

Algorithmische Aufgabenerstellung unter Verwendung von Computeralgebra

Dissertation zur Erlangung des akademischen Grades
Doktorin der Naturwissenschaften (Dr. rer. nat)

von
Hendrikje Schmidtpott-Schulz

Betreut von
Prof. Dr. Wolfram Koepf und Prof. Dr. Andreas Bley

Vorgelegt im Fachbereich
Mathematik und Naturwissenschaften
der Universität Kassel
Februar 2025

Datum der Disputation: 27. August 2025

Inhaltsverzeichnis

1	Einleitung	5
1.1	Einleitung	5
1.2	Technisches	6
1.3	Alternativen zu STACK	6
1.4	Vernetzung, Aufgabensammlungen und didaktische Forschung	8
1.4.1	Aufgabensammlungen	8
1.4.2	Nutzen von Onlineassessments	9
2	Aufgabenerstellung	11
2.1	Grundlagen	11
2.1.1	Matrizentypen	11
2.1.2	Erstellung einer Aufgabe	24
2.1.3	Rechnen mit Restklassen und simplify	35
2.1.4	Graphische Darstellungen	37
2.1.5	Erkennen von komplexen Wurzeln	41
2.1.6	Zufällige, verkettete Funktionen	42
2.2	Neue Aufgabentypen	44
2.2.1	Zufällige Auswertungen	44
2.2.2	Funktionen zu Graphen	47
2.2.3	Testen auf Planarität	49
3	Einsatz der Aufgaben	57
3.1	Einbettung in moodle	57
3.1.1	Einsatz in der Lehre	57
3.1.2	Bestimmung der Schwierigkeit anhand einzelner Aufgaben	61
3.2	Rückmeldung der Studierenden	65
3.2.1	STACKrate	65
3.2.2	Direkte Rückmeldungen	67

4	Fazit und Ausblick	69
4.1	Grenzen der Nutzung	69
4.1.1	Einsatz der Aufgaben	70
4.2	Ausblick	70
A	Codefragmente	79
A.1	Matrizen	79
A.2	Graphen	83
B	Aufgaben	87
B.1	Indexshift	87
B.2	Taylorpolynome	89
B.2.1	Varianten	90
B.3	Bild und Kern (ohne Rechnen)	91
B.3.1	Kern und Bild (3x3, neue Einstellungen, flexibler Rang),	96
C	Dynamische Grafiken mit JSXGraph	101
D	STACKrate	113

Kapitel 1

Einleitung

1.1 Einleitung

Im Rahmen dieser Arbeit sollen Algorithmen entwickelt und erforscht werden, die Aufgaben im Bereich der Linearen Algebra, Analysis und den Diskreten Strukturen generieren können. Diese Aufgaben sollen einen definierten Schwierigkeitsgrad haben. Diese Algorithmen sollen theoretisch beschrieben und in Computeralgebrasystemen (CAS) implementiert werden.

Als wissenschaftliche Mitarbeiterin beim Projekt MAOAM (Mathematische Online Arbeitsmaterialien) habe ich bereits verschiedene Aufgaben für die Lernplattform Moodle unter Verwendung des Computeralgebrasystems MAXIMA erstellt. An diese Vorarbeiten möchte ich anknüpfen und bestehende Algorithmen überarbeiten und verbessern sowie neue, effiziente Algorithmen entwerfen. MAXIMA wird hierzu für die Erstellung der Algorithmen verwendet, umgesetzt werden die fertigen Aufgaben mit dem moodle-Plugin STACK [San16].

Wesentlicher Teil dieser Aufgaben ist das Feedback an die Studentinnen und Studenten. Um Studierenden wirklich helfen zu können, sind differenzierte Rückmeldungen unerlässlich. Das bedeutet, neben der Fragestellung müssen die potentiellen Antworten der Studierenden analysiert werden. Das Plugin STACK in Moodle liefert hier gute Werkzeuge, um die Antworten der Studierenden entlang binärer Bäume auf mathematische Eigenschaften hin zu untersuchen und entsprechende Rückmeldung zu geben. Diese Möglichkeit muss bereits bei der Erstellung der Frage berücksichtigt werden [Kal16a] und [Kal16b].

Nicht zuletzt muss auf die technischen Möglichkeiten eingegangen werden, die von Studierenden parallel zur Bearbeitung einer Onlineaufgabe potentiell genutzt werden können. Tools wie wolframalpha.com oder Photomath liefern auf Knopfdruck rechnerische Lösungen mit Zwischenschritten. Das Üben der rechnerischen Fähigkeiten rückt somit in den Hintergrund und wir müssen oder dürfen uns auf die Interpretation der Ergebnisse konzentrieren.

In dieser Arbeit möchte ich einige von mir entwickelte Algorithmen vorstellen und anhand einer Aufgabe aus der Linearen Algebra auch das Vorgehen und die Überlegungen beim Erstellen einer Aufgabe erläutern, die sich aus meiner Sicht als zielführend herausgestellt haben. Zudem erläutere ich den Einsatz der von mir entwickelten Aufgaben in der Lehre und ihre Einbettung in moodle. Hierbei werde ich auf Schwierigkeiten eingehen, auf die wir im Zuge der Randomisierung gestoßen sind.

1.2 Technisches

An der Universität Kassel wird STACK, das als moodle-Plugin vom Rechenzentrum verwaltet wird, verwendet, bei der Erstellung von Aufgaben sind wir an die jeweils aktuell installierte Version gebunden. Zum jetzigen Zeitpunkt läuft STACK in der Version 4.7.0 vom Juli 2024.

Die Version von moodle, die an der Universität Kassel genutzt wird, ist Version 4.1.12. Entsprechend sind alle Bilder, die aus moodle entnommen wurden, in diesen oder Vorgängerversionen erstellt.

1.3 Alternativen zu STACK

STACK ist natürlich nicht das einzige Tool zur Erstellung von mathematischen Aufgaben, die in Learning-Management-Systeme eingebaut werden können. An dieser Stelle möchte ich kurz eine Auswahl anderer Systeme vorstellen.

Numbas

Aus der Universität Newcastle stammt das Numbas-System. Mit Numbas lassen sich Tests erstellen, die auf den ersten Blick Ähnlichkeit mit STACK-Tests haben. Numbas-Tests können als Stand-Alone-Tests zum Üben verwendet werden. Um sie z.B. in moodle einzubinden, müssen sie als SCORM-Module eingepflegt werden. Hinter Numbas steckt kein CAS, allerdings ist die

Nutzung verschiedenster JavaScript-Bibliotheken und anderer Erweiterungen möglich, so lassen sich in Numbas-Aufgaben auch durchaus JSXGraph oder Geogebra integrieren. Randomisieren von Variablen ist möglich. Ein schönes Feature ist die Handschrifterkennung. Das System wird laufend erweitert und bietet auch direkt eine öffentliche Bibliothek für Aufgaben.

Wiris

Bei Wiris handelt es sich um eine kommerzielle Software, die z.B. als Plugin für moodle dient. Ähnlich wie bei Numbas ist eine handschriftliche Eingabe möglich. Inwiefern ein Computeralgebrasystemen im Hintergrund läuft, ist mir leider nicht bekannt.

Geogebra

Geogebra ist eine Geometriesoftware mit Schnittstelle zu algebraischen Elementen. Sie lässt sich sowohl für Demonstrationszwecke als auch zur Abfrage von Lösungen und Lösungswegen verwenden. Geogebra lässt sich als Software installieren oder im Browser nutzen. Hier stellt Geogebra auch eine Plattform, um eigene Aufgaben in Form von Applets zu hosten und freigegebene Applets anderer User zu finden. Auf diese Weise unterscheidet sich Geogebra grundlegend von den anderen Lösungen, da es sich hierbei um eine eigenständige Software handelt, die der Allgemeinheit direkt zur Verfügung steht. Darüber hinaus lässt sich Geogebra auch direkt in moodle integrieren: man kann es als eigenes Aufgabenformat in Tests verwenden oder aber auch in andere Aufgabentypen wie STACK oder Numbas einbinden, indem auf gehostete Applets zugegriffen wird. Erstellt man z.B. Aufgaben in Geogebra für moodle, können geometrische Konstruktionen abgefragt werden, oder man kann eine grafische Lösung einfordern. Den Applets, die man selbst oder andere mit Geogebra erstellt haben und über die Plattform aufgerufen werden, lassen sich bei Aufruf Eingabevariablen übergeben, und so können sie an randomisierte Aufgaben direkt angepasst werden. Nutzen die Studierenden das Applet, so kann der Status des Applets nach Bearbeitung zurück an das aufrufende System übergeben und Bepunktung sowie Feedback zu Aufgaben vorgenommen werden, sei es durch moodle direkt oder durch eine STACK-Aufgabe, in die das Applet eingebettet ist.

1.4 Vernetzung, Aufgabensammlungen und didaktische Forschung

1.4.1 Aufgabensammlungen

Die Aktualität des Themas und die Vielzahl technischer Alternativen legen nahe, dass ähnlich gelagerte Projekte bereits existieren. An dieser Stelle soll ein kurzer Überblick über einige relevante Initiativen gegeben werden.

Im deutschsprachigen Raum ist insbesondere der Arbeitskreis Mathe-Digital [Dig21] zu nennen. In dessen Rahmen entstand die Datenbank DOMAIN, in der STACK-Aufgaben gesammelt und heruntergeladen werden können, um sie in eigene Online-Tests zu integrieren. Auch einige der in dieser Arbeit vorgestellten Aufgaben stellen wir in dieser Datenbank zu Verfügung.

International ist die Vernetzung innerhalb der STACK-Community sehr stark. Regelmäßiger Austausch finden im Rahmen von STACK-Konferenzen [Con] und des STACK Professional Network [Net] statt, bei denen technische Neuentwicklungen sowie neue Einsatzmöglichkeiten von STACK diskutiert werden.

Eine umfangreiche Sammlung von Aufgaben wird direkt mit STACK verbreitet. Das Moodle-Plugin liefert zudem einen Moodle-Kurs mit Beispielaufgaben, der den Einstieg in die Erstellung eigener Aufgaben erleichtern kann.

Zusätzlich lassen sich zahlreiche Aufgaben aus den HELM-Workbooks für STACK nutzen. HELM (Helping Engineers Learn Mathematics) war ein 2002 für vier Jahre angelegtes Projekt britischer Universitäten, in dessen Rahmen insgesamt 50 Arbeitsbücher für Studierende der Ingenieurwissenschaften entwickelt wurden [HPW09]. Viele der Übungsaufgaben wurden in den letzten Jahren in STACK migriert und online bereitgestellt. Sie sind nun über die STACK-Webseite zugänglich [SD25] und [SCMMBZ25].

Nicht direkt öffentlich zugänglich ist die Datenbank der finnischen Hochschulen, Abacus [Aba]. Neben den ursprünglichen Projektpartnern haben sich inzwischen zahlreiche weitere Hochschulen an der Sammlung beteiligt und eigene Materialien beigesteuert [Ras16].

Hervorzuheben ist zudem IDEMS international, die mit INNODEMS, Onlinekurse und Materialien mit und für STACK entwickelt, um die mathematische Ausbildung, insbesondere an afrikanischen Universitäten, voranzutreiben [ZB23].

Wir sehen, dass in den letzten Jahren eine Fülle an Onlinematerialien für die mathematische Bildung Studierender entstanden ist, ein Großteil hiervon in STACK. Einiges ist randomisiert, häufig aber auf einem eher niedrigen Level. Entsprechend soll in den folgenden Abschnitten der Fokus eher auf

innovativen und neuartigen Beispielen liegen und weniger auf Routineaufgaben, die sich besonders effektiv mit Onlineassessments bearbeiten lassen.

1.4.2 Nutzen von Onlineassessments

Befragt man Studierende nach ihren Wünschen zur Prüfungsvorbereitung, so werden neben der Verfügbarkeit von Altklausuren häufig der Wunsch nach weiteren freiwilligen Übungsmöglichkeiten geäußert. Hier kann automatisiertes und randomisiertes Onlineassessment eine bedeutende Rolle spielen, da direktes Feedback ohne zusätzlichen Korrekturaufwand möglich ist.

Zum generellen Einsatz von Onlineassessments gerade in der Eingangsphase des Studiums existieren Studien, die einen Nutzen für einige Personengruppen nahelegen. Hier verweise ich auf die Untersuchungen zu mathematischen Vorkursen an der Universität, bei denen Blended-Learning-Methoden diskutiert wurden [Fis14]. Reine Online-Angebote erscheinen für bestimmte Zielgruppen jedoch problematisch [WCH15]. Da es in dieser Arbeit jedoch um einzelne Online-Aufgaben und -Tests geht, beschränken wir uns im Folgenden auf die Nutzungspotenziale von STACK.

Onlineassessments, die STACK nutzen, werden bereits an einigen Universitäten eingesetzt. Häufig erfolgt der Einsatz eher pragmatisch. An der Maseno Universität in Kenia wird STACK in größerem Stil eingesetzt und in diesem Zusammenhang wurden Ergebnisse von Prüfungen mit der Nutzung von Onlinematerialien verglichen. In [OPSS21] wird ein Setup beschrieben, bei dem verschiedene Onlinetests zur Gesamtnote eines Kurses beitragen, und die Ergebnisse dieser Vortests mit den Prüfungsergebnissen verglichen. Ein Zusammenhang zwischen kontinuierlichen erfolgreich bearbeiteten STACK-Aufgaben und guten Klausurergebnissen ließ sich herstellen. In einer späteren Analyse [JAOO22] wurde auf Studierendengruppen eingegangen, bei denen Vortests und Prüfung nicht den Erwartungen entsprachen. Sie stießen hier neben Problemen der technischen Infrastruktur auch auf Studierende, die andere dafür bezahlten, ihre Test zu bearbeiten. Einzelne bemängelten das Feedback in einigen Aufgaben, so dass sie aus Fehlern nichts lernten. Im Allgemeinen fiel das Feedback dieser Analyse aber sehr positiv aus.

Eine weitere Fallstudie wurde an der Universität Triest durchgeführt [DJMWDD24] und [JCDL24]. In der Linearen Algebra wurden Klausurergebnisse bei unterschiedlichem Einsatz von STACK-Aufgaben in verschiedenen Gruppen betrachtet. In diesen Datengruppen gab es eine Korrelation zwischen Einsatz und Ergebnis, die aber nicht zwangsläufig kausal sein muss. Auch hier war das Feedback der Studierenden meist positiv, wobei teilweise

Sorge bestand, dass Lösungswege nicht so gut bewertet werden können wie bei Papierabgaben.

STACK lässt sich als Moodle-Fragetyp sehr unterschiedlich einsetzen. Soll es zum einfachen Üben verwendet werden, lassen sich Tests beliebig oft durchführen.

An der Universität Ljubljana wurde analysiert, wie freiwilliges Wiederholen von Tests mit erreichten Punkten in verpflichtenden Prüfungen korreliert [Pre22]. Grundlage hierzu sind Daten aus einer Mathematik 1 Veranstaltung. Hier müssen die Studierenden Vortests in STACK bestehen um zu Klausuren zugelassen zu werden. Eine Begrenzung der Anzahl an Versuchen existiert nicht. Hierbei fällt auf, dass bei schwächeren Studierenden sehr häufiges Wiederholen nicht zu Erkenntnisgewinnen geführt hat.

