbb{Z})$ über $\mathbb{Z}$ kongruent sind, indem Sie eine Matrix $P \in GL(n; \mathbb{Z})$ angeben, sodass $B = P \circ A \circ P^T$ gilt.

Einstellung im Fenster: „Kongruenz symm. Matrizen“

Das Programm gibt sich eine beliebige Matrix $D \in M(n \times n; \mathbb{Z})$ in Diagonalgestalt vor, zu der es eine kongruente Matrix $A \in M(n \times n; \mathbb{Z})$ zusammensetzt und in einem zweiten Schritt eine zu A kongruente Matrix $B \in M(n \times n; \mathbb{Z})$.

Aufgabenstellung 7.2

Zeigen Sie, dass die schiefsymmetrischen Matrizen $A, B \in M(n \times n; \mathbb{Z})$ über $\mathbb{Z}$ kongruent sind, indem Sie eine Matrix $P \in GL(n; \mathbb{Z})$ angeben, sodass $B = P \circ A \circ P^T$ gilt.

Einstellung im Fenster: „Kongruenz schiefsymm. Matrizen“

[4] Vergleiche auch [Holz & Wille, 2002, Seite 212].

Das Programm gibt sich eine beliebige Matrix $D \in M(n \times n; \mathbb{Z})$ in der in Satz 7.4 dargestellten Gestalt vor, zu der es eine kongruente Matrix $A \in M(n \times n; \mathbb{Z})$ zusammensetzt und in einem zweiten Schritt eine zu A kongruente Matrix $B \in M(n \times n; \mathbb{Z})$ erzeugt.

Aufgabenstellung 7.3

Finden Sie zu der über die Matrix $B \in M(n \times n; \mathbb{Z})$ gegebenen Bilinearform eine Basis C, sodass $B_C(\beta) \in M(n \times n; \mathbb{Z})$ eine möglichst einfache Gestalt hat.

Einstellung im Fenster: „Bestimmung der Normalform“

Das Programm entscheidet per Zufall, ob es sich eine symmetrische oder schiefsymmetrische Matrix vorgibt, zu der die Normalform bestimmt werden soll und verfährt dann analog wie in den beiden vorangegangenen Aufgabenstellungen, wobei immer $A = D$ gilt.

Dieser Abschnitt dient, wie oben schon erwähnt, nur der Veranschaulichung, dass man mit dem gleichen Algorithmus auch andere Aufgaben als solche zu ähnlichen Matrizen erstellen kann. Auch hier sei erwähnt, dass die Matrizen $P \in GL(n; \mathbb{Z})$ nicht eindeutig sind und deshalb ein Algorithmus angegeben werden muss, mit dem die berechneten Matrizen P zu überprüfen sind. Dies geschieht analog zu den ähnlichen Matrizen natürlich mit der Eigenschaft $B = P \circ A \circ P^T$ und der Überprüfung, ob $\det(P) \neq 0$ gilt.

7.2 Quelltext von Agla

Im Folgenden ist der Quelltext von Agla in der Abgabeversion dargestellt. Ich bitte zu beachten, dass sich Agla wahrscheinlich weiterentwickeln wird. Der nächste Schritt wird dabei sein, die Aufgabenstellungen zu isolieren und die Klassen besser zu strukturieren als sie es jetzt sind. Man sieht dem Programm sicherlich an, dass es von einem Anfänger geschrieben worden ist, der sich den Möglichkeiten der Programmierung mit Java erst nach und nach bewusst wird.
Erfahrene Programmierer sollten vor jedweder Kritik beachten, dass mein Schwerpunkt nicht unbedingt auf einer guten Programmierung, sondern eher auf der Theorie im Hintergrund lag. Über konstruktive Vorschläge zur Verbesserung von Agla würde ich mich freuen.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Agla extends JApplet implements ActionListener, ItemListener
{
// Deklaration der Grafikflächen

// Startfenster
JPanel sf_ob, sf_un, sf_er, sf_rs, sf_aw, sf_av, sf_bt;
JButton weiter, loeschen, rueck;
JComboBox jc_aw, jc_tc, jc_ok;
JCheckBox ch_tt, ch_ma, ch_oo, ch_tc, ch_ok;
JLabel t_er, t_aw, t_au;
JTextArea ta;
JScrollPane spta;

// Startwerte der Checkboxen:
boolean tt = true;
boolean ma = false;
boolean oo = false;
boolean tc = false;
boolean ok = false;

// Deklaration des Sicherungsstrings:
String sicherung = new String("");

// Deklaration der Nebenfenster
JFrame amf, kbf, ewf, jnf, kmf, lgsf;

// Deklaration der Zustände dieser Fenster, dienen dazu, dass sie nur einmal zu öffnen sind
boolean bamf = false;
boolean blgsf = false;

// Deklaration des Fensters für ähnliche Matrizen (am)
JPanel am_ob, am_un, am_mi, am_av, am_bt;
JButton am_start;
JComboBox jc_amw;
JLabel t_amer, t_amli, t_amz, t_amgr, t_amaz, t_amrl, t_amfl;
JTextField am_z, am_gr, am_az, am_rl, am_fl;

// Deklaration des Fensters für lineare Gleichungssysteme (lgs)
JPanel lgs_ob, lgs_un, lgs_mi, lgs_av, lgs_bt;
JButton lgs_start;
JComboBox jc_lgsw;
JLabel t_lgser, t_lgsli, t_lgsz, t_lgsgr, t_lgsaz, t_lgssp, t_lgsra,t_lgsrl, t_lgsfl;
JTextField lgs_z, lgs_gr, lgs_az, lgs_sp, lgs_ra, lgs_rl, lgs_fl;

//Zeigt an, ob Parameteraufruf stattgefunden hat oder nicht:
static boolean parauf;

// Aufruf des Hauptprogramms, Start des Startfensters
public static void main(String... args)
{
// Hier ist ein Parameteraufruf integriert; sollten die Angaben nicht genügen, dann startet sich das Programm normal.
Agla agf = new Agla();
try
{
int Starttyp = (int) Integer.parseInt(args[0]);
int a = (int) Integer.parseInt(args[1]);
int g = (int) Integer.parseInt(args[2]);
int z = (int) Integer.parseInt(args[3]);
int rl = (int) Integer.parseInt(args[4]);
int fl = (int) Integer.parseInt(args[5]);
int s = (int) Integer.parseInt(args[6]);
int r = (int) Integer.parseInt(args[7]);
parauf = true;
agf.P_Start(Starttyp,a,g,z,rl,fl,s,r);
}
catch(Exception e)
{
JFrame f = new JFrame("Agla");
parauf = false;
agf.init();
agf.start();
f.add(agf);
f.setSize(800,770);
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{
System.exit(0);
```

```
}
}
);
}
}

void P_Start(int Starttyp, int a, int g, int z, int rl, int fl, int s, int r)
{
tt = false;
ok = true;
A_Typen S = new A_Typen();
String AGS[] = new String[2];
AGS = S.AS(Starttyp,z,s,r);
if (Starttyp == 11) S.Schema1(a,g,z,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 12) S.Schema1(a,g,z,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 13) S.Schema1(a,g,z,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 21) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 22) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 23) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 24) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 25) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 26) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 27) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 28) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 29) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 210) S.Schema2(a,g,z,rl,fl,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 31) S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 32) S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 33) S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 34) S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 41) S.Schema4(a,g,z,Starttyp,AGS[0],AGS[1]);
if (Starttyp == 42) S.Schema4(a,g,z,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 43) S.Schema4(a,g,z,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 51) S.Schema5(a,g,z,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 61) S.Schema6(a,g,z,s,r,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 62) S.Schema6(a,g,z,s,r,Starttyp,AGS[0],AGS[1]);
   if (Starttyp == 71) S.Schema7(a,g,z,s,r,rl,fl,Starttyp,AGS[0],AGS[1]);
  if (Starttyp == 72) S.Schema7(a,g,z,s,r,rl,fl,Starttyp,AGS[0],AGS[1]);
System.out.println("ENDE");
}

// Initialisierung des Startfensters

public void init()
{
tt = true;
ok = false;

// Containerinitialisierung
Container sf_c = new Container();
sf_c = this.getContentPane();

// Aufbau des Startfensters
sf_c.setLayout(new BorderLayout());

// oberer Bereich
sf_ob = new JPanel();
sf_ob.setLayout(new GridLayout(1,2));
sf_c.add(sf_ob,BorderLayout.NORTH);

// unterer Bereich
sf_un = new JPanel();
ta = ta = new JTextArea(33,70);
spta = new JScrollPane(ta);
sf_un.add(spta);
sf_c.add(sf_un,BorderLayout.CENTER);

// Aufteilung des oberen Bereiches
// Links: Erklärung
sf_er = new JPanel();
t_er = new JLabel();
t_er.setText("Ähnliche Matrizen");
sf_er.add(t_er);
sf_ob.add(sf_er);

// Rechts wird weiter aufgeteilt:
sf_rs = new JPanel();
sf_rs.setLayout(new BorderLayout());
sf_ob.add(sf_rs);

// Oben: Wahl der Aufgabentypen
sf_aw = new JPanel();
jc_aw = new JComboBox();
```

```
jc_aw.addItem("Ähnliche Matrizen");
jc_aw.addItem("Lineare Gleichungssysteme");
jc_aw.addItemListener(this);
sf_aw.add(jc_aw);
sf_rs.add(sf_aw, BorderLayout.NORTH);

// Mitte: Ausgabevarianten
sf_av = new JPanel();
sf_av.setLayout(new GridLayout(4,2));

ch_ma = new JCheckBox("Ausgabe für Maple");
ch_ma.addActionListener(this);
sf_av.add(ch_ma);

ch_oo = new JCheckBox("Ausgabe für OpenOffice");
ch_oo.addActionListener(this);
sf_av.add(ch_oo);

ch_tc = new JCheckBox("Ausgabe für Tex-Dateien mit");
ch_tc.addActionListener(this);
jc_tc = new JComboBox(); // Auswahl große und kleine Matrizen
jc_tc.addItem("großen Matrizen");
jc_tc.addItem("kleinen Matrizen");
sf_av.add(ch_tc);
sf_av.add(jc_tc);

ch_ok = new JCheckBox("Ausgabe für Okuson mit");
ch_ok.addActionListener(this);
jc_ok = new JComboBox(); // Auswahl große und kleine Matrizen
jc_ok.addItem("großen Matrizen");
jc_ok.addItem("kleinen Matrizen");
sf_av.add(ch_ok);
sf_av.add(jc_ok);

ch_tt = new JCheckBox("Normale Textausgabe", true); // Die Ausgabe der normalen Darstellung ist immer gesetzt!
ch_tt.addActionListener(this);
sf_av.add(ch_tt);
sf_rs.add(sf_av, BorderLayout.CENTER);

// Unten: Buttons
sf_bt = new JPanel();
sf_bt.setLayout(new GridLayout(1,3));

loeschen = new JButton("<html> Textfeldinhalt löschen </html>");
loeschen.addActionListener(this);
sf_bt.add(loeschen);

rueck = new JButton("<html> Textfeldinhalt wieder herstellen </html>");
rueck.addActionListener(this);
sf_bt.add(rueck);

weiter = new JButton("<html> Startfenster anzeigen </html>");
weiter.addActionListener(this);
sf_bt.add(weiter);
sf_rs.add(sf_bt, BorderLayout.SOUTH);
}

// Schreibt eine kurze Erklärung dessen, was die einzelnen Auswahlen können in ein Label.
public void itemStateChanged(ItemEvent ie)
{
if (ie.getItem().equals("Ähnliche Matrizen"))
{
t_er.setText("Ähnliche Matrizen");
}
if (ie.getItem().equals("Lineare Gleichungssysteme"))
{
t_er.setText("Lineare Gleichungssysteme");
}
}

// Leitet zum Nebenfenster über, verwaltet die Textfeldaktionen und verwaltet die Checkboxen.
public void actionPerformed(ActionEvent ae)
{
// Verwaltung der Checkboxen
if (ae.getSource()==ch_tt) tt = !tt;
if (ae.getSource()==ch_oo) oo = !oo;
if (ae.getSource()==ch_ma) ma = !ma;
if (ae.getSource()==ch_tc) tc = !tc;
if (ae.getSource()==ch_ok) ok = !ok;

// Überleitung zu den Nebenfenstern, die nur einmal geöffnet werden können, ansonsten werden sie einfach wieder >>
>> sichtbar gemacht
if (ae.getSource()==weiter)
```

```
{
sicherung = ta.getText();
if (jc_aw.getSelectedItem().equals("Ähnliche Matrizen"))
{
if (bamf) amf.toFront();
else
{
amf = new JFrame("Ähnliche Matrizen");
Fenster_am am = new Fenster_am();
am.init();
am.start();
amf.add(am);
amf.setSize(500,250);
amf.setVisible(true);
amf.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{
bamf = false;
}
}
);
bamf = true;
}
   }
if (jc_aw.getSelectedItem().equals("Lineare Gleichungssysteme"))
{
if (blgsf) lgsf.toFront();
else
{
lgsf = new JFrame("Lineare Gleichungssysteme");
Fenster_lgs lgs = new Fenster_lgs();
lgs.init();
lgs.start();
lgsf.add(lgs);
lgsf.setSize(500,250);
lgsf.setVisible(true);
lgsf.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{
blgsf = false;
}
}
);
blgsf = true;
}
}
}

// Textfeldaktionen
if (ae.getSource()==loeschen)
{
sicherung = ta.getText();
ta.setText("");
}

if (ae.getSource()==rueck)
{
String h_sicherung = new String();
h_sicherung = ta.getText();
ta.setText(sicherung);
sicherung = h_sicherung;
}
}


// Beginn der Deklaration der einzelnen Fenster für die Aufgabentypen
// Ähnliche Matrizen
class Fenster_am extends JApplet implements ActionListener, ItemListener
{
public void init()
{
// Containerinitialisierung
Container am_c = new Container();
am_c = this.getContentPane();

// Aufbau des am-Fensters
am_c.setLayout(new BorderLayout());

// oberer Bereich
am_ob = new JPanel();
t_amli = new JLabel("Optionen für diese Aufgaben:");
```

```
am_ob.add(t_amli);
jc_amw = new JComboBox();
jc_amw.addItem("keine Optionen");
jc_amw.addItem("Ausschluss über char. Polynom");
jc_amw.addItem("Ausschluss über Eigenwerte");
jc_amw.addItem("Ausschluss über Jordanform");
jc_amw.addItem("Ausschluss über Minimalpolynom");
jc_amw.addItem("Ausschluss über Nilpotenzgrad");
jc_amw.addItem("char. Polynom bestimmen");
jc_amw.addItem("Diagonalisierbarkeit");
jc_amw.addItem("Diagonalisierung");
jc_amw.addItem("Diagonalmatrizen");
jc_amw.addItem("Eigenvektoren bestimmen");
jc_amw.addItem("Eigenwerte bestimmen");
jc_amw.addItem("falsche Eigenwerte");
jc_amw.addItem("falsche Jordanform");
jc_amw.addItem("falsches char. Polynom");
jc_amw.addItem("falsches Minimalpolynom");
jc_amw.addItem("ganzzahlige Eigenwerte");
jc_amw.addItem("Jordanform bestimmen");
jc_amw.addItem("Minimalpolynom bestimmen");
jc_amw.addItem("Nilpotenzgrad bestimmen");
jc_amw.addItem("Umkehr der Diagonalisierung");
jc_amw.addItemListener(this);
am_ob.add(jc_amw);
am_c.add(am_ob, BorderLayout.NORTH);