Auch in [THN25] wird ebenfalls ein Szenario mit beliebig vielen Wiederholungen beschrieben. Der Fokus hier liegt allerdings auf Feedbackbäumen. Es wird untersucht, wie Studierende sich durch unterschiedliche Wege dem korrekten Ergebnis annähern. Diese Vorarbeiten münden in einem neuen Ansatz Feedbackbäume zur Erarbeitung korrekter Lösungen zu nutzen, so dass das Feedback auf die Übereinstimmung mit der Musterlösung einer Aufgabe hinwies und Studierende sich so der korrekten Lösung annähern konnten [NFE⁺25].

Kapitel 2

Aufgabenerstellung und verschiedene Typen

2.1 Grundlagen

2.1.1 Matrizentypen

Matrizen spielen in den unterschiedlichsten Gebieten der Mathematik, insbesondere auch bei mathematischen Veranstaltungen in der Ingenieurmathematik, eine wesentliche Rolle und tauchen in den unterschiedlichsten Aufgaben auf. Im Folgenden möchte ich auf die Erstellung von verschiedenen Matrixtypen eingehen. Da die erstellten Matrizen zur Generierung von Übungsaufgaben verwendet werden sollen, liegt ein besonderer Fokus auf ganzzahligen Matrizen. Einiger dieser Überlegungen finden sich auch in [Ste03].

Eine absolut beliebige Matrix ist benötigt lediglich eine Funktion, die jedem Matrixeintrag $a[i, j]$, einen randomisierten Wert zuweist und einen Befehl, der hieraus ein Matrixobjekt in der gewünschten Dimension erstellt. Wir verwenden diese Art Matrizen nur bei Aufgaben zu elementaren Matrixoperationen. Eine ganzzahlige $n \times m$ -Matrix mit Einträgen zwischen $-k$ und k wird mit

```
a[i, j] := rand_with_step(-k, k, 1);  
A: genmatrix(a, n, m)
```

erstellt.

Matrizen mit vielen Nulleinträgen

Bei bestimmten Aufgaben ist es sinnvoll eine Mindestanzahl an Nulleinträgen pro Zeile oder Spalte festzulegen. Dies lässt sich realisieren, indem man eine Matrix aus Zeilen mit einer bestimmten Anzahl an Nulleinträgen generiert.

Um eine $n \times m$ -Matrix mit genau oder mindestens k Nulleinträgen zu generieren, wird eine Liste aus k Nullen und $m - k$ Einsen generiert, n Permutationen hiervon ergeben dann die Matrix. Nun können alle Einseinträge durch eine zufällige Zahl ersetzt werden. Sinnvollerweise überprüft man, ob hierbei Nullspalten entstehen. Beispielcode findet sich im Anhang A.1.

Algorithmus 2.1:

Erstellung einer quadratischen $n \times n$ Matrix mit k Nulleinträgen pro Zeile

Eingabe: n , k , `max`, mit `max` als Betragsmaximum der Einträge

Ausgabe: gesuchte Matrix

1. überprüfe, ob Eingabe sinnvoll, sonst Ausgabe: Nullmatrix
2. erstelle Liste `line` mit k Nulleinträgen und $n - k$ Einseinträgen
3. erstelle n Permutationen der Liste
4. ersetze die 1-Einträge der Listen durch zufällige ganze Zahlen zwischen `-max` und `max` ohne Null
5. erstelle mögliche Ausgabematrix
6. überprüfe Matrix auf gewünschte Eigenschaften (Nullspalten, Rang, etc.), dann Ausgabe, sonst zu (2.)

Effizienz: Abhängig von den gewünschten Eigenschaften, siehe Tabelle 2.1

Zur Einordnung der Effizienz des Algorithmus 2.1 betrachten wir die Erstellung von 500, nicht notwendigerweise verschiedenen, 10×10 -Matrizen mit k Nulleinträgen in wxMaxima. Hierbei ermitteln wir die durchschnittliche Anzahl an Durchläufen, bis eine valide Matrix erstellt ist.

Die Effizienz dieses Algorithmus hängt, wie wir erkennen sehr stark von den gewünschten Eigenschaften ab. Betrachten wir die einzelnen Schritte: Zunächst vergleichen wir lediglich die Eingabegrößen und entscheiden über die Lösbarkeit. Das einmalige Erstellen einer Liste aus Null- und Einseinträgen lässt sich ebenfalls vernachlässigen. Interessant ist die Erstellung der n Permutationen. Verwendet wird hier der Befehl `random_permutation(line)`

	max	$k = 0$	$k = 2$	$k = 4$	$k = 6$	$k = 8$
keine Nullspalte	2	1	1	1.018	1.312	13.634
voller Rang	2	1.004	1.008	1.056	1.682	68.166
keine Nullspalte	5	1	1	1.002	1.128	5.718
voller Rang	5	1	1	1.006	1.136	13.268

Tabelle 2.1: durchschnittliche Anzahl an nötigen Neuerstellungen einer 10×10 -Matrix nach Algorithmus 2.1

auf eine Liste `line`. Maxima gibt hier Knuths Algorithmus als eine Version von Fisher-Yates an, deren Aufwand linear mit n steigt ([FY48] und [Knu97]).

Reguläre Matrizen

Wesentlich sind reguläre Matrizen.

Definition 2.1:

Eine quadratische Matrix A über einem Körper $\mathbb{K}$ heißt regulär, wenn sie in $\mathbb{K}$ invertierbar ist.

Bemerkung 2.1:

Die regulären $n \times n$ -Matrizen bilden einen Ring über $\mathbb{K}$ und wir bezeichnen die Menge dieser Matrizen als allgemeine lineare Gruppe $GL(n, \mathbb{K})$.

Bekanntlich ist eine Matrix genau dann regulär, wenn ihre Determinante von Null verschieden ist. Entsprechend kann man eine reguläre Matrix finden, indem man Ihre Komponenten absolut zufällig besetzt, und anschließend überprüft, ob die Determinante von Null verschieden ist. Sofern keine weiteren Eigenschaften erforderlich sind, ist dies ein praktikables Vorgehen (vgl. Tabelle 2.2 für 5000 Durchläufe). Wir sehen anhand dieser Tabelle, dass die Wahrscheinlichkeit direkt eine reguläre Matrix zu erhalten mit Größe der Komponenten und der Matrix zunimmt. Stehen nur die Komponenten 1, 0 und -1 zur Verfügung, so kommt es noch zu Ausreißern.

Um hier auch eine untere Schranke anzugeben, was die Wahrscheinlichkeit betrifft, eine reguläre Matrix zu erhalten, versuchen wir einige reguläre Matrizen zu zählen und gehen hierbei Zeile für Zeile vor. Der erste Zeilenvektor

$n \backslash \text{max}$	1	2	3	5	7
2	2.6	1.34	1.17	1.07	1.04
3	2.9	1.26	1.10	1.03	1.01
5	2.7	1.09	1.02	1.00	1.00
8	1.8	1.01	1.00	1.00	1.00

Tabelle 2.2: durchschnittliche Anzahl der nötigen Neuerstellungen einer zufällig bestimmten regulären $n \times n$ -Matrix mit Komponenten zwischen $-\text{max}$ und max

v_1 der Matrix ist, sofern er nicht der Nullvektor ist, beliebig wählbar, es gibt demnach $(2 \cdot k + 1)^n - 1$ mögliche Zeilenvektoren der Länge n mit Einträgen zwischen $-k$ und k . Auch die Anzahl der zum ersten Zeilenvektor linear unabhängigen Vektoren für die zweite Zeile lässt sich bestimmen. Hierzu betrachten wir zu welchen Vektoren welche Vielfache existieren können, ohne dass wir Einträge erhalten, die betragsmäßig größer als k sind. Wir zerlegen hierzu die Menge zulässiger Vektoren in disjunkte Teilmengen $\mathcal{T}_1$ bis $\mathcal{T}_k$. Dabei ist $\mathcal{T}_1$ die Menge der Zeilenvektoren, die ausschließlich 0, 1 und -1 enthalten, $\mathcal{T}_i$ die Menge der Vektoren mit Einträgen aus $\{-i, \dots, i\}$ allerdings ohne Vektoren, die bereits in $\cup_{l=1}^{i-1} \mathcal{T}_l$ enthalten sind. Für Vektoren aus $\mathcal{T}_1$ gilt, dass sie mit jedem $i \in \{-k, \dots, k\}$ multipliziert werden können und immer noch passende Komponenten beinhalten, für Vektoren aus $\mathcal{T}_i$ sind Zahlen möglich, die betragsmäßig kleiner oder gleich $\lfloor \frac{k}{i} \rfloor$ sind. Wir können somit zu jedem v_1 bestimmen, wie viele linear abhängige Vektoren es gibt und damit auch berechnen, mit welcher Wahrscheinlichkeit die ersten beiden Vektoren linear unabhängig sind. Für jede Menge $\mathcal{T}_i$ gilt somit $|\mathcal{T}_i| = (2 \cdot i + 1)^n - (2 \cdot i - 1)^n$. Wir definieren $|r_{1,2}|$ als die Anzahl an linear unabhängiger erster zwei Zeilen und erhalten wir somit

$$|r_{1,2}| = \sum_{i=1}^k ((2k+1)^n - (2 \cdot \lfloor \frac{k}{i} \rfloor + 1)^n) \cdot ((2 \cdot i + 1)^n - (2 \cdot i - 1)^n)$$

mögliche von insgesamt $(2 \cdot k + 1)^{2n}$ möglichen Zeilenkombinationen.

Definition 2.1:

Als lineare Hülle oder auch Spann einer Menge Vektoren $a_1, \dots, a_i$ in einem $\mathbb{K}$ -Vektorraum V bezeichnen wir die Menge aller Linearkombinationen dieser Vektoren, also

$$\langle A \rangle := \text{span}\{a_1, \dots, a_i\} = \left\{ \sum_{k=1}^i \lambda_k a_k \mid \lambda_k \in K\mathbb{K} \right\}$$

Da wir für weitere Zeilen sämtliche Linearkombinationen der bisherigen Zeilenvektoren untersuchen müssten, müssen wir uns ab der dritten Zeile mit einer groben Abschätzung begnügen. Nehmen wir an, die bisherigen $i - 1$ Zeilen sind linear unabhängig und spannen einen $i - 1$ -dimensionalen Untervektorraum $\langle A_{i-1} \rangle = \text{span}\{v_1, \dots, v_{i-1}\}$ auf. Wir können für unsere neue Zeile immer einen Zeilenvektor v finden, der linear abhängig von den bisherigen Zeilenvektoren ist, also in dem erwähnten Untervektorraum enthalten ist. Nun finden sich aber auch immer Einheitsvektoren die nicht in $\langle A_{i-1} \rangle$ enthalten sind. Ohne Beschränkung der Allgemeinheit seien dies die Einheitsvektoren E_i bis E_n . Wenn wir nun einen dieser Einheitsvektoren $E_l, l \in \{i, i+1, \dots, n\}$ oder ein Vielfaches von E_l an v addieren, so kann dieser neue Vektor nicht in $\langle A_{i-1} \rangle$ liegen, muss also linear unabhängig sein. Ändern wir also diese l -te Komponente von v , erhalten wir einen neuen Zeilenvektor. Da alle anderen Werte zwischen $-k$ und k möglich sind, erhalten wir mindestens $2 \cdot k$ linear unabhängige Zeilenvektoren v_i für die neue Zeile. Dies lässt sich bis zur n -ten Zeile fortführen. Für jede neue Zeile erhalten wir somit mindestens $2k(n - i + 1)$ Möglichkeiten eine linear unabhängige Zeile hinzuzufügen. Die Wahrscheinlichkeit einen linear unabhängigen neuen Zeilenvektor für die Zeile i zu finden beträgt also mindestens $\frac{2k(n-i+1)}{(2k+1)^n}$, wenn unser System bislang linear unabhängig war. Nun ist aber auch jede Linearkombination dieser Einheitsvektoren linear unabhängig (sofern nicht jeder dieser Linearfaktoren gerade Null ist) und somit können wir diese Abschätzung verbessern, in dem wir nun stattdessen $(2k + 1)^{n-i+1} - 1$ linear unabhängige Kandidaten für unseren Zeilenvektor v_i annehmen und erhalten jetzt eine Wahrscheinlichkeit von

$$P(\dim(\langle A_i \rangle) = i \mid \dim(\langle A_{i-1} \rangle) = i - 1) \geq \frac{(2k + 1)^{n-i+1} - 1}{(2k + 1)^n}$$

dass die neue Zeile linear unabhängig von den vorherigen ist, sofern diese schon linear unabhängig waren. Um eine geschlossene Form zu erhalten er-

$n \backslash \max$	1	2	3	5	7
2	1.23	2.78	5.61	14.14	27.59
3	1.68	3.76	7.65	20.07	38.71
5	2.54	5.92	11.78	30.83	59.33
8	3.75	8.72	17.40	45.55	86.33

Tabelle 2.3: durchschnittlicher Wert der betragsgrößten Komponente bei einer durch LR -Zerlegung entstandenen Matrix

stellen nun auch v_2 auf die eben beschriebene Weise und zusammengefasst erhalten wir also

$$\begin{aligned}
P(\dim(< A_n >) = n) \\
&\geq P(v_1 \neq 0) \cdot \prod_{i=2}^n P(\dim(< A_i >) = i \mid \dim(< A_{i-1} >) = i-1) \\
&= \prod_{i=1}^n \frac{(2k+1)^{n-i+1} - 1}{(2k+1)^n}
\end{aligned}$$

Eine weitere Möglichkeit, eine reguläre Matrix garantiert zu finden, ist die LR -Zerlegung. Hierbei erhält man direkt eine reguläre Matrix, allerdings lassen sich auf diese Weise die Komponenten nicht direkt begrenzen. Theoretisch ist die maximale Obergrenze der Komponenten einer so erstellten $n \times n$ Matrix mit Einträgen zwischen $-\max$ und $\max$ bei $\max^2 \cdot n$, je größer $\max$ gewählt wird, desto weiter entfernt liegt im Schnitt der betragsgrößte Wert (vgl. Tabelle 2.3).

Ganzzahlig invertierbare Matrizen

Für viele Anwendungen verwenden wir zusätzlich die Inverse einer Matrix, beispielsweise zum Lösen von Gleichungssystemen oder bei Basiswechselmatrizen. So ist es wünschenswert, dass die Matrix, auf der eine Aufgabe beruht, nicht nur regulär sondern auch ganzzahlig invertierbar ist. Die vorgestellten Verfahren für reguläre Matrizen lassen sich auf ganzzahlig invertierbare Matrizen übertragen. Aus der Cramerschen Regel können wir folgern, dass eine Matrix ist genau dann ganzzahlig invertierbar, wenn ihre Determinante 1 oder -1 ist. Entsprechend können wir, analog zu der ersten Methode für reguläre Matrizen, nach der Erstellung der Matrix aus zufälligen Einträgen auf

$n \backslash \text{max}$	2	3	5
2	3.0	8.1	21.8
3	6.9	19	86
5	31	203	2700

Tabelle 2.4: durchschnittliche Anzahl der nötigen Neuerstellungen einer zufällig bestimmten ganzzahlig invertierbaren $n \times n$ -Matrix mit Komponenten zwischen $-\text{max}$ und max , die Varianz liegt hierbei in der Größenordnung des Mittelwerts

deren Determinante prüfen, gegebenenfalls eine neue Variante erstellen und dies so lange wiederholen, bis die Matrix diesem Kriterium entspricht. Bei größeren Matrizen ist dies allerdings zeitaufwendig (vgl. Tabelle 2.4).