// Aufteilung des mittleren Bereiches: Wahl der Anzahl der Lösungsvorschläge und weitere Einstellungen
am_mi = new JPanel();
am_mi.setLayout(new BorderLayout());
am_c.add(am_mi, BorderLayout.CENTER);

am_av = new JPanel();
am_av.setLayout(new GridLayout(5,2));
t_amrl = new JLabel("Anzahl der richtigen Lösungen:");
am_av.add(t_amrl);
am_rl = new JTextField("1");
am_av.add(am_rl);
t_amfl = new JLabel("Anzahl der falschen Lösungen:");
am_av.add(t_amfl);
am_fl = new JTextField("0");
am_av.add(am_fl);
t_amz = new JLabel("Zeilenanzahl:");
am_av.add(t_amz);
am_z = new JTextField("3");
am_av.add(am_z);
t_amgr = new JLabel("Maximalbetrag in den Lösungen:");
am_av.add(t_amgr);
am_gr = new JTextField("999");
am_av.add(am_gr);
t_amaz = new JLabel("Anzahl der Aufgaben:");
am_av.add(t_amaz);
am_az = new JTextField("10");
am_av.add(am_az);
am_mi.add(am_av, BorderLayout.CENTER);

// Unten: Buttons
am_bt = new JPanel();
am_start = new JButton("Start");
am_start.addActionListener(this);
am_bt.add(am_start);
am_mi.add(am_bt, BorderLayout.SOUTH);

// unterer Bereich
am_un = new JPanel();
t_amer = new JLabel();
t_amer.setText("Erfragt den Beweis der Ähnlichkeit von 2 Matrizen.");
am_un.add(t_amer);
am_c.add(am_un, BorderLayout.SOUTH);
}

// Verändert Text im Startfenster abhängig von der Auswahl.
public void itemStateChanged(ItemEvent ie)
{
if (ie.getItem().equals("keine Optionen"))
{
am_rl.setText("1");
am_fl.setText("0");
t_amer.setText("Aufgabe: Bew. der Ähnlichkeit von A und B ohne Einschränkungen");
}
if (ie.getItem().equals("ganzzahlige Eigenwerte"))
{
am_rl.setText("1");
```

```
am_fl.setText("O");
t_amer.setText("Aufgabe: Bew. der Ähnlichkeit von A und B; A und B haben Eigenwerte aus Z");
}
if (ie.getItem().equals("Diagonalmatrizen"))
{
am_rl.setText("1");
am_fl.setText("O");
t_amer.setText("Aufgabe: Bew. der Ähnlichkeit von A und B; A und B sind diagonalisierbar");
}
if (ie.getItem().equals("falsche Eigenwerte"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Ausschluss von zu A unähnlichen Matrizen über Eigenwerte");
}
if (ie.getItem().equals("falsches char. Polynom"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Ausschluss von zu A unähnlichen Matrizen über char. Polynom");
}
if (ie.getItem().equals("falsches Minimalpolynom"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Ausschluss von zu A unähnlichen Matrizen über Minimalpolynom");
}
if (ie.getItem().equals("falsche Jordanform"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Ausschluss von zu A unähnlichen Matrizen über Jordanformen");
}
if (ie.getItem().equals("Eigenwerte bestimmen"))
{
am_rl.setText("1");
am_fl.setText("O");
t_amer.setText("Aufgabe: Angabe der Eigenwerte von B");
}
if (ie.getItem().equals("char. Polynom bestimmen"))
{
am_rl.setText("1");
am_fl.setText("O");
t_amer.setText("Aufgabe: Angabe des charakteristischen Polynoms von B");
}
if (ie.getItem().equals("Eigenvektoren bestimmen"))
{
am_rl.setText("1");
am_fl.setText("O");
t_amer.setText("Aufgabe: Angabe der Eigenvektoren von B");
}
if (ie.getItem().equals("Ausschluss über Eigenwerte"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Angabe der Matrizen B mit den gegebenen Eigenwerten");
}
if (ie.getItem().equals("Ausschluss über char. Polynom"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Angabe der Matrizen B mit dem gegebenen char. Polynom");
}
if (ie.getItem().equals("Diagonalisierung"))
{
am_rl.setText("1");
am_fl.setText("O");
t_amer.setText("Aufgabe: Bew. der Diagonalisierbarkeit von B");
}
if (ie.getItem().equals("Diagonalisierbarkeit"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Angabe der diagonalisierbaren Matrizen B");
}
if (ie.getItem().equals("Umkehr der Diagonalisierung"))
{
am_rl.setText("1");
am_fl.setText("O");
t_amer.setText("Aufgabe: Finden einer diagonalisierbaren Matrix zu gegebenen EW und EV");
}
if (ie.getItem().equals("Jordanform bestimmen"))
{
```

```
am_rl.setText("1");
am_fl.setText("0");
t_amer.setText("Aufgabe: Bestimmung der Jordanschen Normalform zu der Matrix B");
}
if (ie.getItem().equals("Ausschluss über Jordanform"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Angabe der Matrizen mit der gegebenen Jordanschen Normalform A");
}
if (ie.getItem().equals("Minimalpolynom bestimmen"))
{
am_rl.setText("1");
am_fl.setText("0");
t_amer.setText("Aufgabe: Angabe des Minimalpolynoms der Matrix B");
}
if (ie.getItem().equals("Ausschluss über Minimalpolynom"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Angabe der Matrizen mit dem gegebenen Minimalpolynom");
}
if (ie.getItem().equals("Nilpotenzgrad bestimmen"))
{
am_rl.setText("1");
am_fl.setText("0");
t_amer.setText("Aufgabe: Angabe des Nilpotenzgrades der Matrix B");
}
if (ie.getItem().equals("Ausschluss über Nilpotenzgrad"))
{
am_rl.setText("1");
am_fl.setText("1");
t_amer.setText("Aufgabe: Angabe der Matrizen mit dem gegebenen Nilpotenzgrad");
}
}

// Hier werden die Werte eigenlesen und die Aktionen gestartet.
public void actionPerformed(ActionEvent ae)
{
Ausgabe aus = new Ausgabe();

// Initialisierung der Variablen mit Standardwerten zur Sicherung bei Fehlern:
int z = 3;
int gr = 999;
int az = 10;
int rl = 1;
int fl = 0;

// Einlesen der Werte aus den Textfeldern und Fehlermeldungen:
try
{
if (am_z.getText().charAt(0)!='-')
{z = Integer.parseInt(am_z.getText());
try
{
if (am_gr.getText().charAt(0)!='-')
{gr = Integer.parseInt(am_gr.getText());
try
{
if (am_az.getText().charAt(0)!='-')
{az = Integer.parseInt(am_az.getText());
try
{
if (am_rl.getText().charAt(0)!='-')
{rl = Integer.parseInt(am_rl.getText());
try
{
if (am_fl.getText().charAt(0)!='-')
{
fl = Integer.parseInt(am_fl.getText());
if (z>=2)
{
A_Typen S = new A_Typen();
/* Hier kommt der Start ==> */ S.AM(az,gr,z,rl,fl);
}
else aus.AusMed("Die Zeilenanzahl darf nicht kleiner als 2 gewählt werden.\n\n");
}
else aus.AusMed("Bitte die Angabe der falschen Lösungen korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der falschen Lösungen korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe der richtigen Lösungen korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der richtigen Lösungen korrigieren!\n\n");};
```

```
}else aus.AusMed("Bitte die Angabe der Aufgabenanzahl korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der Aufgabenanzahl korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe des Maximalbetrages korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe des Maximalbetrages korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe der Zeilenanzahl korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der Zeilenanzahl korrigieren!\n\n");};
}
}

class Fenster_lgs extends JApplet implements ActionListener, ItemListener
{
public void init()
{
// Containerinitialisierung
Container lgs_c = new Container();
lgs_c = this.getContentPane();

// Aufbau des lgs-Fensters
lgs_c.setLayout(new BorderLayout());

// oberer Bereich
lgs_ob = new JPanel();
t_lgsli = new JLabel("Optionen für diese Aufgaben:");
lgs_ob.add(t_lgsli);
jc_lgsw = new JComboBox();
jc_lgsw.addItem("Homogene LGS");
jc_lgsw.addItem("Inhomogene LGS");
jc_lgsw.addItem("Kern-Basis");
jc_lgsw.addItem("Rang");
jc_lgsw.addItemListener(this);
lgs_ob.add(jc_lgsw);
lgs_c.add(lgs_ob, BorderLayout.NORTH);

// Aufteilung des mittleren Bereiches: Wahl der Anzahl der Lösungsvorschläge und weitere Einstellungen
lgs_mi = new JPanel();
lgs_mi.setLayout(new BorderLayout());
lgs_c.add(lgs_mi, BorderLayout.CENTER);

lgs_av = new JPanel();
lgs_av.setLayout(new GridLayout(7,2));
t_lgsz = new JLabel("Zeilenanzahl:");
lgs_av.add(t_lgsz);
lgs_z = new JTextField("3");
lgs_av.add(lgs_z);
t_lgssp = new JLabel("Spaltenanzahl:");
lgs_av.add(t_lgssp);
lgs_sp = new JTextField("3");
lgs_av.add(lgs_sp);
t_lgsra = new JLabel("Rang der Matrix:");
lgs_av.add(t_lgsra);
lgs_ra = new JTextField("3");
lgs_av.add(lgs_ra);
t_lgsgr = new JLabel("Maximalbetrag in den Lösungen:");
lgs_av.add(t_lgsgr);
lgs_gr = new JTextField("999");
lgs_av.add(lgs_gr);
t_lgsaz = new JLabel("Anzahl der Aufgaben:");
lgs_av.add(t_lgsaz);
lgs_az = new JTextField("10");
lgs_av.add(lgs_az);
t_lgsrl = new JLabel("Anzahl der richtigen Lösungen:");
lgs_av.add(t_lgsrl);
lgs_rl = new JTextField("1");
lgs_av.add(lgs_rl);
t_lgsfl = new JLabel("Anzahl der falschen Lösungen:");
lgs_av.add(t_lgsfl);
lgs_fl = new JTextField("0");
lgs_av.add(lgs_fl);
lgs_mi.add(lgs_av, BorderLayout.CENTER);

// Unten: Buttons
lgs_bt = new JPanel();
lgs_start = new JButton("Start");
lgs_start.addActionListener(this);
lgs_bt.add(lgs_start);
lgs_mi.add(lgs_bt, BorderLayout.SOUTH);

// unterer Bereich
lgs_un = new JPanel();
t_lgser = new JLabel();
```

```
t_lgser.setText("Aufgabe: Lösen eines homogenen linearen Gleichungssystems");
lgs_un.add(t_lgser);
lgs_c.add(lgs_un, BorderLayout.SOUTH);
}

public void itemStateChanged(ItemEvent ie)
{
if (ie.getItem().equals("Homogene LGS"))
{
lgs_rl.setText("1");
lgs_fl.setText("0");
t_lgser.setText("Aufgabe: Lösen eines homogenen linearen Gleichungssystems");
}
if (ie.getItem().equals("Inhomogene LGS"))
{
lgs_rl.setText("1");
lgs_fl.setText("0");
t_lgser.setText("Aufgabe: Lösen eines inhomogenen linearen Gleichungssystems");
}
if (ie.getItem().equals("Kern-Basis"))
{
lgs_rl.setText("1");
lgs_fl.setText("1");
t_lgser.setText("Aufgabe: Angabe der Matrizen mit den gegebenen Vektoren als Basis für den Kern");
}
if (ie.getItem().equals("Rang"))
{
lgs_rl.setText("1");
lgs_fl.setText("1");
t_lgser.setText("Aufgabe: Angabe der Matrizen mit dem gegebenen Rang");
}
}

public void actionPerformed(ActionEvent ae)
{
Ausgabe aus = new Ausgabe();

// Initialisierung der Variablen mit Standardwerten zur Sicherung bei Fehlern:
int z = 3;
int gr = 999;
int az = 10;
int sp = 1;
int ra = 0;
int rl = 1;
int fl = 0;

// Einlesen der Werte aus den Textfeldern und Fehlermeldungen:
try
{
if (lgs_z.getText().charAt(0)!='-')
{z = Integer.parseInt(lgs_z.getText());
try
{
if (lgs_gr.getText().charAt(0)!='-')
{gr = Integer.parseInt(lgs_gr.getText());
try
{
if (lgs_az.getText().charAt(0)!='-')
{az = Integer.parseInt(lgs_az.getText());
try
{
if (lgs_sp.getText().charAt(0)!='-')
{sp = Integer.parseInt(lgs_sp.getText());
try
{
try
{
if (lgs_rl.getText().charAt(0)!='-')
{rl = Integer.parseInt(lgs_rl.getText());
try
{
if (lgs_fl.getText().charAt(0)!='-')
{
fl = Integer.parseInt(lgs_fl.getText());
if (lgs_ra.getText().charAt(0)!='-')
{
ra = Integer.parseInt(lgs_ra.getText());
if (ra<=Math.min(z,sp))
{
if (z>=2)
{
if (sp>=2)
{
```

```
A_Typen S = new A_Typen();
/* Hier kommt der Start ==> */ S.LGS(az,gr,z,sp,ra,rl,fl);
}
else aus.AusMed("Die Spaltenanzahl darf nicht kleiner als 2 gewählt werden.\n\n");
}
else aus.AusMed("Die Zeilenanzahl darf nicht kleiner als 2 gewählt werden.\n\n");
}
   else aus.AusMed("Der Rang kann höchstens so groß sein wie das Minimum von Zeilen- und Spaltenanzahl.\n\n");
}
else aus.AusMed("Bitte die Angabe des Rangs korrigieren!\n\n");
}
else aus.AusMed("Bitte die Angabe der falschen Lösungen korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der falschen Lösungen korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe der richtigen Lösungen korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der richtigen Lösungen korrigieren!\n\n");};
}
catch(Exception e){aus.AusMed("Bitte die Angabe des Rangs korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe der Spaltenanzahl korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der Spaltenanzahl korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe der Aufgabenanzahl korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der Aufgabenanzahl korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe des Maximalbetrages korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe des Maximalbetrages korrigieren!\n\n");};
}else aus.AusMed("Bitte die Angabe der Zeilenanzahl korrigieren!\n\n");
}
catch(Exception e){aus.AusMed("Bitte die Angabe der Zeilenanzahl korrigieren!\n\n");};
}
}

// Das eigentliche Programm beginnt hier!

class Ausgabe
{
//Legt das Ausgabemedium fest.
void AusMed(String text)
  {
   if (parauf) System.out.print(text);
   else ta.append(text);
  }