Satz 2.1:

Cramersche Regel Sei $A \in \mathbb{K}^{n \times n}$ eine reguläre Matrix und $b \in \mathbb{K}$ ein Vektor über $\mathbb{K}$. Dann ist das durch $A \cdot x = b$ gegebene Gleichungssystem eindeutig lösbar und die Komponente x_i des Lösungsvektors $x \in \mathbb{K}^n$ ist gegeben durch $x_i = \frac{\det A_i}{\det A}$, wobei A_i entsteht, indem die i -te Spalte von A durch den Vektor b ersetzt wird.

Die zweite Möglichkeit, eine Erstellung der ganzzahlig invertierbaren Matrix A über die LR -Zerlegung, ist an dieser Stelle deutlich effizienter. Dies erhalten wir einfach dadurch, dass wir nur die Werte 1 und -1 auf der Diagonalen der Dreiecksmatrizen zulassen. Auch hier können wir nicht direkt den betragsgrößten Eintrag in der Matrix angeben. Außerdem korreliert die Betragsgröße der Komponenten offensichtlich auch mit der Position dieser Komponenten innerhalb der Matrix, so ist $a_{n,n}$ immer 1 oder -1. Dies kann man mit Hilfe einer Permutationsmatrix ändern, die bei Multiplikation mit A Zeilen oder Spalten von A vertauscht. Der Code dazu findet sich im Anhang bei A.2.

Algorithmus 2.2:

Erstellung einer ganzzahlig invertierbaren $n \times n$ -Matrix

Eingabe: n , $\max$ mit $\max$ als Betragsmaximum der Einträge der Dreiecksmatrizen L und R

Ausgabe: gesuchte Matrix

1. definiere lokale Variable i, j, L, R
2. beginne schleife $i : 1 \dots n$
3. beginne Schleife $j : 1 \dots n$
4. falls $i=j$ setze $L[i, j]$ und $R[i, j]$ auf 1 oder -1
5. falls $i < j$ setze $R[i, j]$ auf zufällige Zahl aus $[-\max, \max]$ und $L[i, j]$ auf 0
6. falls $i < j$ setze $L[i, j]$ auf zufällige Zahl aus $[-\max, \max]$ und $R[i, j]$ auf 0
7. beende Schleife (3)
8. beende Schleife (2)
9. generiere Matrizen aus L und R und gebe das Produkt beider aus.

Effizienz: Ganzzahlig invertierbare Matrix beim ersten Durchlauf

Matrizen mit Rang r

Benötigen wir einen bestimmten Rang r einer Matrix, so können wir Zufallsmatrizen solange erstellen, bis der Rang r erreicht ist, oder wir kontrollieren das Resultat, indem wir eine reguläre $n \times n$ Matrix A mit obigen Verfahren bestimmen und eine eher einfache Matrix X aufstellen, die $n - r$ linear abhängige Zeilen oder Spalten besitzt. Mit Hilfe von zufälligen Permutationen erhält man mit $M = A \cdot X$ oder $M = X \cdot A$ sehr unterschiedliche Matrizen vom Rang r .

Algorithmus 2.3:

Erstellung einer $n \times n$ -Matrix vom Rang r

Eingabe: n, r

Ausgabe: gesuchte Matrix

1. erstelle Liste `liste` aus den ersten Einheitsvektoren
2. wähle r zufällige Elemente von `liste` (in zufälliger Reihenfolge) aus redefiniere `liste` durch diese Auswahl
3. beginne Schleife `i:r+1..n`
4. addiere zwei zufällige Elemente von `liste` und füge sie am Ende von `liste` hinzu
5. definiere Matrix `A` aus `liste`

Anmerkungen: 1. Variationen der Linearkombinationen sind natürlich auch möglich, genauso wie Einschränkungen derart, dass nicht zweimal die gleichen Spalten ausgewählt werden dürfen. Die Umsetzung in Maxima findet sich im Anhang (A.1).

2. Nach Schritt 4 liegen alle Einträge geordnet in Listenform vor, um Matrixoperationen durchführen zu können, fehlt der `genmatrix(liste,n,n)`-Befehl.

Analog lässt sich auf diese Weise natürlich auch eine $n \times m$ -Matrix erstellen, indem man die Matrix X als $n \times m$ - oder $m \times n$ -Matrix aufstellt.

Orthogonale Matrizen

Definition 2.3:

Wir nennen eine reelle Matrix A orthogonal, wenn für ihre Spaltenvektoren a_i, a_j gilt: $i \neq j \Rightarrow a_i^T \cdot a_j = 0$

Definition 2.3:

Wir nennen eine reelle Matrix A orthonormal, wenn für ihre Spaltenvektoren a_i, a_j gilt: $a_i^T \cdot a_j = \begin{cases} 1 & \text{falls } i = j, \\ 0 & \text{sonst.} \end{cases}$.

Wollen wir stattdessen orthogonale Matrizen erzeugen, so nutzen wir das Gram-Schmidt-Verfahren mit einem zufälligen ersten Vektor und den kartesischen Einheitsvektoren. Sollten diese Vektoren nun normiert werden, so erhalten wir gewöhnlich Vektoren mit irrationalen Einträgen. Wollen wir dies vermeiden, können wir die Matrix mit einem skalaren Vielfachen multiplizieren. Ein Codefragment findet sich im Anhang A.6. Andernfalls kann auf die in [MJO24] vorgestellten Matrizen zurück gegriffen werden. Hier werden eine Liste von orthonormalen Matrizen verlinkt und unterschiedliche Verfahren vorgestellt, wie solche Matrizen generiert werden können.

Bestimmte Eigenwerte

Sind die Eigenwerte λ_i der Matrix M vorgegeben, so reicht es eine reguläre Matrix A zu generieren und M mit $M = A^{-1} \cdot D \cdot A$ zu berechnen, wobei D die Diagonalmatrix ist, die aus den vorgegebenen Eigenwerten besteht. Ist $\det(A) = \pm 1$, so ist M sogar ganzzahlig. Mit A sind auch direkt Eigenvektoren zu den gewünschten Eigenwerten bestimmt.

Symmetrische Matrizen

Benötigen wir eine symmetrische Matrix, so können wir einfach eine solche Matrix generieren, indem wir alle Einträge auf und oberhalb der Diagonale zufällig bestimmen und die Einträge unterhalb der Diagonale anpassen.

Wenn wir aber für unsere symmetrische Matrix A schon Eigenwerte vorgeben möchten, da wir z.B. Aussagen zu Definitheit machen wollen, gibt es grundsätzlich wieder zwei Möglichkeiten. Die erste wäre, dass wir eine symmetrische Matrix erstellen und deren Eigenwerte überprüfen, dies wiederholen wir so lange, bis passende Eigenwerte gefunden werden. Da wir tendenziell kleine Dimensionen verwenden, da ab $\dim(n) = 5$ das Lösen des charakteristischen Polynoms ohne Raten einzelner Eigenwerte algebraisch nicht möglich sein könnte, ist dieses Verfahren durchaus vertretbar. Es muss bei der Implementierung allerdings zwingend eine Abbruchbedingung angelegt werden, da selbst bei 3×3 -Matrizen mit Koeffizienten zwischen -5

und 5 gelegentlich mehr als 100 Durchläufe für rationale Eigenwerte benötigt werden. Fragetests sollten in moodle also unbedingt durchgeführt werden, so dass wir nur brauchbare Varianten für unsere Aufgaben verwenden. Möchten wir stattdessen 4×4 -Matrizen einsetzen, sind häufig 500 Durchläufe nicht ausreichend. Entsprechend lohnt es sich hier, brauchbare Matrizen in einer externen Datei zu speichern aus denen in der Aufgabenerstellung eine zufällig ausgewählt wird.

Zum anderen können wir, ähnlich wie bei allgemeinen Matrizen mit vorgegebenen Eigenwerten, auch bei symmetrische Matrizen vorgehen. Wieder erstellen wir A aus der Diagonalmatrix D , welche die Eigenwerte enthält. Optimal wäre es, könnten wir nun eine orthonormale $n \times n$ -Matrix T bestimmen und $A = T^T \cdot D \cdot T$ berechnen. Das Resultat wird eine symmetrische Matrix sein. Orthonormale Matrizen T , für die $T^T \cdot T = E_n$ gilt, sind in den seltensten Fällen ganzzahlig. Dies wäre beispielsweise bei Permutations- oder Dreh- und Spiegelungsmatrizen der Fall, und diese sind eher uninteressant für die Aufgabenerstellung. Auch mit Gram-Schmidt lassen sich orthonormale Matrizen konstruieren, die wegen der Normierung der Vektoren allerdings nicht ganzzahlig sein können. Damit führt das Erstellen unserer symmetrischen Matrix A auf diese Weise nur in Ausnahmefällen bei besonders geschickt gesetzten Diagonalelementen der Matrix D zu einer ganzzahligen oder wenigstens rationalen symmetrischen Matrix.

Wir diskutieren nun die Implementierung zweier unterschiedlicher Verfahren um geeignete Matrizen zu finden. Zunächst erzeugen wir die Matrix T mit dem Gram-Schmidt-Verfahren. Für eine 3×3 -Matrix ist dies vertretbar und wir können aus einer dünn besetzten Matrix die orthonormale Matrix T generieren so dass wir eine zumutbare symmetrische Matrix A erhalten. Bereits 4×4 -Matrizen müssen aber in moodle händisch ausgewählt werden, um den Rechenaufwand für die Studierenden auf ein vernünftiges Maß zu begrenzen. Versehentliche Verwendung trivialer Matrizen sollte ebenfalls entweder durch die Implementierung oder durch Auswahl geeigneter Vertreter in moodle vermieden werden.

Daneben gibt es aber Matrizen, die nicht ganz optimal sind, für unsere Zwecke aber ausreichen. Ein Ansatz hierzu wird in [MJO24] beschrieben. Wir nutzen nun Matrizen, die einer Drehmatrix ähnlich sind.

Definition 2.3:

Wir bezeichnen eine $n \times n$ Matrix M als Drehmatrix wenn sie sich von der Einheitsmatrix E_n nur in vier Komponenten unterscheidet: $m_{i,i} = m_{j,j} = \cos(\phi)$, $m_{i,j} = -\sin \phi = -m_{j,i}$,

Diese Drehmatrix beschreibt eine einfache Drehung um einen Winkel ϕ in der x_i, x_j -Ebene. Die Komponenten sind allerdings nur im Fall $\phi = k \cdot \frac{\pi}{2}$, $k \in \mathbb{Z}$ ganzzahlig und auch wieder Permutationen. Wir können allerdings genau diese Komponenten durch andere Zahlen ersetzen, so dass wir die gesuchte Matrix T konstruieren, in dem wir in der entsprechenden Einheitsmatrix für festes i, j die Koordinaten $t_{i,i} = t_{j,j}$ und $t_{i,j} = -t_{j,i}$ jeweils mit einer beliebigen Zahl, die von 0 verschieden ist, belegen. So unterscheiden sich in $T \cdot T^T$ lediglich die Diagonalelemente $t_{i,i} = t_{j,j}$ von der Einheitsmatrix und $A = T^{-1} \cdot D \cdot T$ ist eine symmetrische Matrix, die unsere gesuchten Eigenwerte besitzt und schlimmstenfalls rationale Einträge enthält. Allerdings sind auch hier nur zwei Einträge $a_{i,j} = a_{j,i}$ ausserhalb der Diagonalen von Null verschieden. Da wir, wie oben diskutiert, nur kleine Dimensionen betrachten, stellt dies aber keine große Einschränkung dar.

Algorithmus 2.4:

Erstellung einer $n \times n$ -Matrix die einer Drehmatrix ähnlich ist mit Einträgen zwischen $-\max$ und $\max$

Eingabe: $n, \max$

Ausgabe: gesuchte Matrix A

1. erstelle Matrix A als Einheitsmatrix
2. wähle zufälliges Tupel (i, j) mit $i \neq j$ und $1 \leq i, j \leq n$
3. bestimme zwei zufällige Zahlen x und y mit $-\max \leq x, y \leq \max$
4. setze $A[i, i] : x$, $A[j, j] : x$, $A[i, j] : y$ und $A[j, i] : -y$
5. gebe A aus

Anmerkungen: Alternativ könnten, bei entsprechender Einschränkung der Randomisierung, auch verschiedene echte Drehmatrizen erstellt werden.

Für größere Dimensionen ließe sich das oben beschriebene Problem der vielen Nulleinträge umgehen, indem wir die Matrix T durch Multiplikation mehrerer dieser Matrizen erstellen. Hier sind nun aber zwei Punkte zu beachten: Zum Einen müssen wir bei der Wahl der Paare i, j darauf achten, dass dieselben Indizes nicht zu häufig auftreten, da sonst die Einträge zu groß werden. Zum Anderen sind wir nun nicht mehr frei in der Auswahl der Einträge unserer Matrix. Um die Symmetrie zu erhalten, müssen wir uns, wie in [MJO24] vorgeschlagen, mit pythagoreischen Tripeln begnügen und so auch eine echte Drehmatrix verwenden, so dass nur Einträge z.B. aus $\{\pm\frac{4}{5}, \pm\frac{3}{5}\}$ gewählt werden, wie im Anhang A.7 implementiert.

2.1.2 Erstellung einer Aufgabe

An dieser Stelle soll der Aufbau und die Erstellung einer STACK-Aufgabe erklärt werden. Als Beispiel dient hier eine Aufgabe der Linearen Algebra zum Thema Eigenwerte.

Aufgabe

Gegeben sei eine quadratische 2×2 oder 3×3 Matrix A . Gesucht ist der kleinste/größte Eigenwert und dessen algebraische sowie geometrische Vielfachheit.

Randomisiert werden die Einträge der Matrix und ob der größte oder kleinste Eigenwert gesucht wird. Um die erstellten Matrizen kontrollieren zu können und zu vermeiden, dass unschöne Ergebnisse auftreten, möchten wir die Eigenwerte vorgeben können und auch die Vielfachheit dieser Werte kontrollieren.

Es wäre möglich, die Komponenten $a_{i,j}$ der Matrix einfach zufällig zu wählen. Ein kleiner Test in Maxima zeigt allerdings, dass von 1000 3×3 -Matrizen dieser Art selten mehr als 10 einen doppelten Eigenwert besitzen und in den anderen Fällen die Eigenwerte auch seltenst ganzzahlig sind.

Dies legt ein Vorgehen ähnlich der Aufgaben zur Diagonalisierung nahe, in denen A mit Hilfe der Diagonalmatrix D und einer regulären Matrix S erstellt wird. Anstelle der Diagonalmatrix D verwenden wir nun aber eine Jordanform J .

Um spätere Anpassungen zu erleichtern, wird die Aufgabe allgemein für $n \times n$ ausgearbeitet. Die Jordanblöcke werden allerdings höchstens Blöcke der Dimension 2 sein.

Gegeben Sei die Matrix $A = \begin{pmatrix} 1 & 2 & -2 \\ -1 & 0 & 2 \\ -1 & 0 & 2 \end{pmatrix}$.

Geben Sie den kleinsten Eigenwert λ_e und seine algebraische und geometrische Vielfachheit $\mu_{alg}(\lambda_e)$ bzw. $\mu_{geo}(\lambda_e)$ an.

$\lambda_e =$

$\mu_{alg}(\lambda_e) =$

$\mu_{geo}(\lambda_e) =$

Abbildung 2.1: Studierendenansicht der Frage

Die Matrix S wird dann als ganzzahlig invertierbare Matrix wie in 2.1.1 beschrieben, erstellt.

Die Aufgabenvariablen

Zunächst gehen wir auf die Aufgabenvariablen der STACK-Aufgabe ein. An diesem Punkt werden alle für die Erstellung der Aufgabe notwendigen Variablen und Funktionen definiert. Die Definition erfolgt in Maxima mit wenigen Abweichungen durch STACK-spezifische Befehle. Angenehmer ist es, diesen Teil zunächst in einem externen Maxima-Editor durchzuführen, wobei man entweder die STACK-Sandbox nutzt oder die wenigen abweichenden Funktionen selbst definiert.

- Zunächst setzen wir die Dimension n der Matrix A fest. Dies kann einfach durch

```
n:3;
```

oder aber auch durch eine zufällige Größe wie

```
n:rand([2,3])
```

geschehen.

- Als nächstes wird eine Begrenzung für die Komponenten der A zugrundeliegenden Matrizen gegeben. Beispielsweise wird mit

```
max:2
```

festgelegt, dass alle Eigenwerte und Einträge der Matrizen L und R betragmäßig kleiner gleich 2 sind.

- Nun wird entschieden, ob nach dem größten oder kleinsten Eigenwert gesucht wird. Wir setzen

```
kg:rand([0,1])
```

und suchen, wenn $kg = 0$ gilt, den kleinsten Eigenwert. Die Programmierung der Aufgabe lässt sich vereinfachen, wenn man jeweils eine einzelne Aufgabe für beide Typen erstellt und Moodle dann zufällig entscheiden lässt, welche Aufgabe im Test erscheint.

- Als nächste Variable wird die Anzahl der 2×2 -Jordanblöcke bestimmt. Hier verwenden wir

```
k:rand(floor(n/2)+1)
```

Dies garantiert $0 \leq k \leq \frac{n}{2}$. Möglich wäre es auch hier eine Zahl festzulegen, dass alle dieselbe Anzahl solcher Blöcke bekommen.

- Mit der Funktion

```
makeRegNN(n,n)
```

wird eine ganzzahlig invertierbare $n \times n$ -Matrix S aus den Matrizen L und R generiert. (vgl. Kap. 2.1.1)

- Um die Jordanform zu generieren, wird zunächst eine Liste der Eigenwerte für jeden Block bestimmt.

$EW = [ew_1, \dots, ew_{n-k}]$, $ew_i \in \mathbb{N} \cap [-\max, \max]$. Mit Hilfe dieser Eigenwerte wird ein Array d definiert, aus dem die Jordanmatrix J generiert werden kann.

```
d[i,j]:=if i<=2*k and j<=2*k
then block([1:ceiling(i/2)]),
if i=j then EW[1]
else if i=j-1 and evenp(j) then 1
else 0)
else block(if i=j then EW[i-k]
else 0);

J:genmatrix(d,n,n)
```

- Wir erhalten nun die anzuzeigende Matrix $A = S \cdot J \cdot S^{-1}$.
- Die Musterlösung wird als Liste angegeben mit $\text{TansEW} = [\lambda_{ges}, \mu_{alg}, \mu_{geo}]$, wobei λ_{ges} der gesuchte Eigenwert ist. Da wir bei der Erstellung der Matrix A weder den gesuchten Eigenwert, noch die algebraische oder geometrische Vielfachheit explizit vorgegeben haben, müssen wir diese bestimmen, indem wir uns die Jordan-Matrix J anschauen aus der wir jeden Eigenwert, sowie deren jeweilige Vielfachheiten ablesen können. Dafür müssen wir aus der Liste möglicher Eigenwerte zunächst den gesuchten Eigenwert (also größten oder kleinsten) herausuchen, und im Anschluss die Vielfachheiten zählen.

```

ews: makelist(J[i,i],i,1,n);

TansEW: block([antw:[first(ews),1,1]],
  if kg=0 then for i:2 thru length(ews) do
    block(if ews[i]<antw[1] then antw:[ews[i],1,1]
      else if antw[1]=ews[i]
        then antw[2]:antw[2]+1)
  else for i:2 thru length(ews) do
    block(if ews[i]>antw[1] then antw:[ews[i],1,1]
      else if antw[1]=ews[i]
        then antw[2]:antw[2]+1),
  antw);

TansEW[3]: block([li:EW,i:0],
  while member(TansEW[1],li) do
    block(i:i+1,li:delete(TansEW[1],li,1))
  ,i)

```

Beispielswerte

Setzen wir zur Veranschaulichung Werte für die obigen Variablen ein. Wir wählen eine 4×4 -Matrix, um die Funktionsweise der verwendeten Algorithmen besser nachvollziehen zu können. Wir suchen den größten Eigenwert, also ist $kg = 1$ und wir verwenden kleine Zahlen, setzen deswegen `max` auf zwei. Mit $k = 1$ besitzt die Jordannormalform einen 2×2 -Jordanblock. In dem oben beschriebenen Ablauf wird lediglich Dimension und `max` durch den Dozenten, der die Aufgabe stellt, gesetzt und alles weitere randomisiert. Randomisiert wird zunächst die reguläre Matrix S . In diesem Beispiel wäre das etwa

$$S = \begin{pmatrix} -1 & -1 & 2 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}.$$

Nun erhalten wir die Liste der Eigenwerte. Wegen $k = 1$ enthält sie nun drei nicht zwangsläufig verschiedene Eigenwerte. Hier z.B. $EW = [-1, -2, -1]$. Aus diesen Eigenwerten generieren wir eine Jordanmatrix:

$$J = \begin{pmatrix} -1 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

Zusammen mit S generieren wir nun die Matrix A , die in der Aufgabe untersucht werden soll. Wir erhalten

$$A = \begin{pmatrix} -4 & -3 & -4 & 3 \\ 1 & 0 & 2 & -1 \\ 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

Da die Matrizen zufallsbasiert entstanden sind, müssen wir nun eine Musterlösung erstellen. Hierfür suchen wir aus der Liste aus Diagonalelementen $[-1, -1, 2, -1]$ von J nun den größten Wert, also -1 , aus, da ja $kg = 1$ gilt. Wir zählen, wie oft -1 in der Liste auftritt und erhalten die algebraische Vielfachheit, hier also $\mu_{alg} = 3$. Danach überprüfen wir, wie oft der gesuchte Eigenwert in der Liste EW auftritt und ermitteln so, dass $\mu_{geo} = 2$ gelten muss. Die Musterlösung, auf die STACK zur Korrektur zugreifen kann, ist demnach $\mathbf{TAns} = [-1, 3, 2]$.

Der Fragetext

Im Fragetext wird die für die Studierenden anzuzeigende Frage eingegeben. Hier ist $\text{\LaTeX}$ -Code möglich, auf Aufgabenvariablen kann mit $\{@var@\}$ zugegriffen werden. In dieser Aufgabe werden programmierbare Question-Blocks eingesetzt, so dass je nach Typ kg (2.1.2) entweder das Wort größter oder kleinster angezeigt wird. Hierzu muss überprüft werden, ob `test` mit

```
test = "is(kg=0)"
```

wahr oder falsch ist (Abbildung 2.2).

Wesentlich sind in diesem Eingabefeld die Antwort- und Validationsblöcke. Prinzipiell sind hier auch eigene Bezeichnungen der Antwortfelder möglich um die Aufgabe übersichtlich zu gestalten. Jede Antwort muss auch validiert werden können. Bei diesem Beispiel, indem nur numerische Werte abgefragt werden, ist der Gewinn für die Studierenden gering, in Fragen nach Termen allerdings äußerst hilfreich.

Allgemeines

Weitere allgemeine Eingabefelder sind Name der Frage, die Kategorie, in der sie gespeichert wird (wird beim Erstellen angelegt), die erreichbaren Punkte (default: 1) und Abzüge (nur bei Mehrfachbeantwortung nötig). Außerdem gibt es die Feedbackfelder, das spezifische Feedback, das die im folgenden beschriebenen Feedbackbäume aufzählt, die die Antworten der Studierenden überprüfen sollen und das allgemeine Feedback, das jeder Testteilnehmer unabhängig von der eigenen Antwort angezeigt bekommt. Auch hier ist

↓

A ▾

B

I

≡

≡

≡

≡

🔗

🔗

😊

🖼️

📄

HP

♻️

👤

Gegeben sei die Matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 3 & 3 & 1 \end{pmatrix}$.

Geben Sie den ☐ kleinsten ☐ größten ☐ Eigenwert λ_e und seine algebraische und geometrische Vielfachheit $\mu_{alg}(\lambda_e)$ bzw. $\mu_{geo}(\lambda_e)$ an.

$\lambda_e =$

$\mu_{alg}(\lambda_e) =$

$\mu_{geo}(\lambda_e) =$

Abbildung 2.2: Backend eines Fragetexts mit Question Block

Gegeben Sei die Matrix $A = \begin{pmatrix} -12 & -11 & 25 \\ 0 & 2 & 0 \\ -4 & -2 & 8 \end{pmatrix}$.

Geben Sie den größten Eigenwert λ_e und seine algebraische und geometrische Vielfachheit $\mu_{alg}(\lambda_e)$ bzw. $\mu_{geo}(\lambda_e)$ an.

$\lambda_e =$

Ihre letzte Antwort wurde folgendermaßen interpretiert: 2

$\mu_{alg}(\lambda_e) =$

Ihre letzte Antwort wurde folgendermaßen interpretiert: 1

$\mu_{geo}(\lambda_e) =$

Ihre letzte Antwort wurde folgendermaßen interpretiert: 1

Abbildung 2.3: Frontend der Fragestellung mit Validierung

Aufgabenhinweis



$A: \{A\}, J: \{J\},$ gesuchter Eigenwert mit Häufigkeiten: $\{TansEW\}$

Abbildung 2.4: Aufgabenhinweise mit zufälliger Matrix

Variante	Aufgabenhinweis
571728499	$A: \begin{pmatrix} -12 & -11 & 25 \\ 0 & 2 & 0 \\ -4 & -2 & 8 \end{pmatrix}, J: \begin{pmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 2 \end{pmatrix},$ gesuchter Eigenwert mit Häufigkeiten: $[2, 1, 1]$
2055475512	$A: \begin{pmatrix} -2 & 1 & 0 \\ -4 & 2 & 0 \\ 0 & 0 & -2 \end{pmatrix}, J: \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -2 \end{pmatrix},$ gesuchter Eigenwert mit Häufigkeiten: $[0, 2, 1]$

Abbildung 2.5: eingesetzte Varianten der Aufgabe

L^AT_EX-Code möglich. Die Feedbackbäume können beliebig benannt werden, hier verwende ich aber die Standardbenennung `prt1`, `prt2` und `prt3`.

Bei randomisierten Aufgaben sind die Aufgabenhinweise notwendig. Dies Feld darf in diesem Fall nicht leer bleiben. Es empfiehlt sich die Aufgabe und Lösung hier einzutragen, so dass man bei Rückfragen der Studierenden auch die Lösung schnell zur Hand hat. Erklärender Text kann auch hilfreich sein, wie in Abbildung 2.4 verwendet.

Beim automatischen Erstellen eines Satzes Aufgaben bricht STACK bei gleichen Aufgabenhinweisen ab, es sollten also alle wesentlichen randomisierten Variablen aufgeführt werden.

Sind alle beschriebenen Felder ausgefüllt, kann der Aufgabentext überprüft werden und die Felder aktualisiert werden. Hierbei werden Eingabefelder und Feedbackbäume angelegt oder umbenannt, die nun diskutiert werden.

Eingabe: Antwort

Für jedes Eingabefeld wird ein solcher Block angelegt. Hier wird der Typ und die Größe der Antwort festgelegt sowie eine Musterlösung hinterlegt. Als Typ sind algebraische Eingaben aber auch Matrizen oder Dropdownlisten möglich. Auch bei offenen Aufgaben muss eine Musterlösung angegeben werden. Ansonsten dürfen an dieser Stelle auch Einschränkungen an die Eingabe gemacht werden, wie z.B. den Begriff **int** verbieten. Man kann hier auch die Validierung ausschalten, falls gewünscht.

▼ Eingabe: ans1

Eingabetyp	?	Algebraische Eingabe ▾
Musterlösung	?	TansEW[1]
Größe der Eingabebox	?	15
Strenge Syntax	?	Ja ▾
Sternchen einfügen	?	Keine Sternchen einfügen
Syntax-Hinweis	?	
Hinweis-Attribut	?	Wert ▾
Verbotene Wörter	?	
Erlaubte Wörter	?	

Abbildung 2.6: Backend eines Eingabefelds

Feedbackbäume

In einer Aufgabe können beliebig viele Feedbackbäume erstellt werden. Da ein Feedbackbaum nur durchlaufen wird, wenn alle in ihm verwendeten Eingabefelder einen Wert besitzen, kann es sinnvoll sein, für jedes Aufgabenfeld einen Feedbackbaum zu definieren, so dass Teilpunkte auf eine Aufgabe gegeben werden, wenn die letzten Felder leer geblieben sind. Bei unserer Aufgabe haben wir uns somit für drei Feedbackbäume entschieden.

Jeder Feedbackbaum besteht aus mindestens einem Knoten und zwei Blättern. Der Begriff Baum ist allerdings etwas irreführend, es handelt sich hier um gerichtete Graphen, die aber nicht notwendigerweise kreisfrei sind. Ein Knoten kann also, wenn sinnvoll, auch auf unterschiedlichen Pfaden erreicht werden. In jedem Knoten wird eine Studierendenantwort mit einer Musterantwort verglichen und als wahr oder falsch klassifiziert.

STACK liefert unterschiedliche Überprüfungsmethoden. AlgEquiv ist aber oft zielführend. Es sind auch numerische Überprüfungen oder welche, die auf algebraische Vereinfachungen, Integral- und Differenzialrechnung optimiert sind, möglich. Bei dem Feedback, das man an dieser Stelle unterdrücken kann, handelt es sich nur um die Meldung, wie dieser Test ausging, nicht um das unten eingetragene Feedback. Nun wird die Reaktion auf die Überprüfung definiert, wir geben an, an welchem Knoten es weitergeht, ob es

Knoten 2

Antwortüberprüfung: AlgEquiv SAns: ans2 TAns: TansEW[2]

Test-Optionen: Feedback unterdrücken: Ja

Knoten 2 wenn WAHR: Mod: + Score: 1 Abzüge: Nächster: [stop] Antworthinweis: prt2-2-T

Knoten 2 WAHR feedback: Sie haben die dazugehörige algebraische Vielfachheit korrekt bestimmt.