// Legt die Ausgabeformatierung fest.
void Ausdruck(int M[][], int x, int y)
    {
Ausgabe aus = new Ausgabe();
// einl sagt dem User, welche Matrix er vor sich hat, und ist abhängig von x.
String einl = new String();
switch(x)
{
case 1:
{
einl = "A := ";
break;
}
case 2:
{
einl = "B := ";
break;
}
case 3:
{
aus.AusMed("Beispiellösung: ");
einl = "P := ";
break;
}
case 4:
{
einl = "B_f := ";
break;
}
}
// Hier kommt die Deklaration für alle die verschiedenen Ausgabevarianten.
// Und gleich dahinter steht der Aufruf für die Ausgabe.
if (y==1) //normaler Text
{
String at_mb = new String("([ ");
String at_sw = new String(", ");
String at_zw = new String("],[ ");
String at_me = new String("]) ");
```

```
String at_ende = new String("");
aus.AusMed(einl);
aus.Ausdruck_komplex(M,at_mb,at_sw,at_zw,at_me,at_ende);
    }
if (y==2) // Maple - gut zum Testen und zur Fehlersuche
{
String at_mb = new String("matrix([[ ");
String at_sw = new String(", ");
String at_zw = new String("],[ ");
String at_me = new String("]]) ");
String at_ende = new String("; ");
aus.AusMed(einl);
aus.Ausdruck_komplex(M,at_mb,at_sw,at_zw,at_me,at_ende);
}
if (y==3) // OpenOffice - nicht so aufwendig wie Tex
{
String at_mb = new String("left( matrix{ ");
String at_sw = new String("# ");
String at_zw = new String("## ");
String at_me = new String("} right) ");
String at_ende = new String("newline ");
aus.AusMed(einl);
aus.Ausdruck_komplex(M,at_mb,at_sw,at_zw,at_me,at_ende);
}
if (y==4) // Tex
{
String at_mb = new String("\\begin{pmatrix} ");
if (jc_tc.getSelectedItem().equals("kleinen Matrizen")) at_mb = "\\left( \\begin{smallmatrix} ";
String at_sw = new String("& ");
String at_zw = new String("\\\\ ");
String at_me = new String("\\end{pmatrix} ");
if (jc_tc.getSelectedItem().equals("kleinen Matrizen")) at_me = "\\end{smallmatrix} \\right)";
String at_ende = new String(" $ ");
aus.AusMed(" $ "+einl+" $ ");
aus.AusMed(" $ ");
aus.Ausdruck_komplex(M,at_mb,at_sw,at_zw,at_me,at_ende);
}
if (y==5) // Okuson
{
String at_mb = new String("\\begin{pmatrix} ");
try {if (jc_ok.getSelectedItem().equals("kleinen Matrizen")) at_mb = "\\left( \\begin{smallmatrix} ";} >>
>> catch(Exception e){}
String at_sw = new String("&amp; ");
String at_zw = new String("\\\\ ");
String at_me = new String("\\end{pmatrix} ");
try {if (jc_ok.getSelectedItem().equals("kleinen Matrizen")) at_me = "\\end{smallmatrix} \\right)";} >>
>> catch(Exception e){}
String at_ende = new String(" $ ");
aus.AusMed(" $ ");
aus.Ausdruck_komplex(M,at_mb,at_sw,at_zw,at_me,at_ende);
}
  } // Ende

void Ausdruck_LGS(int M[][], int z, int s, int r, int y)
  {
Ausgabe aus = new Ausgabe();
int h = Math.max(z,s);
if (y==1)
{
String at_mb = new String("([ ");
String at_sw = new String(", ");
String at_zw = new String("],[ ");
String at_me = new String("]) ");
String at_ci = new String("* ");
String at_gl = new String("= ");
String at_ende = new String("");
String at_kl = new String("( ");
String at_lk = new String(") ");
String at_pl = new String("+ ");
String at_mu = new String("* ");
String at_anf = new String("");
aus.Ausdruck_komplex_LGS(M,z,s,r,at_mb,at_sw,at_zw,at_me,at_ci,at_gl,at_ende,at_kl,at_lk,at_pl,at_mu,at_anf);
}
if (y==2)
{
String at_mb = new String("matrix([[ ");
String at_sw = new String(", ");
String at_zw = new String("],[ ");
String at_me = new String("]]) ");
String at_ci = new String("&* ");
String at_gl = new String("= ");
String at_ende = new String("; evalm(%); ");
String at_kl = new String("( ");
```

```
String at_lk = new String(") ");
String at_pl = new String("&+ ");
String at_mu = new String("&* ");
String at_anf = new String("");
aus.Ausdruck_komplex_LGS(M,z,s,r,at_mb,at_sw,at_zw,at_me,at_ci,at_gl,at_ende,at_kl,at_lk,at_pl,at_mu,at_anf);
}
if (y==3)
{
String at_mb = new String("left( matrix{ ");
String at_sw = new String("# ");
String at_zw = new String("## ");
String at_me = new String("} right) ");
String at_ci = new String("circ ");
String at_gl = new String("= ");
String at_ende = new String("newline ");
String at_kl = new String("left( ");
String at_lk = new String("right) ");
String at_pl = new String("+ ");
String at_mu = new String("cdot ");
String at_anf = new String("");
aus.Ausdruck_komplex_LGS(M,z,s,r,at_mb,at_sw,at_zw,at_me,at_ci,at_gl,at_ende,at_kl,at_lk,at_pl,at_mu,at_anf);
}
if (y==4)
{
String at_mb = new String("\\begin{pmatrix} ");
if (jc_tc.getSelectedItem().equals("kleinen Matrizen")) at_mb = "\\left( \\begin{smallmatrix} ";
String at_sw = new String("& ");
String at_zw = new String("\\\\ ");
String at_me = new String("\\end{pmatrix} ");
if (jc_tc.getSelectedItem().equals("kleinen Matrizen")) at_me = "\\end{smallmatrix} \\right)";
String at_ci = new String("\\circ ");
String at_gl = new String("= ");
String at_ende = new String(" $ ");
String at_kl = new String("\\left( ");
String at_lk = new String("\\right) ");
String at_pl = new String("+ ");
String at_mu = new String("\\cdot ");
String at_anf = new String(" $ ");
aus.Ausdruck_komplex_LGS(M,z,s,r,at_mb,at_sw,at_zw,at_me,at_ci,at_gl,at_ende,at_kl,at_lk,at_pl,at_mu,at_anf);
}
if (y==5)
{
String at_mb = new String("\\begin{pmatrix} ");
try {if (jc_ok.getSelectedItem().equals("kleinen Matrizen")) at_mb = "\\left( \\begin{smallmatrix} ";} >>
>> catch(Exception e) {};
String at_sw = new String("&amp; ");
String at_zw = new String("\\\\ ");
String at_me = new String("\\end{pmatrix} ");
try {if (jc_ok.getSelectedItem().equals("kleinen Matrizen")) at_me = "\\end{smallmatrix} \\right)";} >>
>> catch(Exception e) {};
String at_ci = new String("\\circ ");
String at_gl = new String("= ");
String at_ende = new String(" $ ");
String at_kl = new String("\\left( ");
String at_lk = new String("\\right) ");
String at_pl = new String("+ ");
String at_mu = new String("\\cdot ");
String at_anf = new String(" $ ");
aus.Ausdruck_komplex_LGS(M,z,s,r,at_mb,at_sw,at_zw,at_me,at_ci,at_gl,at_ende,at_kl,at_lk,at_pl,at_mu,at_anf);
}
  }

// Druckt fröhlich alle Matrizen aus die man übergibt.
void Ausdruck_komplex(int M[][], String at_mb, String at_sw, String at_zw, String at_me, String at_ende)
  {
   Ausgabe aus = new Ausgabe();
   aus.AusMed(at_mb);
   int z = M.length;
   int s = M[0].length;
for (int l=0; l<z; l++)
{
for (int k=0; k<s; k++)
{
aus.AusMed(M[l][k]+" ");
if (k<s-1) aus.AusMed(at_sw);
if ((k==s-1) & (l!=z-1)) aus.AusMed(at_zw);
}
}
aus.AusMed(at_me+at_ende+"\n\n");
  }

void Ausdruck_komplex_LGS(int M[][], int z, int s, int r, String at_mb, String at_sw, String at_zw, String at_me, >>
>> String at_ci, String at_gl, String at_ende, String at_kl, String at_lk, String at_pl, String at_mu, String at_anf)
```

```
  {
   Ausgabe aus = new Ausgabe();
   int h = Math.max(z,s);
   if (s-r==0)
   {
   at_kl = "";
   at_lk = "";
   }
   aus.AusMed(at_anf+at_mb);
for (int l=0; l<z; l++)
     {
     for (int k=0; k<s; k++)
     {
     aus.AusMed(M[l][k]+" ");
     if (k<s-1) aus.AusMed(at_sw);
     if ((k==s-1) & (l!=z-1)) aus.AusMed(at_zw);
     }
     }
aus.AusMed(at_me +"\n"+at_ci+" x \n");
aus.AusMed(at_gl+"\n"+at_mb);
for (int l=0; l<z; l++)
{
aus.AusMed(M[l][2*h]+" ");
if (l!=z-1) aus.AusMed(at_zw);
}
aus.AusMed(at_me+"\n"+at_ende+"\n\n");
aus.AusMed("Lösung:\n\n"+at_anf+" x := "+at_kl+at_mb);
for (int l=0; l<s; l++)
{
aus.AusMed(M[l][h]+" ");
if (l!=s-1) aus.AusMed(at_zw);
}
aus.AusMed(at_me+"\n");
for (int l=0; l<s-r; l++)
{
aus.AusMed(at_pl+"\n"+at_kl+"t_"+(l+1)+" "+at_mu+at_mb);
for (int k=0; k<s; k++)
{
aus.AusMed(M[k][h+1+l]+" ");
if (k!=s-1) aus.AusMed(at_zw);
}
aus.AusMed(at_me+at_lk+"\n");
}
aus.AusMed(at_lk+at_ende+"\n");
  }

   void ausEW(int M[][],int y)
  {
   Ausgabe aus = new Ausgabe();
   int z = M.length;
   if (y!=5) aus.AusMed("Die Eigenwerte sind: ");
  boolean hbf[] = new boolean[4*z+1];
for (int hl=0; hl<4*z+1; hl++) hbf[hl]=true;
for (int i=0; i<z; i++) if (hbf[M[i][i]+(2*z)]) hbf[M[i][i]+(2*z)]=false;
int zaehler=0;
for (int hl=0; hl<4*z+1; hl++) if (!hbf[hl])
{
zaehler=zaehler+1;
if (y==4) aus.AusMed("$\\lambda_"+zaehler+" = ");
else if (y!=5) aus.AusMed("l_"+zaehler+" = ");
aus.AusMed((hl-(2*z))+"");
if (y==4) aus.AusMed("$");
if (y==5) aus.AusMed(",");
else aus.AusMed("   ");
}
  }

   void ausCP(int M[][], int y)
  {
   Ausgabe aus = new Ausgabe();
   if ((y==4)|(y==5)) aus.AusMed("$");
   int z = M.length;
   aus.AusMed("p_B(x)= ");
   if (z % 2 ==1) aus.AusMed("-");
  boolean hbf[] = new boolean[4*z+1];
for (int hl=0; hl<4*z+1; hl++) hbf[hl]=true;
int hif[] = new int[4*z+1];
for (int hl=0; hl<4*z+1; hl++) hif[hl]=0;
  for (int i=0; i<z; i++)
  {
  if (hbf[M[i][i]+(2*z)]) hbf[M[i][i]+(2*z)]=false;
  hif[M[i][i]+(2*z)]=hif[M[i][i]+(2*z)]+1;
  }
```

```
  for (int hl=0; hl<4*z+1; hl++) if (!hbf[hl])
  {
  if (hl<2*z) aus.AusMed("(x+"+((2*z)-hl)+")");
   else if (hl>2*z) aus.AusMed("(x-"+(hl-(2*z))+")");
   else aus.AusMed("x");
   aus.AusMed("^"+hif[hl]+" ");
   }
   if ((y==4)|(y==5)) aus.AusMed("$");
  }

   void ausMP(int M[][], int y)
  {
   Ausgabe aus = new Ausgabe();
   int z = M.length;
   if ((y==4)|(y==5)) aus.AusMed("$");
   aus.AusMed("m_B(x)= ");
  int i,j;
   int BF[][]= new int[2][z];
   for (int hl=0; hl<z; hl++)
   {
   BF[0][hl]=1;
  BF[1][hl]=0;
  }
   i = 0;
   for (int hl=0; hl<z-1; hl++)
   {
   if (M[hl][hl+1]==1) BF[0][i]=BF[0][i]+1;
   else i=i+1;
   }
i=0; j=0;
   while (j<z)
   {
   BF[1][i]=BF[0][i]-1+j;
   j=j+BF[0][i];
   i=i+1;
   }
   boolean hbf[] = new boolean[4*z+1];
for (int hl=0; hl<4*z+1; hl++) hbf[hl]=true;
i=0; j=0;
   while (j<z)
   {
   if (hbf[M[BF[1][i]][BF[1][i]]+(2*z)])
   {
   hbf[M[BF[1][i]][BF[1][i]]+(2*z)]=false;
   if (M[BF[1][i]][BF[1][i]]<0) aus.AusMed("(x+"+(-M[BF[1][i]][BF[1][i]])+")");
   else if (M[BF[1][i]][BF[1][i]]>0) aus.AusMed("(x-"+M[BF[1][i]][BF[1][i]]+")");
   else aus.AusMed("x");
   aus.AusMed("^"+BF[0][i]+" ");
   }
   j=j+BF[0][i];
   i=i+1;
   }
   if ((y==4)|(y==5)) aus.AusMed("$");
  }

// Jedes Aufgabenschema benötigt seine eigene Ausgabeform.

   void ausdruck_S1(int A[][][][], int a, int z, int Starttyp, String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
aus.AusMed("\n");
   for (int l=0; l<a; l++)
   {
   if (y==5) aus.AusMed("Die Matrizen werden in der Schreibweise für Okuson ausgegeben,\nallerdings ist die >>
   >> Eindeutigkeit der Matrix P nicht gegeben.\nDeshalb werden keine vollständigen Aufgaben generiert.\n\n");
   aus.AusMed(AGS); // Schreiben der Aufgabenstellung
   // Matrizen werden ausgegeben.
   aus.Ausdruck(A[0][l],1,y); // A
   aus.Ausdruck(A[1][l],2,y); // B
   aus.Ausdruck(A[2][l],3,y); // P
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
   aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
  }

   void ausdruck_S2(int A[][][][], int a, int z, int rl, int x, int Starttyp, String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
aus.AusMed("\n");
Matrizenrechnung MR = new Matrizenrechnung();
if (y!=5)
{
for (int l=0; l<a; l++)
```

```
   {
   aus.AusMed(AGS); // Schreiben der Aufgabenstellung
   // Matrizen werden ausgegeben.
   for (int k=0; k<x; k++)
{
if (k==0)
  switch(Starttyp)
  {
  case 21:
  {
aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
  case 22:
  {
  aus.Ausdruck(A[k][l],1,y); // A
  break;
}
  case 23:
  {
  aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
case 24:
  {
  aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
  case 25:
{
  aus.ausEW(A[0][l],y);
  aus.AusMed("\n\n");
  break;
  }
  case 26:
  {
  aus.ausCP(A[0][l],y);
  aus.AusMed("\n\n");
break;
  }
  case 27:
  {
  // Bei 27 soll nichts ausgegeben werden.
  break;
}
  case 28:
  {
  aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
  case 29:
{
  aus.ausMP(A[0][l],y);
  aus.AusMed("\n\n");
  break;
  }
  case 210:
  {
  int M[][] = new int[z][z];
M=MR.Gleich(A[k][l]);
   int i,j;
int BF[][]= new int[2][z];
   for (int hl=0; hl<z; hl++)
   {
BF[0][hl]=1;
    BF[1][hl]=0;
  }
   i = 0;
for (int hl=0; hl<z-1; hl++)
{
if (M[hl][hl+1]==1) BF[0][i]=BF[0][i]+1;
else i=i+1;
}
if ((y==4)|(y==5)) aus.AusMed("$");
aus.AusMed("k ="+BF[0][0]);
if ((y==4)|(y==5)) aus.AusMed("$");
aus.AusMed("\n\n");
  break;
  }
   }
   else if (k<=rl) aus.Ausdruck(A[k][l],2,y); // B
   else aus.Ausdruck(A[k][l],4,y); //Falsch_B
   }
```

```
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
   aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
   }
   else for (int l=0; l<a; l++)
   {
   boolean hbf[] = new boolean[x-1];
   for (int k=0;k<x-1;k++) hbf[k]=false;
   int pos[]= new int[x-1];
   for (int k=0;k<x-1;k++)
   {
   int h;
   do h=(int)(Math.random()*(x-1)); while (hbf[h]);
   hbf[h]=true;
   pos[k]=h+1;
   } //Lösungen und falsche Lösungen müssen durcheinander geworfen werden, das ist hier gerade geschehen.
   // Und jetzt wird die Okuson-Aufgabe generiert:
   aus.AusMed("<QUESTION>\n<ANSWERS type=\"c\">");
   for (int k=0;k<x-1;k++)
   {
   aus.AusMed(""+(k+1));
   if (k<x-2) aus.AusMed(" | ");
   }
   aus.AusMed("</ANSWERS>\n<VARIANT solution=\"");
   int rlz = 0;
   for (int k=0;k<x-1;k++)
   {
   if (pos[k]<rl+1)
   {
   aus.AusMed(""+(k+1));
   rlz = rlz + 1;
   if (rlz<rl) aus.AusMed(" | ");
   }
   }
   aus.AusMed("\">\n"+AGS);
   for (int k=0; k<x; k++)
{
if (k==0)
  switch(Starttyp)
  {
  case 21:
  {
aus.AusMed("$A=$");
aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
  case 22:
  {
  aus.AusMed("$A=$");
  aus.Ausdruck(A[k][l],1,y); // A
  break;
}
  case 23:
  {
  aus.AusMed("$A=$");
  aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
case 24:
  {
  aus.AusMed("$A=$");
  aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
  case 25:
{
  aus.ausEW(A[0][l],4); //die Tex-formatierung, damit es auch schön aussieht
  aus.AusMed("\n\n");
  break;
  }
  case 26:
  {
  aus.ausCP(A[0][l],y);
  aus.AusMed("\n\n");
break;
  }
  case 27:
  {
  // Bei 27 soll nichts ausgegeben werden.
  break;
}
  case 28:
  {
```

```
  aus.AusMed("$A=$");
  aus.Ausdruck(A[k][l],1,y); // A
  break;
  }
  case 29:
{
  aus.ausMP(A[0][l],y);
  aus.AusMed("\n\n");
  break;
  }
  case 210:
  {
  int M[][] = new int[z][z];
M=MR.Gleich(A[k][l]);
   int i,j;
int BF[][]= new int[2][z];
   for (int hl=0; hl<z; hl++)
   {
BF[0][hl]=1;
    BF[1][hl]=0;
  }
   i = 0;
for (int hl=0; hl<z-1; hl++)
{
if (M[hl][hl+1]==1) BF[0][i]=BF[0][i]+1;
else i=i+1;
}
if ((y==4)|(y==5)) aus.AusMed("$");
aus.AusMed("k ="+BF[0][0]);
if ((y==4)|(y==5)) aus.AusMed("$");
aus.AusMed("\n\n");
  break;
  }
   }
   else
   {
   aus.AusMed("$B_"+k+"=$");
   aus.Ausdruck(A[pos[k-1]][l],2,y); // B
   }
   }
   aus.AusMed("</VARIANT>\n</QUESTION>\n\n\n");
   }
  }

   void ausdruck_S3(int A[][][][], int a, int z, int Starttyp, String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
Matrizenrechnung MR = new Matrizenrechnung();
Generierung G = new Generierung();
aus.AusMed("\n");
   if (y!=5)
   {
   for (int l=0; l<a; l++)
   {
   aus.AusMed(AGS); // Schreiben der Aufgabenstellung
   // Matrizen bzw. Werte daraus werden ausgegeben.
   aus.Ausdruck(A[1][l],2,y); // B
   switch(Starttyp)
  {
  case 31:
  {
  aus.ausEW(A[0][l],y);
  aus.AusMed("\n\n");
  break;
  }
  case 32:
  {
  aus.ausCP(A[0][l],y);
  aus.AusMed("\n\n");
  break;
  }
  case 33:
  {
  aus.ausMP(A[0][l],y);
  aus.AusMed("\n\n");
  break;
  }
  case 34:
  {
  int M[][] = new int[z][z];
M=MR.Gleich(A[1][l]);
   int i,j;
   int BF[][]= new int[2][z];
```

```
for (int hl=0; hl<z; hl++)
{
   BF[0][hl]=1;
  BF[1][hl]=0;
    }
i = 0;
   for (int hl=0; hl<z-1; hl++)
   {
   if (M[hl][hl+1]==1) BF[0][i]=BF[0][i]+1;
   else i=i+1;
   }
   if ((y==4)|(y==5)) aus.AusMed("$");
   aus.AusMed("k ="+BF[0][0]);
   if ((y==4)|(y==5)) aus.AusMed("$");
   aus.AusMed("\n\n");
  break;
  }
  }
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
   aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
   }
   else for (int l=0; l<a; l++)
   {
   // Das Problem an Okuson ist die teilweise komplizierte Angabe der Polynome und anderes
   // Um das Problem zu umgehen, wird hier eine MC-Aufgabe erzeugt wie schon oben.
   // Zuerst muss hier jetzt ein Feld von 2 Lösungen (richtig + falsch) erzeugt werden.
   int LFeld[][][] = new int[2][z][z];
   LFeld[0]=MR.Gleich(A[0][l]);
   switch(Starttyp)
  {
  case 31:
  {
  LFeld[1]= MR.erstelle_M_mit_fEW(A[0][l]);
  break;
  }
  case 32:
  {
  LFeld[1]= MR.erstelle_M_mit_fcP(A[0][l]);
  break;
  }
  case 33:
  {
  LFeld[1]= MR.erstelle_M_mit_fmP(A[0][l]);
  break;
  }
  case 34:
  {
  LFeld[1] = MR.erstelle_M_mit_fmP(A[0][l]);
  int za = 0;
   for (int hl=0; hl<z; hl++)
   for (int hk=0; hk<z; hk++)
   za=za+LFeld[1][hl][hk];
   if (za==0) LFeld[1][0][0]=1;
  break;
  }
  }
   //An dieser Stelle ist man damit fertig; jetzt müssen die Lösungen permutiert werden.
   boolean hbf[] = new boolean[2];
   for (int k=0;k<2;k++) hbf[k]=false;
   int pos[]= new int[2];
   for (int k=0;k<2;k++)
   {
   int h;
   do h=(int)(Math.random()*2); while (hbf[h]);
   hbf[h]=true;
   pos[k]=h;
   } // Ende der Permutation; jetzt geht es wie oben weiter.
   aus.AusMed("<QUESTION>\n<ANSWERS type=\"c\">");
   aus.AusMed("1|2");
   aus.AusMed("</ANSWERS>\n<VARIANT solution=\"");
   int rlz = 0;
   if (pos[0]<1) aus.AusMed(""+(1));
   else aus.AusMed(""+(2));
   aus.AusMed("\">\n"+AGS+"$B=$");
   aus.Ausdruck(A[1][l],2,y); // B
   for (int k=0; k<2; k++)
{
aus.AusMed((k+1)+" : ");
switch(Starttyp)
  {
  case 31:
  {
```

```
  aus.ausEW(LFeld[pos[k]],y);
  break;
  }
  case 32:
  {
  aus.ausCP(LFeld[pos[k]],y);
  break;
  }
  case 33:
  {
  aus.ausMP(LFeld[pos[k]],y);
  break;
  }
  case 34:
  {
  int M[][] = new int[z][z];
M=MR.Gleich(LFeld[pos[k]]);
   if (M[0][0]==0)
   {
   int i,j;
   int BF[][]= new int[2][z];
for (int hl=0; hl<z; hl++)
{
   BF[0][hl]=1;
  BF[1][hl]=0;
    }
i = 0;
   for (int hl=0; hl<z-1; hl++)
   {
   if (M[hl][hl+1]==1) BF[0][i]=BF[0][i]+1;
   else i=i+1;
   }
   if ((y==4)|(y==5)) aus.AusMed("$");
   aus.AusMed("k ="+BF[0][0]);
   }
   else aus.AusMed("k = - ");
   if ((y==4)|(y==5)) aus.AusMed("$");
  break;
  }
  }
  aus.AusMed("\n\n");
   }
   aus.AusMed("</VARIANT>\n</QUESTION>\n\n\n");
   }
  }

   void ausdruck_S4(int A[][][][], int a, int z, int Starttyp, String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
aus.AusMed("\n");
   for (int l=0; l<a; l++)
   {
   if (y==5) aus.AusMed("Die Matrizen werden in der Schreibweise für Okuson ausgegeben,\nallerdings ist die >>
   >> Eindeutigkeit der Matrix P nicht gegeben.\nDeshalb werden keine vollständigen Aufgaben generiert.\n\n");
aus.AusMed(AGS); // Schreiben der Aufgabenstellung
   // Matrizen werden ausgegeben.
   aus.Ausdruck(A[1][l],2,y); // B
   aus.Ausdruck(A[0][l],1,y); // A
   aus.Ausdruck(A[2][l],3,y); // P
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
   aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
  }

   void ausdruck_S5(int A[][][][], int a, int z, int Starttyp, String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
aus.AusMed("\n");
   for (int l=0; l<a; l++)
   {
   if (y==5) aus.AusMed("Die Matrizen werden in der Schreibweise für Okuson ausgegeben,\nallerdings ist die >>
   >> Eindeutigkeit der Matrix P nicht gegeben.\nDeshalb werden keine vollständigen Aufgaben generiert.\n\n");
aus.AusMed(AGS); // Schreiben der Aufgabenstellung
   // Matrizen werden ausgegeben.
   aus.Ausdruck(A[0][l],1,y); // A
   if (Starttyp==51) aus.AusMed("Die Diagonaleinträge sind die Eigenwerte.\n\n");
   aus.Ausdruck(A[2][l],3,y); // P
   if (Starttyp==51) aus.AusMed("Die Spalten von P sind die Eigenvekoren. (Reihenfolge beachten)\n\n");
   aus.Ausdruck(A[1][l],2,y); // B
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
   aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
  }
```

```
   void ausdruck_S6(int A[][][], int z, int a, int s, int r, int Starttyp, String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
aus.AusMed("\n");
   for (int l=0; l<a; l++)
   {
   if (y==5) aus.AusMed("Die Aufgabe wird in der Schreibweise für Okuson ausgegeben,\nallerdings ist die >>
   >> Eindeutigkeit der Lösung nicht in jedem Fall gegeben.\nDeshalb werden keine vollständigen Aufgaben >>
   >> generiert.\n\n");
   aus.AusMed(AGS);
   aus.Ausdruck_LGS(A[l],z,s,r,y);
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
    aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
  }

   void ausdruck_S7(int A[][][][], int z, int a, int s, int r, int rl, int x, int Kern[][][], int Starttyp, >>
   >> String AGS, int y)
  {
Ausgabe aus = new Ausgabe();
aus.AusMed("\n");
if (y!=5)
{
   for (int l=0; l<a; l++)
   {
   aus.AusMed(AGS); // Schreiben der Aufgabenstellung
   // Matrizen werden ausgegeben.
   for (int k=0; k<x; k++)
   {
   if (k==0)
   {
   if (Starttyp==71)
   {
   int kg = s-r;
   if (kg==0) kg = 1;
   int Kernbasis[][] =new int [s][kg];
   for (int i=0; i<s; i++)
   for (int j=0; j<kg; j++)
   if (s-r!=0) Kernbasis[i][j]=Kern[l][i][r+j];
   else Kernbasis[i][j]=0;
   aus.Ausdruck(Kernbasis,1,y);
   }
   }
   else if (k<=rl) aus.Ausdruck(A[k][l],2,y); // B
   else aus.Ausdruck(A[k][l],4,y); //Falsch_B
   }
   aus.AusMed("*** Ende der Aufgabe *** \n\n");
   }
   aus.AusMed("*** *** *** Ende des Aufgabentyps *** *** *** \n\n");
   }
   else for (int l=0; l<a; l++)
   {
   boolean hbf[] = new boolean[x-1];
   for (int k=0;k<x-1;k++) hbf[k]=false;
   int pos[]= new int[x-1];
   for (int k=0;k<x-1;k++)
   {
   int h;
   do h=(int)(Math.random()*(x-1)); while (hbf[h]);
   hbf[h]=true;
   pos[k]=h+1;
   } //Lösungen und falsche Lösungen müssen durcheinander geworfen werden, das ist hier gerade geschehen.
   // Und jetzt wird die Okuson-Aufgabe generiert:
   aus.AusMed("<QUESTION>\n<ANSWERS type=\"c\">");
   for (int k=0;k<x-1;k++)
   {
   aus.AusMed(""+(k+1));
   if (k<x-2) aus.AusMed(" | ");
   }
   aus.AusMed("</ANSWERS>\n<VARIANT solution=\"");
   int rlz = 0;
   for (int k=0;k<x-1;k++)
   {
   if (pos[k]<rl+1)
   {
   aus.AusMed(""+(k+1));
   rlz = rlz + 1;
   if (rlz<rl) aus.AusMed(" | ");
   }
   }
   aus.AusMed("\">\n"+AGS);
   for (int k=0; k<x; k++)
```