Knoten 2 wenn FALSCH: Mod: - Score: 0 Abzüge: Nächster: Knoten 3 Antworthinweis: prt2-2-F

Knoten 2 FALSCH Feedback: Leider ist die dazugehörige algebraische Vielfachheit falsch.

Abbildung 2.7: Knoten eines Feedbackbaums

Punkte gibt oder sogar Abzüge, und ob und was für ein Feedback auf diese Antwort angezeigt wird. (vgl: Abbildung 2.7) Wir können beliebig viele Knoten hinzufügen um eine Antwort unterschiedlichen Tests zu unterziehen.

In diesem Beispiel entscheidet der erste Feedbackbaum nur, ob der richtige Eigenwert gefunden wurde. Ist dieser korrekt, gibt es für diesen Feedbackbaum einen Punkt und eine entsprechende Meldung. Andernfalls den Hinweis, dass die Eigenwerte bestimmt werden müssen und zeigt das korrekte charakteristischen Polynom an.

Der zweite Feedbackbaum verwendet zwei Eingaben, sowohl die für den Eigenwert als auch die für die algebraische Vielfachheit. Wurde der falsche Eigenwert angegeben, bricht der Baum sofort ab, da eine Überprüfung der Vielfachheit so nicht zielführend ist, da wir nicht überprüfen, wie dieser Eigenwert zustande kam. Im zweiten Knoten vergeben wir einen Punkt für diesen Baum, falls die Vielfachheit korrekt ist. Falls nicht, überprüfen wir im dritten Knoten noch, ob hier eventuell 0 als Antwort angegeben wurde und weisen darauf hin, dass dies per Definition ausgeschlossen ist.

Der dritte Feedbackbaum zur Überprüfung der geometrischen Vielfachheit sieht ähnlich aus, hier wurde noch ein vierter Knoten hinzugefügt, der überprüft, ob die angegebene algebraische Vielfachheit kleiner als die geometrische ist.

Insgesamt sind also drei Feedbackbäume mit einem Wert von jeweils einem Punkt erstellt worden, der Aufgabenwert selber ist ein Punkt, damit trägt jeder Feedbackbaum also einen drittel Punkt bei.

Optionen, Schlagwörter

Für diese Aufgabe benötigen wir keine Änderungen in den Einstellungen, hier können automatische Feedbacktexte angepasst, Klammern bei Matrizen

▼ Test-Eingaben

ans1

TansEW[1]

ans2

TansEW[2]

ans3

0

Füllen Sie den Rest des Formulars aus, um einen Testfall zu erstellen

▼ Erwartete Ergebnisse

prt1: [ans1]

Score

1

Abzüge

0

Antworthinweis

prt1-1-T

prt2: [ans1, ans2]

Score

1

Abzüge

0

Antworthinweis

prt2-2-T

prt3: [ans1, ans2, ans3]

Score

0

Abzüge

0.1

Antworthinweis

prt3-3-T

Testfall erstellen

Abbrechen

Abbildung 2.8: Testfall mit falscher geometrischer Vielfachheit

geändert werden oder ein paar andere häufige Konventionen in der Anzeige angepasst werden. Schlagwörter können gerne eingegeben werden, Sie erleichtern das Wiederfinden der Aufgabe in der Fragesammlung.

Fragetest und eingesetzte Varianten

Ist die Frage gespeichert, kann sie vom System automatisch getestet werden. Hierfür werden Testfälle erstellt, z.B. was passiert, wenn wir die Musterlösung eingeben, was, wenn wir einen typischen Fehler gemacht haben. Für die Eingaben müssen wir auf die obigen Variablen zugreifen, so dass diese Testfälle wirklich für alle Versionen funktionieren können.

Nun kann man STACK automatisch einige Varianten einsetzen lassen, die dann den Studierenden gestellt werden. Hierbei wird auch überprüft, ob sie diesen Testfällen genügen. Die Lehrenden können in dieser Anzeige auch einzelne Varianten löschen, falls diese trivial oder zu schwer erscheinen, so dass diese nicht in den Tests auftauchen werden. Oft lohnt es sich in diesen Fällen aber auch die Aufgabenvariablen noch einmal unter die Lupe zu nehmen, so dass diese Fälle beim nächsten Einsatzversuch gar nicht erst erstellt werden.

Auslagern von Programmcode

Werden viele Zeilen Code für eine Aufgabe benötigt oder tauchen bestimmte eigene Funktionen in vielen Aufgaben auf, so kann es sinnvoll werden, Aufgabenteile auszulagern. Dies ist seit Version 4.4 aus dem Juli 2022 möglich. Hierfür muss eine Textdatei aufgerufen werden in der die benötigten Inhalte zu finden sind. Dies kann beispielsweise Maximacode sein, der innerhalb der Aufgaben- oder Feedbackvariablen benötigt wird, oder auch Bilder und JSX-Code für die Anzeige von Frage, Feedback und Musterlösung. Ein Minimalbeispiel könnte beispielsweise folgendermaßen aussehen:

```
a:rand(10);
b:rand(20);
stack_include("https://www.mathematik.uni-kassel.de/~
heschmpo/test_include.txt");
/*laedt Textdatei, die die Funktion addOneMore
enthaelt*/
lsng: addOneMore(a,b)
```

Sehr sinnvoll ist dies auch, wenn man die Randomisierung kontrollieren will. So kann man eine Liste von besonders geeigneten Matrizen für eine Aufgabe in einer Datei ablegen und muss nicht später anhand der Fragetests die ungeeigneten Matrizen händisch aussieben. Wie im folgenden in Kapitel 2.2.3 diskutiert, kann Randomisierung in einer Aufgabe auch dazu führen, dass wir sie selber nicht mit den zur Verfügung stehenden Mitteln allgemein lösen können. Hier hilft es, eine Liste der Versionen vorliegen zu haben, von denen wir die Eigenschaften bereits kennen.

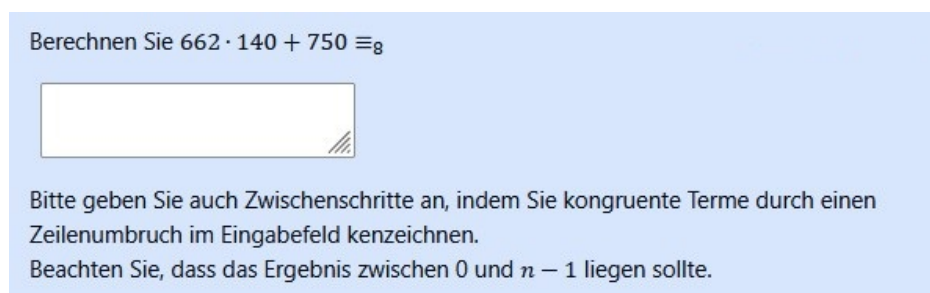
Wird die Aufgabe gespeichert, wird der Code aus der aufgerufenen Textdatei geladen und der gespeicherten Aufgabe hinzugefügt. Ein Aufrufen der Frage durch Studierende lädt die Datei nicht neu. Änderungen in der Quelldatei haben dementsprechend erst mit einem erneuten Speichern der erstellten Aufgabe einen Effekt [SD22]. Dies ermöglicht die Nutzung des extern gelagerten Codes, auch wenn keine Verbindung zum Server, auf dem die Quelldatei liegt, existiert. Es bietet sich an, in den Quelldateien die Aufgaben, die Code aus der Datei benötigen, anzugeben, so dass erneuerter Code auch in den entsprechenden Fragen übernommen wird.

2.1.3 Rechnen mit Restklassen und simplify

Das Rechnen mit Restklassen stellt ein Computeralgebrasystem vor keine großen Herausforderungen. Das Ergebnis von Termumformungen kann mithilfe des `mod(a,n)`-Befehls leicht überprüft werden. Schwieriger wird es, wenn man Zwischenschritte abfragen möchte. In dieser Aufgabe soll mindestens ein Zwischenergebnis angegeben werden. An dieser Stelle müssen Vereinfachungen abgeschaltet werden und die Zwischenschritte manuell gesetzt werden. Die eingebauten Antworttest helfen leider nicht weiter, da Kongruenzen nicht vorgesehen sind.

Wir erstellen in dieser Aufgabe die Zahlen $n \in \{4, 5, 6, 8, 9\}$ und $a, b, c \in \{10n + 1, 10n + 2, \dots, 101n + 1\}$ und lassen die Studierenden den Term $a \cdot b + c$ als Kongruenz modulo n bestimmen, wobei das Endergebnis zwischen 0 und $n - 1$ liegen soll.

Wir setzen die Lösung: `lsng:mod(a*b+c,n)` und berechnen `a1:mod(a,n)` und



Berechnen Sie $662 \cdot 140 + 750 \equiv_g$

Bitte geben Sie auch Zwischenschritte an, indem Sie kongruente Terme durch einen Zeilenumbruch im Eingabefeld kenzeichnen.

Beachten Sie, dass das Ergebnis zwischen 0 und $n - 1$ liegen sollte.

Abbildung 2.9: Frage zu Kongruenzen

analog b_1 und c_1 sowie den Zwischenschritt $a1 * b1 + c1$.

Nach diesem Schritt muss das Vereinfachen der Terme durch `simp:false` abgeschaltet werden. Wir erstellen zuletzt eine Lösungsmatrix mit `lsngMat:matrix([ev(a1*b1+c1,noeval)], [10], [lsng])`, die in der ersten Zeile den Term $a1 * b1 + c1$, in der zweiten dann das Ergebnis dieser Summation und in der dritten Zeile das Endergebnis enthält.

Die Eingabe soll hier als Matrix mit variabler Zeilenzahl erfolgen, so dass beliebig viele Zwischenschritte möglich sind. Offensichtlich könnten die Zeilen zwei und drei der Musterlösung identisch sein. Wahlweise wäre eine Eingabe als Liste oder in unterschiedliche Felder möglich, die Eingabe in der hier verwendeten Form erscheint aber am handlichsten aus Studierendensicht.

Um die Antwort allerdings auszuwerten, müssen wir die Matrix zunächst

wieder in eine Listenform bringen und diese Liste in das Endergebnis `ansFinal` $\in \mathbb{N}$ und die Zwischenschritte `ansSteps` als Liste der Terme ohne das Endergebnis aufteilen. An dieser Stelle können wir das `simplify` wieder aktivieren, so dass wir wieder mit den einzelnen Ergebnissen arbeiten können. Zunächst überprüfen wir, ob das Endergebnis korrekt ist und geben hierfür einen Punkt. Wurden aber keine Zwischenschritte angegeben, so reduziert sich das Ergebnis wieder. Zuletzt überprüfen wir, ob sich in den Umformungen Fehler eingeschlichen haben, indem wir von Zeile zu Zeile die Kongruenzen mittels der Funktion `findFalseEquiv(ansList, a*b+c,n)` überprüfen.

2.1.4 Graphische Darstellungen

Zur Veranschaulichung mathematischer Inhalte sind graphische Darstellungen häufig hilfreich. In der Analysis werden beispielsweise Funktionen und ihre Eigenschaften diskutiert und viele dieser Eigenschaften können am Funktionsgraphen verdeutlicht werden. Die Anzeige eines Plots einer Funktion ist also wünschenswert.

Geometrische Problemstellungen ausschließlich textuell zu beschreiben ist in erster Linie mühselig und auch die Graphentheorie profitiert von anschaulichen Skizzen. Aus diesem Grund bietet es sich an auch graphische Elemente in einige Aufgaben zu integrieren. Hierzu kann der STACK-eigene `plot`-Befehl genutzt werden. Mehr Möglichkeiten bietet aber die Verwendung von Geogebra oder JSXGraph.

Plots in Maxima

Verwendet man Maxima in der wxMaxima-Umgebung stehen einem die Befehle `(wx)plot2d` und `(wx)plot3d` zur Verfügung. Mit entsprechenden Optionen kann man so einiges an grafischen Ausgaben erzeugen. Das vorangestellte `wx` sorgt hier dafür, dass die erzeugten Plots direkt als Bild in der wxMaxima-Sitzung ohne eigenes Fenster angezeigt werden, dafür stehen so aber keine Werkzeuge wie Verschieben oder Koordinatenanzeigen zur Verfügung. Das `draw`-Paket liefert hier noch weitere vielfältige Möglichkeiten [Haa14]. Maxima verwendet hier das Programm Gnuplot. Um eine sichere Verwendung unter moodle zu gewährleisten wurden diese Befehle deaktiviert, der neue Befehl `plot` dient als Wrapper und erzeugt Plots, die direkt in der Aufgabe angezeigt werden können. Die Optionen sind hierbei etwas eingeschränkt worden, dreidimensionale Plots sind nicht möglich. Nichtsdestotrotz liefert STACK auch mit seinem `plot`-Befehl ein nützliches Werkzeug um relativ einfach eine grafische Ausgabe zu erzeugen.

Ein Beispiel zum Einsatz des STACK-eigenen Plots finden wir bei einer Frage zu der Lösungsmenge einer kubischen Ungleichung.

Zu der Lösung wird hier auch ein Plot der Funktion angezeigt.

Geben Sie die Lösungsmenge $L \subset \mathbb{R}$ der folgenden Ungleichung an. $(16 \cdot x^2 - 9) \cdot (5 \cdot x - 1) > 0$.

((Nicht beantwortet) (Nicht beantwortet))

Hinweis: Geben Sie eine Lösung der Form $(0,6)$ ein als $0 < x$ and $x < 6$, hängen Teile der Lösung nicht zusammen verwenden Sie $x < 0$ or $6 < x$

Abbildung 2.10: Gesucht wird die Lösungsmenge einer Ungleichung

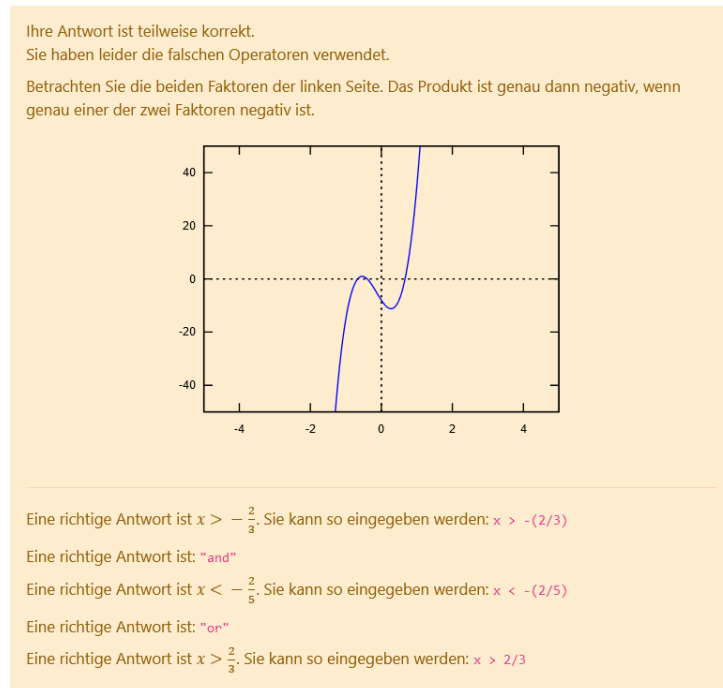


Abbildung 2.11: allgemeines Feedback und Musterlösung zur Ungleichung aus Abb. 2.10

Ein diskreter Plot wird bei einer Fragestellung zu Parallelogrammen verwendet. Der Code 2.13 liefert mit diversen vorher gesetzten zufälligen Werten für ax , ay , ux , uy , vx und vy eine Skizze des Parallelogramms.