```
   {
   if (k==0)
   {
   if (Starttyp==71)
   {
   int kg = s-r;
   if (kg==0) kg = 1;
   int Kernbasis[][] =new int [s][kg];
   for (int i=0; i<s; i++)
   for (int j=0; j<kg; j++)
   if (s-r!=0) Kernbasis[i][j]=Kern[l][i][r+j];
   else Kernbasis[i][j]=0;
   aus.Ausdruck(Kernbasis,1,y);
   }
   }
   else
   {
   aus.AusMed("$B_"+k+"=$");
   aus.Ausdruck(A[pos[k-1]][l],2,y);
   }
   }
   aus.AusMed("</VARIANT>\n</QUESTION>\n\n\n");
   }
  }
}

class Algorithmen
{
int[] Euklidischer_Algorithmus( int a, int b)
  {
   int A[]= new int[3];
   int h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, d1, d2, d3, d4;
   d1 = 0; d2 = 0; d3 = 0;
   if (Math.abs(a)>Math.abs(b))
   {
   h1 = Math.abs(a);
   h2 = Math.abs(b);
   }
   else
   {
   h1 = Math.abs(b);
   h2 = Math.abs(a);
   };
   h3 = 1; h4 =0; h5 = 0; h6 =1;
   if (h1 % h2 !=0)
   while (h2!=0)
      {
      d1 = h2;
      d2 = h5;
       d3 = h6;
      h8 = h1 % h2;
       h7 = (h1 - h8) / h2;
      h9 = h3 - h7 * h5;
      h10 = h4 - h7 * h6;
       h1 = h2;
       h2 = h8;
      h3 = h5;
       h5 = h9;
       h4 = h6;
       h6 = h10;
      }
   else
      {
         d1 = h2;
          d2 = 1;
          d3 = 1 - h1 / h2;
       }
   if (Math.abs(a)<Math.abs(b))
   {
       d4 = d2;
      d2 = d3;
      d3 = d4;
   }
   A[0]=d1; A[1]=d2; A[2]=d3;
   return(A);
     }
}

class Endomorphismen
{
int[][] Kombination(int z, int a, int b, int c, int d)
  {
   int M[][] = new int[z][z];
```

```
   Matrizenrechnung H = new Matrizenrechnung();
   M = H.erstelle_Einheitsmatrix(z,z);
   int A[] = new int[3];
   Algorithmen Eukl = new Algorithmen();
   A = Eukl.Euklidischer_Algorithmus(c,d);
   M[a-1][a-1] = c / A[0];
   M[a-1][b-1] = d / A[0];
   M[b-1][b-1] = c / Math.abs(c) * A[1];
   M[b-1][a-1] = - d / Math.abs(d) * A[2];
   return(M);
  }

   int[][] vielfache_Zeilenaddition(int z, int a, int b, int c)
  {
   int M[][] = new int[z][z];
   Matrizenrechnung H = new Matrizenrechnung();
   M = H.erstelle_Einheitsmatrix(z,z);
   M[a-1][b-1]=c;
   return(M);
  }

   int[][] Vorzeichenwechsel(int z, int a)
  {
   int M[][] = new int[z][z];
   Matrizenrechnung H = new Matrizenrechnung();
   M = H.erstelle_Einheitsmatrix(z,z);
   M[a-1][a-1]=-1;
   return(M);
  }

   int[][] Zeilentausch(int z, int a, int b)
  {
   int M[][] = new int[z][z];
   Matrizenrechnung H = new Matrizenrechnung();
   M = H.erstelle_Einheitsmatrix(z,z);
   M[a-1][a-1]=0;
   M[b-1][b-1]=0;
   M[a-1][b-1]=1;
   M[b-1][a-1]=1;
   return(M);
  }

int[][] Invertierung(int M[][], int a, int b)
  {
   int z = M.length;
   int I[][] = new int[z][z];
   Matrizenrechnung H = new Matrizenrechnung();
   I = H.erstelle_Einheitsmatrix(z,z);
   int n = M[a-1][a-1]*M[b-1][b-1]-M[b-1][a-1]*M[a-1][b-1];
   I[a-1][a-1] = M[b-1][b-1]/n;
   I[a-1][b-1] = M[a-1][b-1]/(-n);
   I[b-1][a-1] = M[b-1][a-1]/(-n);
   I[b-1][b-1] = M[a-1][a-1]/n;
   return(I);
  }
}

class Matrizenrechnung
{
    int[][] Gleich(int M[][])
  {
   int H[][] = new int[M.length][M[0].length];
   for (int l=0; l<M.length;l++)
   for (int k=0; k<M[0].length;k++) H[l][k]=M[l][k];
   return(H);
  }

   int[][] erstelle_Jordanform(int z)
  {
   int M[][] = new int[z][z];
   int h = z;
   int b = 0;
   int p = 0;
   int e = 0;
   for (int l=0; l<z; l++)
   for (int k=0; k<z; k++) M[l][k] = 0;
   do
   {
   do
   {
   b = (int)((Math.random())*h)+1;
   }
   while (b==0);
```

```
   e =(int)(Math.random()*(4*z+1))-(2*z);
   for (int l=p; l<p+b; l++)
   {
   M[l][l] = e;
   if (l!=p+b-1) M[l][l+1] = 1;
   }
   p = p + b;
   if (h!=0) h = Math.min(h-b,b);
   }
   while(p<z);
   return(M);
  }

int[][] erstelle_M_mit_fJN(int H[][])
  {
int z = H.length;
   int e;
   int M[][] = new int[z][z];
   for (int l=0; l<z; l++)
   for (int k=0; k<z; k++)
   M[l][k]=H[l][k];
   for (int l=0; l<z-1; l++)
   if (l==0)
   if (M[0][1]==1) M[0][1]=0;
   else
   {
   M[0][1]=1;
   M[1][1]=M[0][0];
   }
   else if (M[l][l+1]==1) M[l][l+1]=(int)(Math.random()*2);
return(M);
  }

   int[][] erstelle_M_mit_fEW(int H[][])
  {
   int z = H.length;
   int e;
   int M[][] = new int[z][z];
   for (int l=0; l<z; l++)
   for (int k=0; k<z; k++)
   M[l][k]=H[l][k];
   boolean hbf[] = new boolean[4*z+1];
   for (int l=0; l<4*z+1; l++) hbf[l]=false;
   for(int l=0; l<z; l++) hbf[M[l][l]+(2*z)]=true;
   M[z-2][z-1]=0;
   do e =(int)(Math.random()*(4*z+1));
   while (hbf[e]);
   M[z-1][z-1]=e-(2*z);
   return(M);
  }

int[][] erstelle_M_mit_fcP(int H[][])
  {
   int z = H.length;
   int e;
   int M[][] = new int[z][z];
   for (int l=0; l<z; l++)
   for (int k=0; k<z; k++)
   M[l][k]=H[l][k];
   if ((M[z-2][z-1]==0)&(M[z-1][z-1]!=M[z-2][z-2]))
   {
   M[z-2][z-1]=1;
   M[z-1][z-1]=M[z-2][z-2];
}
   else
   {
   M[z-2][z-1]=0;
   do e =(int)(Math.random()*(4*z+1))-(2*z);
   while (e==M[z-1][z-1]);
   M[z-1][z-1]=e;
   }
   return(M);
  }

int[][] erstelle_M_mit_fmP(int H[][])
  {
int z = H.length;
   int e,i,j;
   int M[][] = new int[z][z];
   for (int l=0; l<z; l++)
   for (int k=0; k<z; k++)
   M[l][k]=H[l][k];
   int BF[][]= new int[2][z];
```

```
   for (int l=0; l<z; l++)
   {
   BF[0][l]=1;
   BF[1][l]=0;
   }
   i = 0;
   for (int l=0; l<z-1; l++)
   {
   if (M[l][l+1]==1)
   BF[0][i]=BF[0][i]+1;
   else i=i+1;
   }
   i=0; j=0;
   while (j<z)
   {
   BF[1][i]=BF[0][i]-1+j;
   j=j+BF[0][i];
   i=i+1;
   }
   if (M[0][1]==0)
   {
   M[0][1]=1;
   M[1][1]=M[0][0];
}
   else
   for (int l=0; l<z; l++)
   if (BF[0][l]==BF[0][0])
   M[BF[1][l]-1][BF[1][l]]=0;
   return(M);
  }

   int[][] erstelle_zufDiagMatrix(int z, int s, int g)
  {
   int M[][] = new int[z][s];
   Matrizenrechnung MR = new Matrizenrechnung();
   M = MR.erstelle_Einheitsmatrix(z,s);
   int h = Math.min(z,s);
   for (int l=0;l<h;l++) M[l][l] = (int)(Math.random()*(4*g+1))-(2*g);
   return(M);
  }

   int[][] erstelle_zufMatrix(int z, int s, int g)
  {
   int M[][] = new int[z][s];
   Matrizenrechnung MR = new Matrizenrechnung();
   M = MR.erstelle_Einheitsmatrix(z,s);
   for (int l=0;l<z;l++)
   for (int k=0;k<s;k++) M[l][k] = (int)(Math.random()*(2*g+1))-g;
   return(M);
  }

int[][] erstelle_Einheitsmatrix(int m, int n)
  {
int M[][] = new int[m][n];
    for (int l=0; l<m; l++)
     for (int k=0; k<n; k++)
     if (l!=k) M[l][k]=0;
     else M[l][k]=1;
    return(M);
  }

   int[][] Matrizenmultiplikation(int A[][], int B[][])
  {
   int z = A.length;
   int s = A[0].length;
   int b = B[0].length;
   int M[][] = new int[z][b];
   for (int l=0; l<=z-1; l++)
   for (int m=0; m<=b-1; m++)
   {int h=0;
   for (int n=0; n<=s-1; n++) h=h+A[l][n]*B[n][m];
   M[l][m]=h;};
   return(M);
  }

   int[][] Transponierung(int M[][])
  {
   int z = M.length;
   int s = M[0].length;
   int H[][] = new int[s][z];
   for (int l=0; l<=z-1; l++)
   for (int k=0; k<=s-1; k++)
   H[k][l]=M[l][k];
```

```
   return(H);
  }

int[][] Negation(int M[][])
  {
   int z = M.length;
   int s = M[0].length;
   int H[][] = new int[z][s];
   for (int l=0; l<=z-1; l++)
   for (int k=0; k<=s-1; k++)
   H[l][k]=-M[l][k];
   return(H);
  }
}

class Generierung
{
int[][][] generiere_LR(int z)
  {
Generierung G = new Generierung();
return(G.generiere_LR(z,z));
  }

int[][][] generiere_LR(int z, int a)
  {
   Endomorphismen Endo = new Endomorphismen();
   Matrizenrechnung MR = new Matrizenrechnung();
   int L[][] = new int[z][z];
   int R[][] = new int[z][z];
   L = MR.erstelle_Einheitsmatrix(z,z);
   R = MR.erstelle_Einheitsmatrix(z,z);
   int h;
   for (int l=0; l<2*a; l++)
   {
   h = (int)(Math.random()*4);
     switch (h)
     {
   case 0 :
   {
      int h1 = (int)(Math.random()*z)+1;
      int h2; do h2 = (int)(Math.random()*z)+1; while (h1==h2);
      int h3; do h3 = (int)(Math.random()*(2*z+1))-z; while (h3==0);
      int h4; do h4 = (int)(Math.random()*(2*z+1))-z; while (h4==0);
      int H[][] = new int[z][z];
      H = Endo.Kombination(z,h1,h2,h3,h4);
      L = MR.Matrizenmultiplikation(H,L);
      R = MR.Matrizenmultiplikation(R,Endo.Invertierung(H,h1,h2));
        break;
     }
   case 1 :
   {
      int h1 = (int)(Math.random()*z)+1;
      int h2; do h2 = (int)(Math.random()*z)+1; while (h1==h2);
      int h3 = (int)(Math.random()*(2*z+1))-z;
      int H[][] = new int[z][z];
      H = Endo.vielfache_Zeilenaddition(z,h1,h2,h3);
      L = MR.Matrizenmultiplikation(H,L);
      R = MR.Matrizenmultiplikation(R,Endo.Invertierung(H,h1,h2));
        break;
   }
   case 2 :
   {
      int h1 = (int)(Math.random()*z)+1;
      int h2; do h2 = (int)(Math.random()*z)+1; while (h1==h2);
      int H[][] = new int[z][z];
      H = Endo.Zeilentausch(z,h1,h2);
      L = MR.Matrizenmultiplikation(H,L);
      R = MR.Matrizenmultiplikation(R,Endo.Invertierung(H,h1,h2));
        break;
   }
   case 3 :
   {
      int h1 = (int)(Math.random()*z)+1;
      int H[][] = new int[z][z];
      H = Endo.Vorzeichenwechsel(z,h1);
      L = MR.Matrizenmultiplikation(H,L);
      R = MR.Matrizenmultiplikation(R,H);
      break;
   }
}
}
int H[][][] = new int[2][z][z];
H[0]=L;
```

```
H[1]=R;
return(H);
     }

int[][] Gemmatrix(int A[][], int B[][])
  {
   int H[][] = new int[A.length][A[0].length+B[0].length];
   for (int l=0; l<H.length; l++)
   for (int k=0; k<H[l].length; k++)
   if (k<A[l].length) H[l][k]=A[l][k];
   else H[l][k]=B[l][k-A[l].length];
   return(H);
  }

int[][] Gemmatrix(int A[][], int B[][],int C[][])
  {
   int H[][] = new int[A.length][A[0].length+B[0].length+C[0].length];
   Generierung G = new Generierung();
   H = G.Gemmatrix(A,G.Gemmatrix(B,C));
   return(H);
  }

boolean Ueberpruefung(int H[][], int A[][][], int pa, int pe, int z, int s)
  {
   Generierung u = new Generierung();
   if (H[z][s]==A[pa][z][s])
   if (s<H[0].length-1) return(u.Ueberpruefung(H,A,pa,pe,z,s+1));
   else if (z<H.length-1) return(u.Ueberpruefung(H,A,pa,pe,z+1,0));
      else return(false);
   else if (pa<pe) return(u.Ueberpruefung(H,A,pa+1,pe,0,0));
    else return(true);
  }

    boolean Ueberpruefung2(int H[][], int z, int s, int g)
  {
   Generierung u = new Generierung();
   if (Math.abs(H[z][s])>g) return(false);
   else if (s<H[0].length-1) return(u.Ueberpruefung2(H,z,s+1,g));
    else if (z<H.length-1) return(u.Ueberpruefung2(H,z+1,0,g));
     else return(true);
  }

boolean Ueberpruefung3(int A[][], int B[][])
  {
   boolean h = false;
   for (int l=0; l<A.length; l++)
   for (int k = 0; k<A[0].length; k++)
   if (A[l][k]!=B[l][k]) h = true;
   return(h);
  }

int[][][] Speicherung(int H[][], int A[][][], int s)
  {
   for (int l=0; l<H.length; l++)
   for (int k=0; k<H[l].length; k++)
   A[s][l][k]=H[l][k];
   return(A);
  }

int[][] KB_Start(int v, int b)
  {
   int H[][] = new int[v][v];
   Matrizenrechnung MR = new Matrizenrechnung();
   H = MR.erstelle_Einheitsmatrix(v,v);
   if (v!=b) for (int l=b; l<v; l++) H[l][l]=0;
   return(H);
  }

   int[][][] erstelle_LGS(int z, int s, int r, boolean inhom)
  {
int h = Math.max(z,s);
int LGS[][][] = new int[2][h][h];
for (int l=0; l<2; l++)
for (int k=0; k<h; k++)
for (int j=0; j<h; j++) LGS[l][k][j]=0;
for (int l=0; l<r; l++)
{
LGS[0][l][l]=1;
if (inhom) LGS[1][l][0]=(int)(Math.random()*(2*z+1))-z;
}
for (int l=0; l<s-r; l++)
{
LGS[1][l+r][l+1]=1;
```

```
}
    return(LGS);
  }
}

class A_Typen
{

// Hier findet sich der Erstellungsalgorithmus für den Bereich der ähnlichen Matrizen 1-5 und den der LGS 6-7
// Die Aufgabenerstellung folgt verschiedenen Schemata.

// Vorgabe von A und B; Beispiellösung P; kein Falsch_B benötigt
void Schema1(int a, int g, int z, int Starttyp, String AGS, String AGSt)
  {
   Ausgabe aus = new Ausgabe();
   aus.AusMed("Bei der Auswahl dieser Option wird lediglich eine richtige und keine falsche Lösung produziert!\n");
int x = 3; // A,B,P sind drei Matrizen
int A[][][][] = new int[x][a][z][z]; // A ist der Aufgabenspeicher ein 4-dim. Feld
   Generierung G = new Generierung(); // Erzeugen der benötigten Objekte
   Matrizenrechnung MR = new Matrizenrechnung();
   int stelle = 0; // Setzen des Zählers
   int FLR[][][] = new int[2][z][z]; // Reservierung des Speicherplatzes für L,R
   while(stelle<a) // Abbruchbedingung
  {
  int Aufgabe[][][] = new int[3][z][z];
  switch(Starttyp)
  {
  case 11:
  {
  Aufgabe[0] = MR.erstelle_zufMatrix(z,z,z);
  break;
  }
  case 12:
  {
  FLR = G.generiere_LR(z);
  Aufgabe[0] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(MR.erstelle_Jordanform(z),FLR[1]));
  break;
  }
  case 13:
  {
  FLR = G.generiere_LR(z);
  Aufgabe[0] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(MR.erstelle_zufDiagMatrix(z,z,z),>>
  >> FLR[1]));
  break;
  }
  }
// Ausgabe[0] enthält jetzt A
  FLR = G.generiere_LR(z); // Erstellung von L,R
   Aufgabe[1] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(Aufgabe[0],FLR[1]));
   Aufgabe[2] = FLR[0];
   // Ausgabe[1] enthält B, Ausgabe[2] enthält P
   // Abfrage, ob es die Aufgage schon gibt (Ueberpruefung)
   // Abfrage, ob die Aufgabe Beträge größer g enthält (Ueberpruefung2)
   // Abfrage, ob A=B (Ueberpruefung3)
   int H[][] = new int[z][2*z];
   H = G.Gemmatrix(Aufgabe[1],Aufgabe[2]);
   if (G.Ueberpruefung(Aufgabe[0],A[0],0,stelle,0,0) & G.Ueberpruefung2(H,0,0,g) & G.Ueberpruefung3(Aufgabe[0],>>
   >> Aufgabe[1]))
   {
   for (int l=0; l<3; l++) A[l][stelle] = Aufgabe[l];
   stelle = stelle +1;
   }
   // Besteht die Aufgabe des Test, dann wird sie in den Speicher geschrieben und der Zähler heraufgesetzt.
   // Ansonsten wird sie einfach gelöscht und der Zähler behält seinen Wert.
   }
   // Beginn der Ausgabe
   if (tt) aus.ausdruck_S1(A,a,z,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S1(A,a,z,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S1(A,a,z,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S1(A,a,z,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S1(A,a,z,Starttyp,AGSt,5);
  }

// Vorgabe von A, B und Falsch_B; Lösung B; kein P benötigt
void Schema2(int a, int g, int z, int rl, int fl, int Starttyp, String AGS, String AGSt)
  {
   Ausgabe aus = new Ausgabe();
   int x = 1+rl + fl; //Eine Startmatrix, rl richtige Matrizen, fl falsche Matrizen
   int A[][][][] = new int[x][a][z][z]; // A ist der Aufgabenspeicher ein 4-dim. Feld
   Generierung G = new Generierung(); // Erzeugen der benötigten Objekte
   Matrizenrechnung MR = new Matrizenrechnung();
   int stelle = 0; // Setzen des Zählers
   int FLR[][][] = new int[2][z][z]; // Reservierung des Speicherplatzes für L,R
```

```
 while(stelle<a) // Abbruchbedingung
 {
 // Diese Aufgabentypen arbeiten mit Jordanformen als Startmatrix und bauen daraus A.
 int J[][] = new int[z][z];
 if ((Starttyp!=27)&(Starttyp!=210)) J = MR.erstelle_Jordanform(z); // Besonderheit von 27 und 210 beachten
 else if (Starttyp==27) J = MR.erstelle_zufDiagMatrix(z,z,z);
 else
 {
 int za;
 do
 {
 za=0;
 J = MR.erstelle_Jordanform(z);
 for (int l=0; l<z; l++) J[l][l]=0;
 for (int l=0; l<z; l++)
 for (int k=0; k<z; k++)
 za=za+J[l][k];
 }
 while (za==0);
 }
 FLR = G.generiere_LR(z,2);
 int Aufgabe[][][] = new int[x][z][z];
 if (Starttyp<25) Aufgabe[0] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(J,FLR[1]));
 else Aufgabe[0]= MR.Gleich(J);
 // Ausgabe[0] enthält jetzt schon A, Erstellung von L,R
 for (int l=1; l<rl+1; l++)
 {
 FLR = G.generiere_LR(z);
 Aufgabe[l] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(Aufgabe[0],FLR[1]));
 }
 // Im Feld Aufgabe stehen also nach A, die richtigen Lösungen B
 // Abhängig von der gewählten Option werden für falschen Lösungen nun andere Jordanformen erzeugt.
 for (int l=0; l<fl; l++)
 {
 int NJ[][] = new int[z][z];
 switch(Starttyp)
{
case 21:
{
NJ = MR.erstelle_M_mit_fEW(J);
break;
}
case 22:
{
NJ = MR.erstelle_M_mit_fcP(J);
break;
}
case 23:
{
NJ = MR.erstelle_M_mit_fmP(J);
break;
}
case 24:
{
NJ = MR.erstelle_M_mit_fJN(J);
break;
}
case 25:
{
NJ = MR.erstelle_M_mit_fEW(J);
break;
}
case 26:
{
NJ = MR.erstelle_M_mit_fcP(J);
break;
}
case 27:
{
NJ = MR.erstelle_M_mit_fJN(J);
break;
}
case 28:
{
NJ = MR.erstelle_M_mit_fJN(J);
break;
}
case 29:
{
NJ = MR.erstelle_M_mit_fmP(J);
break;
}
case 210:
```

```
  {
  NJ = MR.erstelle_M_mit_fmP(J);
  int za = 0;
   for (int hl=0; hl<z; hl++)
   for (int k=0; k<z; k++)
   za=za+NJ[hl][k];
   if (za==0) NJ[0][0]=1;
  break;
  }
  }
   FLR = G.generiere_LR(z);
   Aufgabe[rl+1+l] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(NJ,FLR[1]));
   } // Die letzten Plätze im Feld Aufgabe nehmen die falschen Lösungen ein.
   // Jetzt wird überprüft, dass der Betrag jedes einzelnen Wertes kleiner als g ist.
   boolean w = true;
   for (int l=1; l<x; l++)
   {
   if (!(G.Ueberpruefung2(Aufgabe[l],0,0,g))) w = false;
   }
   // Nun kommt der A!=B Test.
   for (int l=1; l<rl+1; l++)
   {
   if (!(G.Ueberpruefung3(Aufgabe[0],Aufgabe[l]))) w = false;
   }
   // Zusätzlich kommt die Abfrage, ob es die erste der Matrizen B schon einmal gibt. Das schließt gleiche >>
   >> Aufgaben aus.
   if (G.Ueberpruefung(Aufgabe[1],A[1],0,stelle,0,0) & w)
   {
   for (int l=0; l<x; l++) A[l][stelle] = Aufgabe[l];
   stelle = stelle +1;
   }
   // Besteht die Aufgabe des Test, dann wird sie in den Speicher geschrieben und der Zähler heraufgesetzt.
   // Ansonsten wird sie einfach gelöscht und der Zähler behält seinen Wert.
   }
   // Beginn der Ausgabe
   if (tt) aus.ausdruck_S2(A,a,z,rl,x,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S2(A,a,z,rl,x,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S2(A,a,z,rl,x,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S2(A,a,z,rl,x,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S2(A,a,z,rl,x,Starttyp,AGSt,5);
  }

// Vorgabe von B; Lösung A (Einträge); kein P und kein Falsch_B benötigt
void Schema3(int a, int g, int z, int Starttyp, String AGS, String AGSt)
  {
   Ausgabe aus = new Ausgabe();
   aus.AusMed("Bei der Auswahl dieser Option wird lediglich eine richtige und keine falsche Lösung produziert mit >>
   >> Ausnahme der Okusonausgabe,\nbei der immer eine richtige und eine falsche Lösung produziert werden.\n");
int x = 3; // A,B sind zwei Matrizen
int A[][][][] = new int[x][a][z][z]; // A ist der Aufgabenspeicher ein 4-dim. Feld
   Generierung G = new Generierung(); // Erzeugen der benötigten Objekte
   Matrizenrechnung MR = new Matrizenrechnung();
   int stelle = 0; // Setzen des Zählers
   int FLR[][][] = new int[2][z][z]; // Reservierung des Speicherplatzes für L,R
   while(stelle<a) // Abbruchbedingung
  {
  int Aufgabe[][][] = new int[2][z][z];
  switch(Starttyp)
  {
  case 31:
  {
  Aufgabe[0] = MR.erstelle_Jordanform(z);
  break;
  }
  case 32:
  {
  Aufgabe[0] = MR.erstelle_Jordanform(z);
  break;
  }
  case 33:
  {
  Aufgabe[0] = MR.erstelle_Jordanform(z);
  break;
  }
  case 34:
  {
  int za;
   do
   {
   za=0;
   Aufgabe[0] = MR.erstelle_Jordanform(z);
   for (int l=0; l<z; l++) Aufgabe[0][l][l]=0;
   for (int l=0; l<z; l++)
```

```
   for (int k=0; k<z; k++)
   za=za+Aufgabe[0][l][k];
   }
   while (za==0);
  break;
  }
  }
// Ausgabe[0] enthält jetzt A
  FLR = G.generiere_LR(z); // Erstellung von L,R
   Aufgabe[1] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(Aufgabe[0],FLR[1]));
   // Ausgabe[1] enthält B
   // Abfrage, ob es die Aufgabe schon gibt (Ueberpruefung)
   // Abfrage, ob die Aufgabe Beträge größer g enthält (Ueberpruefung2)
   // Abfrage, ob A=B (Ueberpruefung3)
   if (G.Ueberpruefung(Aufgabe[1],A[1],0,stelle,0,0) & G.Ueberpruefung2(Aufgabe[1],0,0,g) & >>
   >> G.Ueberpruefung3(Aufgabe[0],Aufgabe[1]))
   {
   for (int l=0; l<2; l++) A[l][stelle] = Aufgabe[l];
   stelle = stelle +1;
   }
   // Besteht die Aufgabe des Test, dann wird sie in den Speicher geschrieben und der Zähler heraufgesetzt.
   // Ansonsten wird sie einfach gelöscht und der Zähler behält seinen Wert.
   }
   // Beginn der Ausgabe
   if (tt) aus.ausdruck_S3(A,a,z,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S3(A,a,z,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S3(A,a,z,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S3(A,a,z,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S3(A,a,z,Starttyp,AGSt,5);
  }

// Vorgabe von A; Lösung B mit P; kein Falsch_B benötigt
void Schema4(int a, int g, int z, int Starttyp, String AGS, String AGSt)
  {
   Ausgabe aus = new Ausgabe();
   aus.AusMed("Bei der Auswahl dieser Option wird lediglich eine richtige und keine falsche Lösung produziert!\n");
int x = 3; // A,B,P sind drei Matrizen
int A[][][][] = new int[x][a][z][z]; // A ist der Aufgabenspeicher ein 4-dim. Feld
   Generierung G = new Generierung(); // Erzeugen der benötigten Objekte
   Matrizenrechnung MR = new Matrizenrechnung();
   int stelle = 0; // Setzen des Zählers
   int FLR[][][] = new int[2][z][z]; // Reservierung des Speicherplatzes für L,R
   while(stelle<a) // Abbruchbedingung
  {
  int Aufgabe[][][] = new int[3][z][z];
  switch(Starttyp)
  {
  case 41:
  {
  Aufgabe[0] = MR.erstelle_zufDiagMatrix(z,z,z);
  break;
  }
  case 42:
  {
  Aufgabe[0] = MR.erstelle_zufDiagMatrix(z,z,z);
  break;
  }
  case 43:
  {
  Aufgabe[0] = MR.erstelle_Jordanform(z);
  break;
  }
  }
// Ausgabe[0] enthält jetzt A
  FLR = G.generiere_LR(z); // Erstellung von L,R
   Aufgabe[1] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(Aufgabe[0],FLR[1]));
   Aufgabe[2] = FLR[0];
   // Ausgabe[1] enthält B, Ausgabe[2] enthält P
   // Abfrage, ob es die Aufgage schon gibt (Ueberpruefung)
   // Abfrage, ob die Aufgabe Beträge größer g enthält (Ueberpruefung2)
   // Abfrage, ob A=B (Ueberpruefung3)
   int H[][] = new int[z][2*z];
   H = G.Gemmatrix(Aufgabe[1],Aufgabe[2]);
   if (G.Ueberpruefung(Aufgabe[1],A[1],0,stelle,0,0) & G.Ueberpruefung2(H,0,0,g) & G.Ueberpruefung3(Aufgabe[0],>>
   >> Aufgabe[1]))
   {
   for (int l=0; l<3; l++) A[l][stelle] = Aufgabe[l];
   stelle = stelle +1;
   }
   // Besteht die Aufgabe den Test, dann wird sie in den Speicher geschrieben und der Zähler heraufgesetzt.
   // Ansonsten wird sie einfach gelöscht und der Zähler behält seinen Wert.
   }
   // Beginn der Ausgabe
```

```
   if (tt) aus.ausdruck_S4(A,a,z,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S4(A,a,z,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S4(A,a,z,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S4(A,a,z,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S4(A,a,z,Starttyp,AGSt,5);
  }