Grafiken mit JSXGraph

Dynamische Grafiken sind mit JSXGraph möglich. JSXGraph ist eine JavaScript Bibliothek, die an der Universität Bayreuth entwickelt wurde [GVW10] und [Vas18]. In Aufgabenstellung und Feedback können JSXGraph-Applets eingefügt werden. Die zu skizzierenden Elemente werden hier als Objekte definiert mit allen benötigten Eigenschaften. Wir können beispielsweise Punkte, Geraden und Funktionsgraphen definieren, diesen ihre Koordinaten, Label, Farben, etc. zuschreiben und erklären, ob sie fixiert werden sollen oder ob sie frei bewegbar sind. Auf diese Art und Weise können nicht nur beliebige Plots erzeugt werden, sondern Studierende können auch mit den Plots experimentieren. Ebenso ist es möglich Eingaben durch diesen Plot als Ergebnis

Gegeben sei ein Parallelogramm $ABCD$ bei dem die Eckpunkte gegen den Uhrzeigersinn benannt sind.

Der Ortsvektor des Punktes A sei $\begin{pmatrix} -3 \\ 0 \end{pmatrix}$.

Die von A ausgehenden Seiten haben die Richtungsvektoren $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$ und $\begin{pmatrix} 5 \\ -5 \end{pmatrix}$.

- Berechnen Sie die Ortsvektoren der Punkte B , C und D :

$B =$

$$\begin{pmatrix} \boxed{} \\ \boxed{} \end{pmatrix}$$

Abbildung 2.12: Bestimmung von weiteren Eckpunkten eines Parallelogramms

```
{@plot([[discrete,[[ax,ay],[ax+vx,ay+vy],
[ax+vx+ux,ay+vy+uy],[ax+ux,ay+uy],
[ax,ay],[ax+vx+ux,ay+vy+uy],
[ax+ux,ay+uy],[ax+vx,ay+vy]]]],
[x,ax-2,ax+vx+ux+2],[y,ay+vy-1,ay+uy+1],
[label,["A",ax,ay-0.5],["B",ax+vx,ay+vy-0.5],
["C",ax+vx+ux,ay+vy+uy-0.5],["D",ax+ux,ay+uy-0.5]])@}
```

Abbildung 2.13: Codefragment zur Erstellung eines Parallelogramms

zuzulassen. Zum Beispiel kann ein Punkt verschoben werden und die neue Position des Punktes wird als Antwort eingelesen und muss nicht zusätzlich in ein Feld eingegeben werden.

Ein Beispiel zur Graphentheorie diskutieren wir in Abschnitt 2.2.3, eine kurze Einführung zu JSXGraph findet sich in Anhang C.

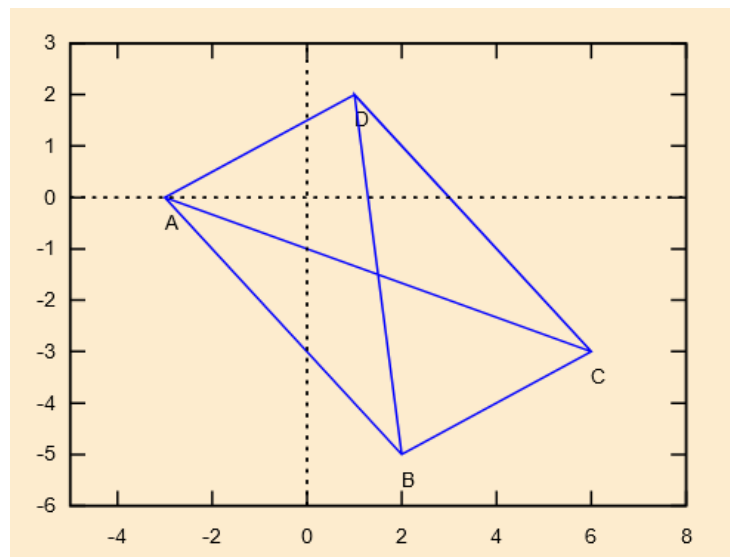
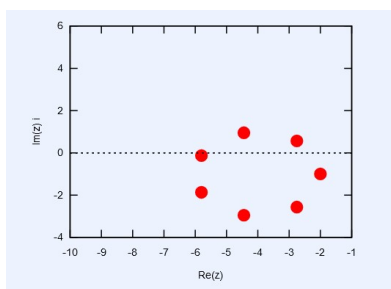


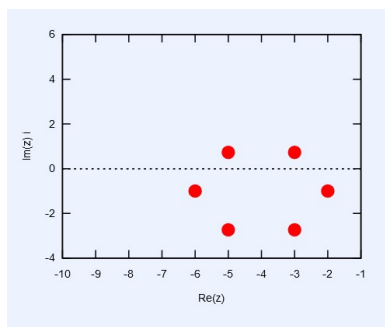
Abbildung 2.14: gezeichnetes Parallelogramm zur Frage aus Abb. 2.12

2.1.5 Erkennen von komplexen Wurzeln

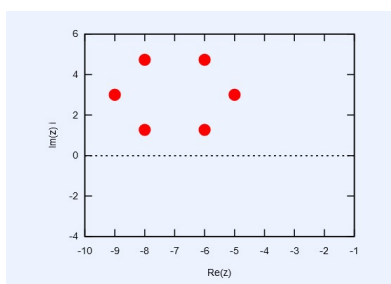
Um den Umgang mit komplexen Wurzeln zu üben, werden Studierende in einer Aufgabe aufgefordert, eine komplexe Potenzgleichung der Form $(z - z_0)^n = a$ mit $z_0 \in \mathbb{C}$ mit Hilfe von grafischen Methoden zu lösen. Hierbei gilt $n \in \{3, \dots, 9\}$ und Real- bzw. Imaginärteil von z_0 liegen zwischen -4 und 4. Die rechte Seite bestimmen wir mit $a = b^n$, wobei $b \in \{1, 2, 3, 4\}$ gilt. Den Studierenden werden vier Plots angezeigt, wovon einer die richtigen Lösungen der Wurzelgleichung zeigt, ein zweiter verschobene Lösungen, ein dritter Lösungen, die zu einer anderen Potenz gehören und ein vierter Lösungen, die zu einem anderen Radius passen. Der Code zur Erstellung der Aufgabe findet sich unter Anhang A.15.



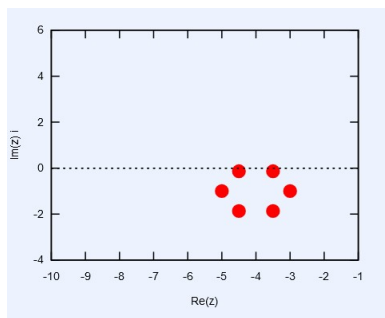
(a) falsche Anzahl Lösungen



(b) korrekte Lösung



(c) verschobene Lösung



(d) Radius falsch

Abbildung 2.15: Lösungen zu $(z + 1 + 4)^4 = 64$

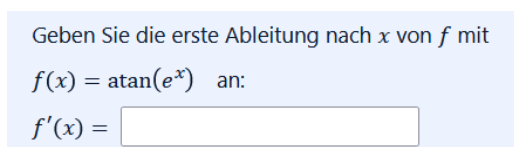
2.1.6 Zufällige, verkettete Funktionen

Es liegt nahe, Studierende das Differenzieren verketteter Funktionen mit Hilfe von STACK-Aufgaben üben zu lassen. Um einen großen Pool Aufgaben zu erhalten, kann man eine Liste verschiedener Funktionstypen, teilweise auch mit Zufallsvariablen, erstellen und daraus zwei zufällige Funktionen g und h auswählen und diese verketteten. Dies könnten beispielsweise

```
fktAuswahl: [sin(x), cos(x), tan(x), atan(x), acos(x), asin(x), log(x), e^x,
(rand(n)+1)^x, (x-rand_with_step(-n,n,1))^(rand(n)+1),
```

```
sum(rand_with_step(-n,n,1)*x^i,i,0,4)] sein. Das Ableiten dieser Funktionen für sich stellt die wenigsten Studierenden vor zu große Herausforderungen. Sollen Sie allerdings die Kettenregel anwenden, so tauchen sehr typische Fehler auf. In der entsprechenden Aufgabe werden diese in den Feedbackbäumen abgefragt. Beispielsweise wird  $(g \circ h)(x)$  mit  $(g \circ h)'(x) = g'(h'(x))$  oder mit  $(g \circ h)'(x) = g'(h(x))$  angegeben. Diese typischen Fehler rückmelden zu können ist durchaus ein Mehrwert.
```

Betrachtet man allerdings die obige Funktionenauswahl, dann lässt sich,



Geben Sie die erste Ableitung nach x von f mit
 $f(x) = \operatorname{atan}(e^x)$ an:
 $f'(x) =$

Abbildung 2.16: Frage zur Kettenregel

wenn wirklich innere und äußere Funktion beliebig gewählt werden, auch $g: \mathbb{R} \rightarrow \mathbb{R}$ mit $g(x) = 4x^4 + 2x^3 + 5x^2 + 2x + 1$ als äußere Funktion wählen. Entsprechend lang können die abgeleiteten Funktionen werden, so dass Eingabefehler sehr wahrscheinlich sind.

An dieser Stelle wird ein Feature von STACK hilfreich, das unter Abschnitt 2.1.2 beschrieben wird. Zu einer Frage können Fragetests erstellt werden und die eingesetzten Varianten ausgewählt werden. Nehmen wir nun die Frage zur Kettenregel, so können wir das System auffordern eine Auswahl an Fragen, die eingesetzt werden, zu erstellen. So werden die Fragevarianten definiert und die Antworten der Studierenden lassen sich im Nachhinein auch einfacher analysieren, wenn die jeweilige Fragevariante festgelegt ist. Nebenbei erhält man bei geschickter Eingabe der Aufgabenhinweise auch eine Vorschau der Aufgabenvarianten auf einen Blick. So kann man an dieser Stelle besonders unschöne Varianten direkt aussortieren, so dass weder sehr eingabefehleranfällige noch zu triviale Versionen ausgegeben werden.

Möchte man an dieser Stelle auch Fragetests erstellen, so sollte zumindest überprüft werden, ob die Musterlösung auch richtig gewertet wird. Diese Aufgabe ist allerdings auch ein gutes Beispiel für Schwierigkeiten, die bei diesem automatisierten Feedback auftreten können: Betrachten wir die Funktionsvorschrift: $x \mapsto \arctan(e^x)$ mit $g(x) = \arctan(x)$ und $h(x) = e^x$. Um hilfreiches Feedback bei dieser Aufgabe zu geben, untersuchen wir die Eingabe der Studierenden auf typische Fehler. Dazu gehört auch, zu überprüfen, ob etwa $g'(h'(x))$ oder $g'(h(x))$ eingegeben worden ist. Hier ist nun aber $g'(h'(x)) = g'(h(x)) = \frac{1}{e^{2x}+1}$. Damit wird eine eventuell falsche Einschätzung zum Fehler, der dem Studierenden unterlaufen ist, gegeben und die entsprechenden Testfälle, die beim automatischen Testen der STACK-Frage angelegt werden, versagen.

2.2 Neue Aufgabentypen

2.2.1 Zufällige Auswertungen

Anders als eine Künstliche Intelligenz arbeitet ein Computeralgebrasystem nicht mit Erfahrungen. Während Studierende durch häufige Übung Strategien entwickeln und Zusammenhänge *einfach sehen*, sind z.B. manche Fragestellungen für ein CAS nicht trivial lösbar. Stellt man z.B. die Frage nach der Monotonie einer Funktion oder Folge, lässt sich dies auf dem Papier häufig schlüssig beantworten, MAXIMA liefert aber nur ein undefined.

Nun sind offene Aufgaben aber gerade sinnvoll für das Verständnis mathematische Zusammenhänge. Dies wird ausführlich von Chris Sangwin [San16] in seinem Buch, diskutiert, findet sich aber auch auf Fachtagungen Zustimmung [RV17]. Zum Thema Folgen und Grenzwerte bietet sich demnach an, die Studierenden eine Folge angeben zu lassen, die beispielsweise monoton fallend und beschränkt ist. Nun ist MAXIMA leider nicht geeignet Ungleichungen auszuwerten. Man könnte an dieser Stelle versuchen einige Standardfolgen, wie etwa $1/n$ einzuplanen, die vermutlich hauptsächlich eingegeben werden. Aber kleinste Änderungen wie eine einfache Substitution zu $1/(n+1)$ stellen schon eine große Herausforderung da. Auch Terme wie n benötigen Zusatzinformationen an Maxima, um Vereinfachungen durchführen zu können. Einige Eingaben lassen sich durch die gängigen Monotoniekriterien überprüfen und letztendlich kann mit Hilfe der eingebauten STACK-Antwortentests auch ein Großteil der eingegebenen Folgen abgefangen werden. Allerdings ist es nicht weiter schwer immer neue Eingaben zu konstruieren, die wieder durch alle analytischen Tests durchfallen.

Ein erster Ansatz ist hier das Überprüfen der ersten Folgenglieder. Sollte hier schon ein Widerspruch auftreten, kann die Folge gleich mit entsprechendem Feedback verworfen werden. Allerdings ist hier natürlich die Frage, ab dem wievielten Folgenglied dieser Versuch abgebrochen werden kann und was die damit verbundene Rechenzeit ist. Insbesondere bei Folgen mit Fakultäten können hier schnell große Rechenzeiten zusammenkommen (vgl. Abb. 2.17).

```
(%i29) for i:1 thru 10000 do if i!/(i+1)!>1 then display("false");  
Evaluation took 13.1875 seconds (13.2061 elapsed) using 2547.765 MB.  
(%o29) done
```

Abbildung 2.17: Überprüfung der ersten 10000 Folgenglieder der Folge $(a_n)_n$ mit $a_n = n \cdot (n-1) \dots 1$

Um dieses Problem zu umgehen, wird nach wenigen Versuchen, die Monotonie der eingegebenen Folge zu bestätigen oder zu widerlegen, ein an-

deres Verfahren eingesetzt. Wir wählen eine vernünftige Anzahl (Pseudo-) Zufallszahlen bis zu einer ebenfalls sinnvoll gewählten Grenze `max` aus und überprüfen für alle diese `ntest` Indizes $n_1, \dots, n_m$, ob das n_i -te Folgenglied die gewünschten Kriterien erfüllt. Das Feedback an die Studierenden muss in einem solchen Fall entsprechend angepasst werden. Wir können im Feedback nicht erklären, dass die Folge streng monoton ist, sondern nur, dass `max` Versuche die Monotonie zu überprüfen keinen Widerspruch ergeben haben. Sollte die Monotonie widerlegt werden, kann dieses konkrete Gegenbeispiel ausgegeben werden.