// Vorgabe von A und P; Lösung B; kein Falsch_B benötigt
   void Schema5(int a, int g, int z, int Starttyp, String AGS, String AGSt)
  {
   Ausgabe aus = new Ausgabe();
   aus.AusMed("Bei der Auswahl dieser Option wird lediglich eine richtige und keine falsche Lösung produziert!\n");
int x = 3; // A,B,P sind drei Matrizen
int A[][][][] = new int[x][a][z][z]; // A ist der Aufgabenspeicher ein 4-dim. Feld
   Generierung G = new Generierung(); // Erzeugen der benötigten Objekte
   Matrizenrechnung MR = new Matrizenrechnung();
   int stelle = 0; // Setzen des Zählers
   int FLR[][][] = new int[2][z][z]; // Reservierung des Speicherplatzes für L,R
   while(stelle<a) // Abbruchbedingung
  {
  int Aufgabe[][][] = new int[3][z][z];
  switch(Starttyp)
  {
  case 51:
  {
  Aufgabe[0] = MR.erstelle_zufDiagMatrix(z,z,z);
  break;
  }
  }
// Ausgabe[0] enthält jetzt A
  FLR = G.generiere_LR(z); // Erstellung von L,R
   Aufgabe[1] = MR.Matrizenmultiplikation(FLR[0],MR.Matrizenmultiplikation(Aufgabe[0],FLR[1]));
   Aufgabe[2] = FLR[0];
   // Ausgabe[1] enthält B, Ausgabe[2] enthält P
   // Abfrage, ob es die Aufgage schon gibt (Ueberpruefung)
   // Abfrage, ob die Aufgabe Beträge größer g enthält (Ueberpruefung2)
   // Abfrage, ob A=B (Ueberpruefung3)
   int H[][] = new int[z][2*z];
   H = G.Gemmatrix(Aufgabe[1],Aufgabe[2]);
   if (G.Ueberpruefung(Aufgabe[0],A[0],0,stelle,0,0) & G.Ueberpruefung2(H,0,0,g) & G.Ueberpruefung3(Aufgabe[0],>>
   >> Aufgabe[1]))
   {
   for (int l=0; l<3; l++) A[l][stelle] = Aufgabe[l];
   stelle = stelle +1;
   }
   // Besteht die Aufgabe den Test, dann wird sie in den Speicher geschrieben und der Zähler heraufgesetzt.
   // Ansonsten wird sie einfach gelöscht und der Zähler behält seinen Wert.
   }
   // Beginn der Ausgabe
   if (tt) aus.ausdruck_S5(A,a,z,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S5(A,a,z,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S5(A,a,z,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S5(A,a,z,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S5(A,a,z,Starttyp,AGSt,5);
  }


void Schema6(int a, int g, int z, int s, int r, int Starttyp, String AGS, String AGSt)
  {
    Ausgabe aus = new Ausgabe();
    aus.AusMed("Bei der Auswahl dieser Option wird lediglich eine richtige und keine falsche Lösung produziert!\n");
    boolean ih;
    if (Starttyp==62) ih=true; else ih=false;
    Matrizenrechnung MR = new Matrizenrechnung();
    Generierung G = new Generierung();
    int stelle = 0;
    int h = Math.max(z,s);
    int LGS_H[][][] = new int[2][h][h];
    int KM[][] = new int[z][s];
    int KM_H[][] = new int[h][h];
    int Loesung[][] = new int[s][s-r+1];
    int Loesung_H[][] = new int[h][h];
    int B[][] = new int[z][1];
    int B_H[][] = new int[h][1];
    int ZM[][][] = new int[2][s][s];
    int ZU[][][] = new int[2][z][z];
    int A[][][] = new int[a][h][2*h+1];
    for (int l=0; l<a; l++)
for (int k=0; k<h; k++)
for (int j=0; j<h; j++) A[l][k][j]=0;
    while (stelle<a)
   {
   LGS_H = G.erstelle_LGS(z,s,r,ih);
```

```
   for (int l=0; l<z; l++)
     for (int k=0; k<s; k++) KM[l][k] = LGS_H[0][l][k];
    for (int l=0; l<s; l++)
     for (int k=0; k<s-r+1; k++) Loesung[l][k] = LGS_H[1][l][k];
     for (int l=0; l<z; l++) if (l<s) B[l][0] = LGS_H[1][l][0];
     else B[l][0] = 0;
     for (int l=0; l<h; l++)
for (int k=0; k<h; k++)
{
KM_H[l][k]=0;
Loesung_H[l][k]=0;
}
for (int l=0; l<h; l++) B_H[l][0]=0;
   ZM = G.generiere_LR(s);
   KM = MR.Matrizenmultiplikation(KM,ZM[1]);
   Loesung = MR.Matrizenmultiplikation(ZM[0],Loesung);
   ZU = G.generiere_LR(z);
   KM = MR.Matrizenmultiplikation(ZU[0],KM);
   B = MR.Matrizenmultiplikation(ZU[0],B);
   for (int l=0; l<z; l++)
     for (int k=0; k<s; k++) KM_H[l][k] = KM[l][k];
    for (int l=0; l<s; l++)
     for (int k=0; k<s-r+1; k++) Loesung_H[l][k] = Loesung[l][k];
   for (int l=0; l<z; l++) B_H[l][0] = B[l][0];
   int H[][] = new int[h][2*h+1];
   H = G.Gemmatrix(KM_H,Loesung_H,B_H);
   if (G.Ueberpruefung(H,A,0,stelle,0,0) & G.Ueberpruefung2(H,0,0,g))
     {
     A = G.Speicherung(H,A,stelle);
     stelle=stelle+1;
     }
   }
   if (tt) aus.ausdruck_S6(A,z,a,s,r,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S6(A,z,a,s,r,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S6(A,z,a,s,r,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S6(A,z,a,s,r,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S6(A,z,a,s,r,Starttyp,AGSt,5);
      }

   void Schema7(int a, int g, int z, int s, int r, int rl, int fl, int Starttyp, String AGS, String AGSt)
  {
   int mzs = Math.min(z,s);
   int x = 1+rl + fl; //Eine Startmatrix, rl richtige Matrizen, fl falsche Matrizen
   int A[][][][] = new int[x][a][z][s]; // A ist der Aufgabenspeicher ein 4-dim. Feld
   Generierung G = new Generierung(); // Erzeugen der benötigten Objekte
   Matrizenrechnung MR = new Matrizenrechnung();
   Ausgabe aus = new Ausgabe();
   int stelle = 0; // Setzen des Zählers
   int FLR_s[][][] = new int[2][s][s]; // Reservierung des Speicherplatzes für L,R
   int FLR_z[][][] = new int[2][z][z];
   int Kern[][][] = new int [a][s][s];
   while(stelle<a) // Abbruchbedingung
   {
   // Diese Aufgabentypen arbeiten mit Diagonalmatrizen als Startmatrix und bauen daraus A und B
   int D[][] = new int[z][s];
   for (int l=0; l<z; l++)
   for (int k=0; k<s; k++)
   D[l][k]=0;
   for (int l=0; l<r; l++)
   D[l][l]=1;
   int Aufgabe[][][] = new int [x][z][s];
   FLR_s = G.generiere_LR(s);
   if (Starttyp==71)
   {
   for (int l=0; l<s; l++)
   for (int k=0; k<s; k++)
   if ((l==k)&(l>=r)) Kern[stelle][l][k]=1;
   else Kern[stelle][l][k]=0;
   D=MR.Matrizenmultiplikation(D,FLR_s[0]);
   Kern[stelle]=MR.Matrizenmultiplikation(FLR_s[1],Kern[stelle]);
   for (int l=0; l<rl; l++)
   {
   FLR_z = G.generiere_LR(z);
   Aufgabe[l+1]=MR.Matrizenmultiplikation(FLR_z[0],D);
   }
   }  // Erstellung der richtigen Lösungen abgeschlossen
   if (Starttyp==72)
   {
   Aufgabe[0]=MR.Gleich(D);
   for (int l=0; l<rl; l++)
   {
   FLR_z = G.generiere_LR(z);
   Aufgabe[l+1]=MR.Matrizenmultiplikation(FLR_z[0],MR.Matrizenmultiplikation(D,FLR_s[0]));
```

```
   }
   } // Erstellung der richtigen Lösungen abgeschlossen

   // Erstellung falscher Lösungen durch Veränderung des Ranges;
   for (int l=0; l<fl; l++)
   {
   int FD[][] = new int[z][s];
   int fr;
   int zfr = (int)(Math.random()*2);
   if (zfr==0) if (r!=0) fr=r-1;
   else fr=1;
   else if(r!=mzs) fr=r+1;
   else fr=mzs-1;
   for (int i=0; i<z; i++)
   for (int k=0; k<s; k++)
   FD[i][k]=0;
   boolean hb = false;
   if ((Starttyp==71)&(r!=mzs))
   if ((int) (Math.random()*2)==0)
   {
   fr=r;
   hb = true;
   }
   for (int i=0; i<fr; i++) FD[i][i]=1;
   if (hb)
   {
   int tp1 = (int) (Math.random()*fr);
   int tp2 = (int) (Math.random()*(mzs-fr))+fr;
   FD[tp1][tp1]=0;
   FD[tp2][tp2]=1;
   }
   FLR_z = G.generiere_LR(z);
   Aufgabe[rl+1+l] = MR.Matrizenmultiplikation(FLR_z[0],MR.Matrizenmultiplikation(FD,FLR_s[0]));
   } // Die letzten Plätze im Feld Aufgabe nehmen die falschen Lösungen ein.
   // Jetzt wird überprüft, dass der Betrag jedes einzelnen Wertes kleiner als g ist.
   boolean w = true;
   for (int l=1; l<x; l++)
   {
   if (!(G.Ueberpruefung2(Aufgabe[l],0,0,g))) w = false;
   }
   // Zusätzlich kommt die Abfrage, ob es die erste richtige Matrix schon einmal gibt. Das schließt gleiche>>
   >> Aufgaben aus.
   if (G.Ueberpruefung(Aufgabe[1],A[1],0,stelle,0,0) & w)
   {
   for (int l=0; l<x; l++) A[l][stelle] = Aufgabe[l];
   stelle = stelle +1;
   }
   // Besteht die Aufgabe des Test, dann wird sie in den Speicher geschrieben und der Zähler heraufgesetzt.
   // Ansonsten wird sie einfach gelöscht und der Zähler behält seinen Wert.
   }
   // Beginn der Ausgabe
   if (tt) aus.ausdruck_S7(A,z,a,s,r,rl,x,Kern,Starttyp,AGS,1);
   if (ma) aus.ausdruck_S7(A,z,a,s,r,rl,x,Kern,Starttyp,AGS,2);
   if (oo) aus.ausdruck_S7(A,z,a,s,r,rl,x,Kern,Starttyp,AGS,3);
   if (tc) aus.ausdruck_S7(A,z,a,s,r,rl,x,Kern,Starttyp,AGSt,4);
   if (ok) aus.ausdruck_S7(A,z,a,s,r,rl,x,Kern,Starttyp,AGSt,5);
  }

// Je nach Aufgabe ruft AM das passende Schema auf.
void AM(int a, int g, int z, int r, int f)
  {
   A_Typen S = new A_Typen(); // Aufrufe der benötigten Objekte
   Generierung G = new Generierung();
   int Starttyp;
   int s = 0; // AS wird halt auch mit s aufgerufen.
   String AGS[] = new String[2]; // Initialisierung der Aufgabenstellung
   // Beginn der Abfrage
   if (jc_amw.getSelectedItem().equals("keine Optionen"))
   {
   Starttyp = 11;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema1(a,g,z,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_amw.getSelectedItem().equals("ganzzahlige Eigenwerte"))
   {
   Starttyp = 12;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema1(a,g,z,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_amw.getSelectedItem().equals("Diagonalmatrizen"))
   {
   Starttyp = 13;
   AGS = S.AS(Starttyp,z,s,r);
```

```
S.Schema1(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("falsche Eigenwerte"))
{
Starttyp = 21;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("falsches char. Polynom"))
{
Starttyp = 22;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("falsches Minimalpolynom"))
{
Starttyp = 23;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("falsche Jordanform"))
{
Starttyp = 24;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Eigenwerte bestimmen"))
{
Starttyp = 31;
AGS = S.AS(Starttyp,z,s,r);
S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("char. Polynom bestimmen"))
{
Starttyp = 32;
AGS = S.AS(Starttyp,z,s,r);
S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Eigenvektoren bestimmen"))
{
Starttyp = 41;
AGS = S.AS(Starttyp,z,s,r);
S.Schema4(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Ausschluss über Eigenwerte"))
{
Starttyp = 25;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Ausschluss über char. Polynom"))
{
Starttyp = 26;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Diagonalisierung"))
{
Starttyp = 42;
AGS = S.AS(Starttyp,z,s,r);
S.Schema4(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Diagonalisierbarkeit"))
{
Starttyp = 27;
AGS = S.AS(Starttyp,z,s,r);
S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Umkehr der Diagonalisierung"))
{
Starttyp = 51;
AGS = S.AS(Starttyp,z,s,r);
S.Schema5(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Jordanform bestimmen"))
{
Starttyp = 43;
AGS = S.AS(Starttyp,z,s,r);
S.Schema4(a,g,z,Starttyp,AGS[0],AGS[1]);
}
if (jc_amw.getSelectedItem().equals("Ausschluss über Jordanform"))
{
Starttyp = 28;
```

```
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_amw.getSelectedItem().equals("Minimalpolynom bestimmen"))
   {
   Starttyp = 33;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_amw.getSelectedItem().equals("Ausschluss über Minimalpolynom"))
   {
   Starttyp = 29;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_amw.getSelectedItem().equals("Nilpotenzgrad bestimmen"))
   {
   Starttyp = 34;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema3(a,g,z,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_amw.getSelectedItem().equals("Ausschluss über Nilpotenzgrad"))
   {
   Starttyp = 210;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema2(a,g,z,r,f,Starttyp,AGS[0],AGS[1]);
   }
  }

void LGS(int a, int g, int z, int s, int r, int rl, int fl)
  {
   A_Typen S = new A_Typen(); // Aufrufe der benötigten Objekte
   int Starttyp; // Initialisierung der Startmatrix
   String AGS[] = new String[2]; // Initialisierung der Aufgabenstellung
   // Beginn der Abfrage
   if (jc_lgsw.getSelectedItem().equals("Homogene LGS"))
   {
   Starttyp = 61;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema6(a,g,z,s,r,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_lgsw.getSelectedItem().equals("Inhomogene LGS"))
   {
   Starttyp = 62;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema6(a,g,z,s,r,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_lgsw.getSelectedItem().equals("Kern-Basis"))
   {
   Starttyp = 71;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema7(a,g,z,s,r,rl,fl,Starttyp,AGS[0],AGS[1]);
   }
   if (jc_lgsw.getSelectedItem().equals("Rang"))
   {
   Starttyp = 72;
   AGS = S.AS(Starttyp,z,s,r);
   S.Schema7(a,g,z,s,r,rl,fl,Starttyp,AGS[0],AGS[1]);
   }
  }

String[] AS(int Starttyp, int z, int s, int r)
  {
   String AGS = new String();
   String AGSt = new String();
   if (Starttyp==11)
   {
   AGSt = "Zeigen Sie, dass die folgenden Matrizen $A,B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ ähnlich sind,>>
   >> \nindem Sie eine Matrix $P \\in GL("+z+";\\mathbb{C})$ angeben, sodass $B = P \\circ A \\circ P^{-1}$ >>
   >> gilt.\n\n";
   AGS = "Zeigen Sie, dass die folgenden Matrizen A,B aus M("+z+" x "+z+";Z) ähnlich sind,\nindem Sie eine Matrix >>
   >> P aus GL("+z+";C) angeben, sodass B = P * A * P^{-1} gilt.\n\n";
   }
   if (Starttyp==12)
   {
   AGSt = "Zeigen Sie, dass die folgenden Matrizen $A,B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ ähnlich sind,>>
   >> \nindem Sie eine Matrix $P \\in GL("+z+";\\mathbb{C})$ angeben, sodass $B = P \\circ A \\circ P^{-1}$ >>
   >> gilt.\n Hinweis: Die Eigenwerte von $A,B$ liegen in $\\mathbb{Z}$.\n\n";
   AGS = "Zeigen Sie, dass die folgenden Matrizen A,B aus M("+z+" x "+z+";Z) ähnlich sind,\nindem Sie eine Matrix >>
   >> P aus GL("+z+";C) angeben, sodass B = P * A * P^{-1} gilt.\n Hinweis: Die Eigenwerte von A,B liegen in Z.\n\n";
   }
   if (Starttyp==13)
   {
```

```
AGSt = "Zeigen Sie, dass die folgenden Matrizen $A,B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ ähnlich sind,>>
>> \nindem Sie eine Matrix $P \\in GL("+z+";\\mathbb{C})$ angeben, sodass $B = P \\circ A \\circ P^{-1}$ gilt. >>
>> \n Hinweis: Die Eigenwerte von $A,B$ liegen in $\\mathbb{Z}$. $A,B$ sind diagonalisierbar.\n\n";
AGS = "Zeigen Sie, dass die folgenden Matrizen A,B aus M("+z+" x "+z+";Z) ähnlich sind,\nindem Sie eine Matrix >>
>> P aus GL("+z+";C) angeben, sodass B = P * A * P^{-1} gilt.\n Hinweis: Die Eigenwerte von A,B liegen in Z. >>
>> A,B sind diagonalisierbar.\n\n";
}
if (Starttyp==21)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ sind ähnlich zur Matrix $A >>
>> \\in M("+z+" \\times "+z+";\\mathbb{Z})$?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) sind ähnlich zur Matrix A aus M("+z+" x "+z+";Z)>>
>>?\n\n";
}
if (Starttyp==22)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ sind ähnlich zur Matrix $A >>
>> \\in M("+z+" \\times "+z+";\\mathbb{Z})$?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) sind ähnlich zur Matrix A aus M("+z+" x "+z+";Z)>>
>> ?\n\n";
}
if (Starttyp==23)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ sind ähnlich zur Matrix $A >>
>> \\in M("+z+" \\times "+z+";\\mathbb{Z})$?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) sind ähnlich zur Matrix A aus M("+z+" x "+z+";Z)>>
>> ?\n\n";
}
if (Starttyp==24)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ sind ähnlich zur Matrix $A >>
>> \\in M("+z+" \\times "+z+";\\mathbb{Z})$?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) sind ähnlich zur Matrix A aus M("+z+" x "+z+";Z)>>
>> ?\n\n";
}
if (Starttyp==31)
{
AGSt = "Geben Sie zu der Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ alle Eigenwerte an.\n\n";
AGS = "Geben Sie zu der Matrix B aus M("+z+" x "+z+";Z) alle Eigenwerte an.\n\n";
}
if (Starttyp==32)
{
AGSt = "Geben Sie zu der Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ das charakteristische Polynom >>
>> an.\n\n";
AGS = "Geben Sie zu der Matrix B aus M("+z+" x "+z+";Z) das charakteristische Polynom an.\n\n";
}
if (Starttyp==41)
{
AGSt = "Geben Sie zu der Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ alle Eigenwerte mit den zugehörigen >>
>> Eigenvektoren $\\vec{x} \\in \\mathbb{C}^"+z+"$ an.\n\n";
AGS = "Geben Sie zu der Matrix B aus M("+z+" x "+z+";Z) alle Eigenwerte mit den zugehörigen Eigenvektoren x aus >>
>> C^"+z+" an.\n\n";
}
if (Starttyp==25)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ haben den oder die folgenden >>
>> Eigenwerte $\\lambda_1, \\lambda_2, ...$ ?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) haben den oder die folgenden Eigenwerte l_1, l_2,>>
>> ... ?\n\n";
}
if (Starttyp==26)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";Z)$ haben das folgende charakteristische >>
>> Polynom?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) haben das folgende charakteristische Polynom?\n\n";
}
if (Starttyp==42)
{
AGSt = "Zeigen Sie, dass die folgende Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ diagonalisierbar ist, >>
>> \nindem Sie eine Matrix $P \\in GL("+z+";\\mathbb{C})$ angeben, sodass $B = P \\circ A \\circ P^{-1}$ gilt,>>
>> \nwobei $A$ Diagonalgestalt besitzt.\n\n";
AGS = "Zeigen Sie, dass die folgende Matrix B aus M("+z+" x "+z+";Z) diagonalisierbar ist,\nindem Sie eine >>
>> Matrix P aus GL("+z+";C) angeben, sodass B=P*A*P^(-1) gilt,\nwobei A Diagonalgestalt besitzt.\n\n";
}
if (Starttyp==27)
{
AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ sind diagonalisierbar?\n\n";
AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) sind diagonalisierbar?\n\n";
}
if (Starttyp==51)
{
AGSt = "Geben Sie eine diagonalisierbare Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ an,\ndie die >>
>> gegebenen Eigenwerte und -vektoren besitzt.\nAchten Sie auf die Reihenfolge.\n\n";
```

```
   AGS = "Geben Sie eine diagonalisierbare Matrix B aus M("+z+" x "+z+";Z) an,\ndie die gegebenen Eigenwerte und >>
   >> -vektoren besitzt.\nAchten Sie auf die Reihenfolge.\n\n";
   }
   if (Starttyp==43)
   {
   AGSt = "Geben Sie zu der folgenden Matrix $B$ die zugehörige Jordansche Normalform $A$\nund eine Matrix $P >>
   >> \\in GL("+z+";\\mathbb{C})$ mit $B = P \\circ A \\circ P^{-1}$ an.\n\n";
   AGS = "Geben Sie zu der folgenden Matrix B die zugehörige Jordansche Normalform A\nund eine Matrix P aus >>
   >> GL("+z+";C) mit B=P*A*P^(-1) an.\n\n";
   }
   if (Starttyp==28)
   {
   AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ haben die folgende Jordansche >>
   >> Normalform $A$?\n\n";
   AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) haben die folgende Jordansche Normalform A?\n\n";
   }
   if (Starttyp==33)
   {
   AGSt = "Geben Sie zu der Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ das Minimalpolynom an.\n\n";
   AGS = "Geben Sie zu der Matrix B aus M("+z+" x "+z+";Z) das Minimalpolynom an.\n\n";
   }
   if (Starttyp==29)
   {
   AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ haben das folgende >>
   >> Minimalpolynom?\n\n";
   AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) haben das folgende Minimalpolynom?\n\n";
   }
   if (Starttyp==34)
   {
   AGSt = "Zeigen Sie, dass die Matrix $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ nilpotent ist. Geben Sie >>
   >> den Nilpotenzgrad $k$ an.\n\n";
   AGS = "Zeigen Sie, dass die Matrix B aus M("+z+" x "+z+";Z) nilpotent ist. Geben Sie den Nilpotenzgrad k >>
   >> an.\n\n";
   }
   if (Starttyp==210)
   {
   AGSt = "Welche der folgenden Matrizen $B \\in M("+z+" \\times "+z+";\\mathbb{Z})$ haben den folgenden >>
   >> Nilpotenzgrad $k$?\n\n";
   AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+z+";Z) haben den folgenden Nilpotenzgrad k?\n\n";
   }
   if (Starttyp==61)
   {
   AGSt = "Lösen Sie das folgende homogene lineare Gleichungssystem $A \\circ \\vec{x} = \\vec{0}$.\n\n";
   AGS = "Lösen Sie das folgende homogene lineare Gleichungssystem A*x=0.\n\n";
   }
   if (Starttyp==62)
   {
   AGSt = "Lösen Sie das folgende (in)homogene lineare Gleichungssystem $A \\circ \\vec{x} = \\vec{b}$.\n\n";
   AGS = "Lösen Sie das folgende (in)homogene lineare Gleichungssystem A*x=b.\n\n";
   }
   if (Starttyp==71)
   {
   AGSt = "Für welche der folgenden Matrizen $B_1, B_2, ... \\in M("+z+" \\times "+s+"; \\mathbb{Z})$ bilden die >>
   >> folgenden Vektoren eine Basis des Kerns?\n\n";
   AGS = "Für welche der folgenden Matrizen B aus M("+z+" x "+s+";Z) bilden die folgenden Vektoren eine Basis des >>
   >> Kerns?\n\n";
   }
   if (Starttyp==72)
   {
   AGSt = "Welche der folgenden Matrizen $B_1, B_2, ... \\in M("+z+" \\times "+s+"; \\mathbb{Z})$ haben den Rang >>
   >> $"+r+"$?\n\n";
   AGS = "Welche der folgenden Matrizen B aus M("+z+" x "+s+";Z) haben den Rang "+r+"?\n\n";
   }
   String H[] = new String[2];
   H[0]=AGS;
   H[1]=AGSt;
   return(H);
  }
}
}
```

Literaturverzeichnis

[Anton, 1998] Anton, Howard (1998): *Lineare Algebra. Einführung, Grundlagen, Übungen.* Berlin; Heidelberg (Spektrum Akademischer Verlag).

[Bessenrodt, 2002] Bessenrodt, Christine (2002): *Skript zur Vorlesung Lineare Algebra I an der Universität Hannover im Wintersemester 2002/2003.* (Unveröffentlicht).

[Beutelspacher, 2001] Beutelspacher, Albrecht (2001): *Lineare Algebra. Eine Einführung in die Wissenschaft der Vektoren, Abbildungen und Matrizen.* 5. Auflage. Braunschweig; Wiesbaden (Vieweg).

[Bosch, 2004] Bosch, Siegfried (2004): *Algebra.* 5., überarbeitete Auflage. Berlin; Heidelberg; New York (Springer-Verlag).

[Bothmer, 2004] Bothmer, Hans-Christian Graf von (2004): *Skript zur Vorlesung Algebra I an der Universität Hannover im Wintersemester 2004/2005.* Onlinedokument: http://ada0.ifam.uni-hannover.de:8100/script.html (Aufruf: 1.8.2006).

[Fischer, 2002] Fischer, Gerd (2002): *Lineare Algebra. Eine Einführung für Studienanfänger.* 13., durchgesehene Auflage. Braunschweig; Wiesbaden (Vieweg).

[Holz & Wille, 2002] Holz, Michael & Wille, Detlef (2002): *Repetitorium der linearen Algebra. Teil 2.* Hannover (Binomi Verlag).

[Kampe, 1993] Kampe, Helmut (1993): „Übung als persönlichkeitsfördernder Prozeß. - Erfahrungen, Beispiele und Möglichkeiten.“ In: *mathematik lehren. Die Zeitschrift für den Unterricht in allen Schulstufen.* Heft 60, Oktober 1993, Seite 18-21.

[Kerncurriculum, 2006] Niedersächsisches Kultusministerium (Hrsg.) (2006): *Kerncurriculum für das Gymnasium. Schuljahrgänge 5-10. Mathematik.* Onlinedokument: http://db2.nibis.de/1db/cuvo/datei/kc_gym_mathe_nib.pdf (Aufruf: 1.8.2006).

[Meyer, 2003] Meyer, Hilbert (2003): *Leitfaden zur Unterrichtsvorbereitung.* 12. Auflage. Berlin (Cornelsen).

[Pohlers, 1999] Pohlers, Wolfram (1999): *Lineare Algebra.* Onlinedokument: wwwmath1.uni-muenster.de/logik/publ/lec2/Pohlers/tskript.pdf (Aufruf: 19.04.2006).

[Reineke, 2000] Reineke, Joachim (2000): *Lineare Algebra.* Onlinedokument: www-ifm.math.uni-hannover.de/∼holz/linalg2/la2_S8.pdf (Aufruf: 31.07.2006).

[Schreyer, 1992] Schreyer, Frank Olaf (1991): *Skript zur Vorlesung Algebra I an der Universität Bayreuth im Wintersemester 1991/1992.* (Unveröffentlicht).

[Wille, 2001] Wille, Detlef (2001): *Repetitorium der linearen Algebra. Teil 1.* 4. Auflage. Hannover (Binomi Verlag).

[Winter, 1993] Winter, Heinrich (1993): „Begriff und Bedeutung des Übens im Mathematikunterricht.“ In: *mathematik lehren. Die Zeitschrift für den Unterricht in allen Schulstufen.* Heft 60, Oktober 1993, Seite 4.

[Zech, 2002] Zech, Friedrich (2002): *Grundkurs Mathematikdidaktik. Theoretische und praktische Anleitungen für das Lehren und Lernen von Mathematik.* 10. Auflage. Basel; Weinheim (Beltz Verlag).

[Zimbardo & Gerrig, 2004] Zimbardo, Philip G. & Gerrig, Richard J. (2004): *Psychologie.* 16., aktualisierte Auflage. München (Pearson Studium).

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der technischen Hilfsmittel

Für die Erstellung der vorliegenden Arbeit sowie der integrierten Abbildungen wurden folgende Programme und Systeme verwendet:

- **Adobe ® Reader ® 7.0:** Version 7.0.0, Copyright ©1984-2004, Adobe Systems Incorporated und Lizenzgeber.
- **J2SE™ Development Kit** Version 5.0, Copyright ©2005 Sun Microsystems, Inc..
- **J2SE™ Runtime Environment** Version 5.0, Copyright ©2005 Sun Microsystems, Inc..
- **joe, java oriented editing:** Version 2.3.25, Copyright ©1998-2002, Timo Haberkern.
- **Maple 9:** Version 9.01, Copyright ©1981-2003, Maplesoft.
- **Microsoft ® Paint:** Version 5.1, Copyright ©1981-2001, Microsoft Corporation.
- **Microsoft ® Office Word 2003:** Version 11.5604.5606, Copyright ©1983-2003, Microsoft Corporation.
- **Microsoft ® Windows XP:** Version 5.1.2600, Copyright ©1985-2001, Microsoft Corporation.
- **OpenOffice.org2.0:** Copyright ©2000-2005 Sun Microsystems, Inc..
- **TEXnicCenter:** Version 1 Beta 6.21, Copyright ©1999-2004, www.ToolsCenter.org.

Die in der Arbeit genannten und verwendeten Produkte unterliegen dem Urheberrecht. Aus der Verwendung der jeweiligen Produktnamen in dieser Arbeit darf nicht auf die freie Verwendbarkeit geschlossen werden.

Index

Erklärung

Ich versichere, dass ich die Hausarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt und die Stellen der Arbeit, die im Wortlaut oder Sinn nach anderen Werken entlehnt sind, unter Angabe der Quelle in jedem einzelnen Fall kenntlich gemacht habe.

Steffen Lünne

Barsinghausen, im August 2006.