Eine Kombination mit einer groben analytischen Untersuchung erscheint besonders sinnvoll, da Studierende selbst ihre Wahl der Folge algebraisch begründen können sollten. Klassischerweise wird in Klausuren eine Begründung für jede getroffene Aussage gefordert. In Tutorien wird gewöhnlich Wert darauf gelegt, dass exemplarische Beispiele nicht als Beweis einer Aussage dienen können. Insofern ist im Feedback einer Aufgabe immer ein analytisches Argument zu bevorzugen. Ein Test auf zufällige Werte ermöglicht allerdings einen Hinweis darauf, ob die angegebene Lösung ein guter Kandidat ist. Dieses Verfahren wurde bereits auf der EAMS vorgestellt ([SS23]).

Es ist zu beobachten, dass die meisten von den Studierenden angegebenen Folgen durch das Differenzen- oder das Quotientenkriterium bestätigt werden können und eine stichpunktartige Untersuchung der höheren Folgenglieder selten nötig ist. Für zukünftige STACK-Versionen ist zudem die Integration der `let`-Funktion von Maxima vorgesehen, was die Möglichkeiten algorithmischer Untersuchungen deutlich erhöhen würde.

Eine Möglichkeit dieses zufallsbasierte Verfahren auszutricksen und ein "`vermutlich monoton`" bei einer nicht monotonen Folge zu erhalten, wäre die Definition einer Folge mit einem einzelnen Ausreißer an der Stelle n_k . Dies ist in Maxima auch ohne weiteres möglich, in STACK lassen sich die dafür notwendigen Befehle aber im Eingabefeld verbieten. In den STACK-Kern werden Funktionen nur zugefügt, wenn sie notwendig sind und in ihrem Lisp-Code keine Nebeneffekte versteckt sind. Nach momentanem Stand lässt sich demnach eine abschnittsweise definierte Funktion auch nicht etwa durch Dirac-Impulse oder ähnliches konstruieren.

Algorithmus 2.5:

Überprüfung der Monotonie einer Folge

Eingabe: Folgenglied a_n (`a`), Art der gesuchten Monotonie, Anzahl und maximale Größe der einzusetzenden Zufallszahlen

Ausgabe: `bool`-Wert, Art des bestandenen Tests, gegebenenfalls Anzahl der verwendeten Proben oder Gegenbeispiel

1. Überprüfe mit `checkMonDiff(a,n0,type)` das Differenzenkriterium, Entscheidung möglich: Ausgabe, sonst weiter mit (2)
2. Untersuche Folge a_n mithilfe des `solve`-Befehls auf Nullstellen, sofern keine vorhanden weiter mit (3), sonst mit (4)
3. Überprüfe mit `checkMonQuot(a,n0,type)` das Quotientenkriterium, Entscheidung möglich: Ausgabe, sonst weiter mit (4)
4. Teste mit `checkMonStart(a,n0,type)` die ersten 20 Folgenglieder auf Monotonie, falls Gegenbeispiel gefunden Ausgabe, sonst weiter mit (5)
5. Überprüfe mit `checkMonRand(a,n0,max,ntest,type)` an den `ntest` zufällig ausgewählten Stellen zwischen `n0` und `max` auf Monotonie, Ausgabe Gegenbeispiel oder "`vermutlich monoton`"

Anmerkung: Die Reihenfolge der Tests aus (1) bis (3) ist beliebig. Es mag aus didaktischer Sicht sogar sinnvoll sein, zunächst die ersten Folgenglieder (4) zu testen.

Die Tests (1) und (3) liefern einen `bool`-Wert, der `true`, `false` oder `unknown` ist.

Effizienz: Der Algorithmus liefert immer ein Ergebnis, was beim Verfeinern der algorithmischen Tests nicht gewährleistet ist. Für eine gute Performance sollte `ntest` nicht zu groß gewählt werden, mehr als 500 Tests sind in STACK nicht sinnvoll und werden teilweise vom System geblockt.

2.2.2 Funktionen zu Graphen

Maxima bietet ein Package zur Graphentheorie an [Max18]. Da Maxima allerdings als Lisp-basiertes CAS-System nicht ganz frei von Side-effects, also von Manipulation von Variablen außerhalb des Programms, bei programmierten Funktionen ist, sind nicht alle Packages in STACK integriert, da immer eine sorgfältige Analyse des Programmcodes notwendig ist, um sicherzustellen, dass eben solche Effekte ausgeschlossen werden können. Entsprechend fehlt bislang das `graphs`-Package in STACK. Dies kann sich in Zukunft natürlich ändern und entsprechend habe ich versucht, mich eng an den Definitionen des `graphs`-Package zu bewegen, um umgesetzte Aufgaben auch zukunftsfähig zu halten.

Um einen Graphen G zu definieren, benötigen wir seine Knotenmenge $V = \{v_1, \dots, v_n\}$ und seine Kantenmenge $E = \{e_1, \dots, e_m\}$. Häufig beschränken wir uns hier auf die Kantenmenge und entnehmen die Knoten dieser oder leiten sie aus der angegebenen Anzahl der Knoten ab. Die Kanten werden jeweils als Tupel aus Knoten gegeben, d.h. $[1, 3]$ ist die Kante zwischen den Knoten v_1 und v_3 . Abhängig davon, ob uns gerichtete oder ungerichtete Graphen interessieren, sortieren wir die Knoten in den Kanten nach Indexgröße, so dass bei ungerichteten Graphen immer der kleinere Index zuerst gegeben wird um in einigen Algorithmen Suchen zu beschleunigen. Dafür steht z.B. die Funktion `sortEdges(edges)` zur Verfügung (im Anhang A.9). Gewöhnlich geben wir die Kanten und Knoten als Listen an, so dass die Reihenfolge der angegebenen Knoten entscheidend ist, gelegentlich verwenden wir aber auch die Mengendefinition, um uns Maximas Funktionen von Mengen zu Nutze zu machen. Dabei helfen die Maxima-funktionen `setify(liste)` und `listify(set)`. Ein Graph wird auch durch Inzidenz- und Adjazenzmatrix eindeutig angegeben und auch das `graphs`-Package kennt diese Matrizen, deswegen existieren auch bei uns Funktionen, die diese unterschiedlichen Darstellungen ineinander überführen können wie z.B. `edgeToAdjacence(n, edges)` (A.10).

Definition 2.5:

Sei $G = (V, E)$ ein Graph mit $V = \{v_1, \dots, v_n\}$, $E = \{e_1, \dots, e_m\}$. Wir definieren die Adjazenzmatrix durch $Adj(G) : (a_{vw})_{v,w \in V}$ mit Einträgen

$$a_{vw} = \begin{cases} 1, & vw \in E, \\ 0, & \text{sonst} \end{cases}$$

und die Inzidenzmatrix $Inz(G) := (b_{ve})_{v \in V, e \in E}$ mit Einträgen

$$b_{ve} = \begin{cases} 1, & v \in e, \\ 0, & \text{sonst} \end{cases}$$

. Hierbei ist $Adj(G)$ eine $n \times n$ -Matrix, $Inz(G)$ eine $n \times m$ -Matrix.

Soll ein Graph z.B. mit `jsxGraph` dargestellt werden, wird eine Funktion benötigt, die die gewünschte Anzahl Knoten mit passenden Koordinaten erstellt. Die Koordinaten lassen sich in einer Liste explizit angeben, oder wir nutzen einen Algorithmus, der passende Koordinaten erstellt. Das kann je nach Bedarf absolut zufällig geschehen, so dass wir lediglich das Zusammenfallen von zwei Knoten verbieten (mit `CreateDiffPoints(n,k)`) oder auch mit der Forderung, dass die n Knoten etwa alle auf einem Kreis mit Radius r liegen, wofür wir `createCirclePoints(n,k)` nutzen. Bildet der Graph einen Baum, so ist eine Darstellung mit Wurzel und Blättern natürlich besonders sinnvoll. Während die beiden vorherigen Algorithmen lediglich die Koordinaten erstellen, liefert `createTreeGraph(n,1)` (Anhang A.12) neben Koordinaten auch die Kanten eines zufälligen Graphen mit n Knoten und l Ebenen ab dem Wurzelknoten v_0 . In den beiden anderen Fällen liefert `createDiffEdges(m,n,type)` m Kanten zwischen den Knoten v_0 bis v_{n-1} , wobei *type* hier angibt, ob es sich um gerichtete oder ungerichtete Kanten handelt.

2.2.3 Testen auf Planarität

Viele Aufgaben in der Graphentheorie setzen ein gewisses Geschick im Umgang mit Graphen voraus, das geübt werden muss. Dazu gehört auch das Testen auf Planarität.

Definition 2.5:

Wir nennen einen Graphen $G = (E, V)$ planar, wenn eine überschneidungsfreie Darstellung des Graphen existiert.

Satz 2.5:

- Für einen planaren Graphen $G = (E, V)$ gilt die Ungleichung $|E| \leq 2 \cdot |V| - 6$ als Folgerung aus dem **Eulerschen Polyedersatz**.
- **Satz von Kuratowski:** Ein Graph $G = (E, V)$ ist genau dann planar, wenn er weder den vollständig bipartiten Graphen $K_{3,3}$ noch den vollständigen Graphen K_5 als Minor besitzt.

Definition 2.5:

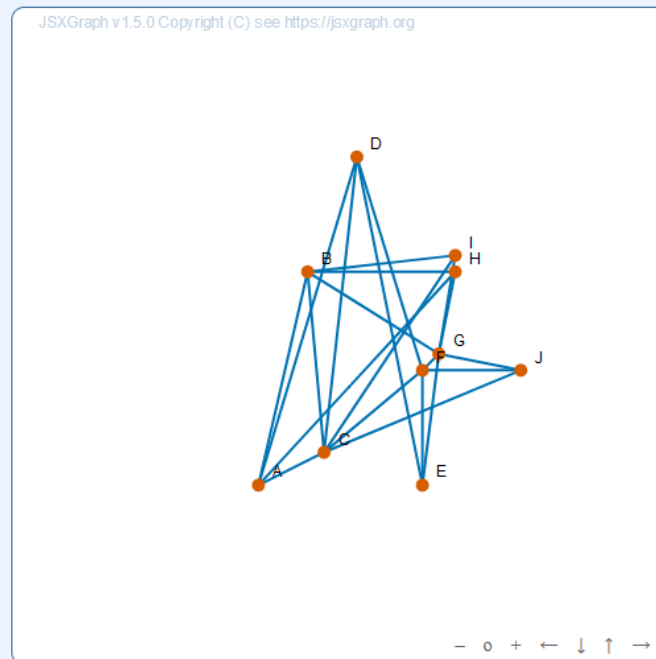
Als Minor eines Graphen $G = (E, V)$ bezeichnen wir einen Graphen, der durch Kantenkontraktionen und Entfernen von Kanten oder Knoten aus dem Graphen G resultiert.

Will man nachweisen, ob ein Graph plättbar ist, kann eine planare Darstellung dieses Graphen angegeben werden. Will man die Planarität widerlegen, so muss der Graph K_5 oder $K_{3,3}$ als Minor gefunden werden, sofern die Anzahl an Knoten und Kanten nicht ohnehin Planarität ausschließt. Hier kann eine digitale Aufgabenstellung eine wertvolle Übungsressource sein, da kein ständiges Neuzeichnen von Graphen erforderlich ist.

Das Setting der Aufgabe

Um die Wahrscheinlichkeit einen planaren Graphen zu erhalten kontrollieren zu können, werden die Knoten und Kanten nicht willkürlich randomisiert.

Consider the graph below. Decide whether it possesses a planar representation and verify your choice. If you find K_5 or $K_{3,3}$, you need to state which vertices you want to merge, so that there are only five respectively six vertices left. If you choose $K_{3,3}$, you might end up with additional edges, you don't need to worry about.



The graph above is

(Nicht beantwortet)

Please enter merged vertices as set of sets, i.e. $\{\{A,B\}, \{D,E,G\}\}$ if you want to merge A and B as well as D, E and G. If the graph is planar, please leave the box empty.

Abbildung 2.18: Studierendenansicht zum experimentellen Prüfen auf Planarität

Vielmehr geben wir eine Auswahl an Graphen vor und randomisieren lediglich die Koordinaten der Knoten. Hierzu legen wir Aufgaben für Graphen, die planar sind, sowie solche die K_5 bzw. $K_{3,3}$ enthalten an und lassen moodle bei der Testerstellung zufällig bestimmen, welche dieser drei Varianten in den Test eingesetzt wird, dabei hilft ein Auslagern der Varianten (vgl. Abschnitt 2.1.2). Um eine vergleichbare Schwierigkeit zu erhalten, setzen wir für alle Versionen 10 Knoten und 20 bis 22 Kanten. Für Maxima selber existiert ein Graphen-Package, in STACK ist dies aber nicht integriert. Entsprechend haben wir zwar Verfahren, die aus einem K_5 bzw. $K_{3,3}$ einen Graph mit

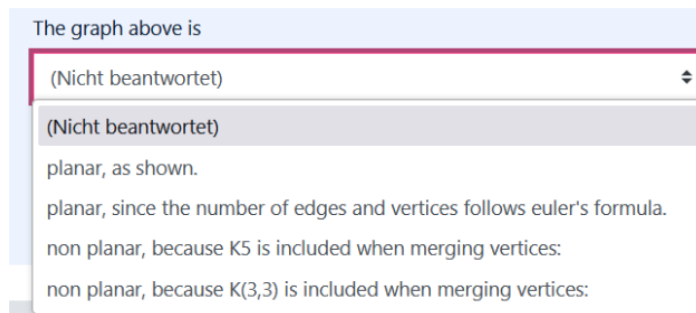


Abbildung 2.19: Antwortmöglichkeiten zur Planarität

10 Knoten generieren oder einen planaren Graphen erstellen, implementiert, trotzdem ist es sinnvoll, hier mit ausgewählten Varianten zu arbeiten. Da wir im Vorfeld die verwendeten Graphen bestimmt haben, können wir zu jedem Graph auch einen Plot erstellen, so dass, sofern nötig, eine Musterlösung angegeben werden kann.

Die Studierendenansicht der Aufgaben unterscheidet sich nur durch die Ausgabe des Graphen. Text und Eingabefelder sind für jeden Typ identisch.

Die Feedbackbäume der Aufgabe zu einem planaren Graphen unterscheiden sich von denen zu Aufgaben für Graphen, die aus K_5 und $K_{3,3}$ entstanden sind.

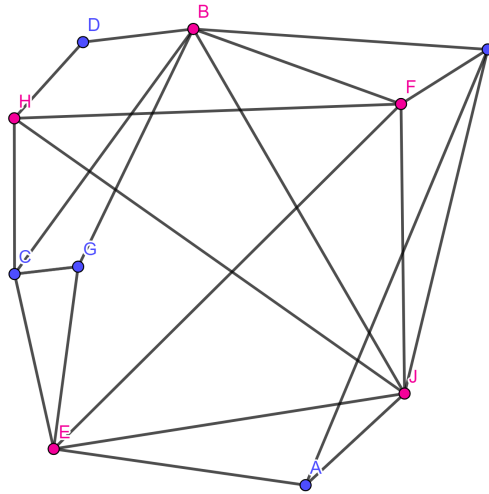


Abbildung 2.20: Graph, der K_5 enthält

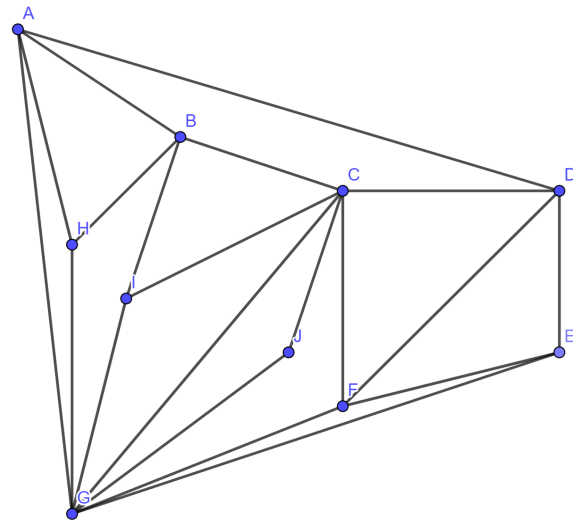


Abbildung 2.21: Planare Darstellung eines Graphen

Die Anzeige des Graphen

Wir erstellen für n Knoten zufällige Koordinaten zwischen $-k$ und k und die vorgesehenen m Kanten zwischen den Knoten. In den Fragetext fügen wir dann den Javascriptcode ein. Beginn und Ende des Blocks wird durch `[[jsxgraph width="500px" height="500px"]]` und `[/jsxgraph]]` gekennzeichnet. Hierbei wird auch gleich die Größe des Applets angegeben.

```
[[jsxgraph width="500px" height="500px"]]
var board =JXG.JSXGraph.initBoard(divid,
    {boundingbox : [-{#k#},{#k#},{#k#},-{#k#}]});

var vPos={#vertices#};
const v=[];
for(var i=0; i< vPos.length; i++)
    {v[i]=board.create('point',vPos[i])}
var e={#edges#};
for(var i=0; i< e.length; i++)
{board.create('line',[v[e[i][0]],v[e[i][1]]],
    {straightFirst:false, straightLast:false,
    strokeWidth:2})
};

stack_jxg.bind_list_of(ans2Ref, v);
[/jsxgraph]]
```

Mit `var board=JXG.JSXGraph.initBoard` wird das Koordinatensystem erstellt. Um auf zuvor in den Aufgabenvariablen erstellte Größen zurückgreifen zu können, verwenden wir `{#variable#}`. Wie man erkennen kann, ist in dem obigen Beispiel die Größe des angezeigten Koordinatensystems von der Variable k abhängig, die auch in den zufälligen Koordinaten *vertices* die Grenze darstellt. Wir erstellen in der ersten Schleife die Knoten $v[i]$ aus unseren Positionen als Objekt 'point' in unserem Koordinatensystem mit dem Befehl `v[i]=board.create('point',vPos[i])`. Die zweite Schleife liefert die Kanten als Geraden die durch einen ersten und letzten Punkt definiert werden. In den Optionen wird definiert, dass sie an diesen Punkten enden. Dadurch, dass sie als Anfangs- und Endpunkte keine Koordinaten direkt, sondern auf die entsprechenden Punkte als Objekt zugreifen, bleiben die Kanten mit dem Verschieben eines Knotens weiterhin mit den entsprechenden Knoten verknüpft. Zuletzt wird durch `stack_jxg.bind_list_of(ans2Ref, v)` die Liste der neuen Koordinaten an STACK in die Antwortbox `ans2` übergeben. Bei der Übergabe von Listen ist zu beachten, dass in JavaScript die Listen mit dem nullten Element beginnen, während bei Maxima eine Liste mit dem ersten Element beginnt. Entsprechend sind die Kanten in Maxima schon als Paar mit der Knotennummerierung von 0 bis $n - 1$ angelegt.

Die Auswertung

Handelt es sich um einen planaren Graphen, so muss eine planare Darstellung des Graphen angegeben werden. Dies kann, dank JSXGraph, einfach durch Verschieben der Knoten in der Anzeige erfolgen. Wird ein Punkt bewegt, so kann die Antwort `ans2` mit den Koordinaten aller Knoten belegt werden. Wenn zusätzlich eine Antwort zur Planarität ausgewählt wurde, kann nun der Feedbackbaum zur Planarität getriggert werden. Hierfür werden sämtliche Kanten auf Überschneidungen geprüft (vgl. hierzu Anhang A.14). Werden keine gefunden und war die Antwort korrekt, werden volle Punkte gegeben.

Ist der Graph nicht plättbar, so müssen die notwendigen Kontraktionen angegeben werden. Da häufig sowohl K_5 als auch $K_{3,3}$ als Minoren in einem Graphen enthalten sind, wird jeder nicht-planare Graph gleich behandelt. Ein Verschieben der Knoten ist in diesem Fall nicht erforderlich, erleichtert

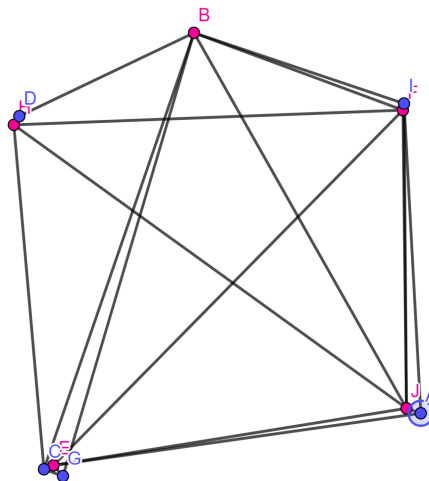


Abbildung 2.22: Möglicher K_5 -Minor

die Lösung aber. Wurde also angegeben, dass der Graph nicht planar ist, weil beispielsweise K_5 enthalten ist, so müssen die Mengen der Knoten, die kontrahiert werden, angegeben werden. Im Beispiel zu Abbildung 2.20 bzw. 2.22 also A und J oder auch C , E und G . Diese Eingaben werden zunächst überprüft, ob eventuell Mengen zusammen gehören. So könnte beispielsweise die Menge $\{C, E\}$ und $\{E, G\}$ statt der Menge $\{C, E, G\}$ gegeben sein. Diese werden dann zusammengefasst. Zusätzlich ist auch die Übersetzung von der Benennung der Knoten durch Buchstaben durch JSXGraph zu den Nummern von Maxima nötig. Ausserdem überprüfen wir, ob denn die behaupteten

Kontraktionen überhaupt möglich sind. Nun sollten, wenn die Existenz des K_5 -Minors behauptet wurde, noch fünf Knoten übrig sein, im Fall von $K_{3,3}$ noch sechs. K_5 lässt sich nun leicht überprüfen, sind noch alle 10 Kanten vorhanden, so ist der angegebene Graph ein passender Minor. Für $K_{3,3}$ ist der Fall etwas schwieriger, da wir zunächst bestimmen müssen, wie die verbleibenden Knoten in die beiden Partitionen aufgeteilt werden können. Da wir an dieser Stelle weder das Löschen der überzähligen Kanten noch die Angabe der Partitionen fordern, gleichen wir hier alle Kombinationen ab. Vorgestellt wurde dieser Aufgabentyp im Rahmen der STACK-Conference 2024 [SS24].

Kapitel 3

Einsatz der Aufgaben

3.1 Einbettung in moodle

3.1.1 Einsatz in der Lehre

STACK ist ein Aufgabentyp in moodle, um STACK-Aufgaben zu verwenden, müssen diese in einem bestimmten Rahmen, einer sogenannten Aktivität, eingebettet werden. Wir verwenden unsere Aufgaben gewöhnlich in Form von Tests. Es gibt allerdings auch andere Formate, die meist durch zusätzliche Plugins zur Verfügung gestellt werden. Zunächst will ich aber auf das Test-Format eingehen.

Aktivitätstyp Test

Ein moodle-Test wird als Aktivität auf der Kursseite angelegt und mit Fragen aus der Fragesammlung gefüllt. Aufgrund der vielen Einstellungsmöglichkeiten können Tests unterschiedlichste Einsatzzwecke erfüllen.

Unter Bewertung wird beispielsweise eingestellt, wie oft ein Test wiederholt wird, und bei mehrfachen Versuchen, welcher denn gewertet wird, oder ob ein Mittelwert gebildet wird. Unter Frageverhalten kann eingestellt werden, ob Antworten sofort ausgewertet werden sollen, oder ob ein Test erst später, nach Abgabe, bewertet wird. Eventuell erlaubt man an dieser Stelle sogar sofort einen neuen Versuch, möglicherweise mit Abzügen. In den Überprüfungseinstellungen kann dann ausgewählt werden, was genau zu welchem Zeitpunkt den Studierenden angezeigt wird.

Darüber hinaus können Deadlines gesetzt werden, Feedback zum gesamten Test in Abhängigkeit der erreichten Punktzahl gegeben werden oder auch andere Voraussetzungen gegeben werden, unter denen der Test überhaupt angezeigt wird.



Abbildung 3.3: Testinhalt mit Eingabehinweisfeld und zufälligen Fragen

Sind die generellen Einstellungen gemacht, kann der Test mit Inhalten gefüllt werden. Hierzu wählt man Fragen aus der Fragensammlung aus oder erstellt neue extra für den Test. Man darf auch eine zufällige Frage aus einer Kategorie der Fragensammlung hinzufügen, wenn man ähnliche Versionen einer Frage zur Verfügung hat, wie es z.B. bei den Aufgaben zur Planarität aus Abschnitt 2.2.3 der Fall war oder bei Aufgaben zur Stammfunktion wie in Abb. 3.4.

An dieser Stelle können die einzelnen Fragen gewichtet und die Gesamtpunktzahl des Tests festgelegt werden.

Unsere Tests

Wir verwenden Tests in erster Linie zum Training der Studierenden, so dass die Tests mit bis zu zehn verschiedenen Aufgaben ab dem Zeitpunkt, zu dem die Themen in der Vorlesung behandelt wurden, unbegrenzt den Studierenden zur Verfügung stehen. Für gewöhnlich verwenden wir hier die spätere Auswertung, so dass Feedback zu den einzelnen Aufgaben erst nach Abgabe erfolgt. Bei einem erneuten Versuch erhalten die Studierenden dann eine neue randomisierte Aufgabe. Ein anderer von uns häufig verwendeter Modus ist der Einsatz einzelner Aufgaben als Hausaufgabe. In diesem Fall werden von uns nur 2-3 Versuche zugelassen. Außerdem wird eine fester Endzeitpunkt gesetzt, bis zu dem abgegeben werden kann.

Andere Formate

Neben Tests gibt es weitere Formate, in denen STACK eingesetzt werden kann. Sinnvoll kann dies z.B. im Rahmen des Plugins Question-Practice, bei dem Fragen einer Kategorie wie Karteikarten abgefragt werden, geschehen. Möglich ist außerdem, sie in das StudentQuiz Plugin einzupflegen, bei dem Studierende selbstständig Aufgaben unterschiedlichen Fragetyps entwickeln und teilen, diese dann trainieren und bewerten können. In diesen beiden Fällen leidet allerdings die Struktur der Fragensammlung.

Information

Frage markieren
Frage bearbeiten

Bitte beachten Sie:

Arcus-Sinus: $\text{asin}(x)$
Arcus-Cosinus: $\text{acos}(x)$
Arcus-Tangens: $\text{atan}(x)$
ln: $\log(x)$

Frage 1

Unvollständig
Erreichbare Punkte: 1,00
Frage markieren
Frage bearbeiten

Geben Sie eine nicht konstante Funktion $f: [6, 11] \rightarrow \mathbb{R}$ an, so dass die Fläche unter der Kurve von f 13 beträgt.

$f(x) =$

Prüfen

Frage 2

Unvollständig
Erreichbare Punkte: 1,00
Frage markieren
Frage bearbeiten

Berechnen Sie folgendes Integral:

$$\int \frac{1}{(x^2 + 1) \cdot \text{atan}(x)} dx$$

Eine Stammfunktion lautet: $F =$

Prüfen

Abbildung 3.4: Fertiger Test

3.1.2 Bestimmung der Schwierigkeit anhand einzelner Aufgaben

Werden STACK-Aufgaben zur Bewertung der Studienleistung herangezogen, stellt das die Randomisierung vor gewisse Herausforderungen. Ein großer Vorteil der Randomisierung ist es, den Studierenden unterschiedliche Varianten einer Aufgabe zuzuweisen, so dass Lösungen der Aufgabe nicht verbreitet werden können. Andererseits muss aber sicher gestellt werden, dass der Schwierigkeitsgrad aller Varianten ähnlich ist. STACK liefert hier Dozenten bei der Durchführung von Fragetests direkt die Möglichkeit, die Varianten selbst auszuwählen, die eingesetzt werden. (Vergleiche Kapitel 2.1.2).

Betrachtet man nach Testdurchführung allerdings die Antworten der Studierenden, stimmen diese aber nicht notwendigerweise mit den Erwartungen der Lehrenden überein. So gelten z.B. eine Matrix, die viele Nullen enthält, allgemein als unproblematisch bei Rechnungen oder rein reelle/imaginäre Zahlen bei Konversionen in Polarkoordinaten als einfache Spezialfälle. Diese Vereinfachung scheint aber in einigen Fällen auch zu Verwirrung zu führen, weswegen eine leichte Klassifizierung (z.B. anhand des relativen Anteils der Nulleinträge) nicht möglich erscheint.

Ich möchte genauer auf eine Aufgabe zur Umrechnung komplexer Zahlen eingehen (Abb. 3.5): Im Wintersemester 2020 und 2021 haben insgesamt 123 Studentinnen und Studenten diese Aufgabe bearbeitet. 19 davon hatten eine einfache Variante erhalten. 13 davon lösten diese korrekt, was einem Anteil von 68% entspricht. Betrachtet man den Anteil der Studierenden, die allgemein eine korrekte Antwort gaben, waren dies 67%. In diesen beiden Jahren lief diese Aufgabe als freiwilliges Übungsangebot. Im Wintersemester 2018 war diese Aufgabe in den freiwilligen Übungen und anschließend als verpflichtende Abgabe zu bearbeiten. Freiwillig wurden 155 Versuche abgegeben. Richtige Ergebnisse mit Argument modulo 2π lagen in 51% der Fälle vor, wird auf das korrekte Intervall Wert gelegt, dann sogar nur 38%. Einfache Aufgaben wurden in 25 Fällen gestellt, davon waren 14 Antworten korrekt, also etwa 54%. Bei diesen Aufgaben war das Argument, sofern richtig, immer im korrekten Intervall. Betrachtet man die Ergebnisse der Hausaufgabe, so wurden 229 Aufgaben bearbeitet. Ein modulo 2π korrektes Ergebnis lag in 63% der Abgaben vor. Einfache Aufgaben wurden in 60 Fällen gestellt, hier lag die korrekte Quote bei 54%. Im darauffolgenden Jahr wurde wieder Hausaufgabe und Übung zur Verfügung gestellt, allerdings gab es bei der Randomisierung Unregelmäßigkeiten und es wurden bei den Hausaufgaben nur vier Varianten erstellt, von denen eine eine einfache war. Bei den Hausaufgaben durften 3 Versuche durchgeführt werden und so sind von 546 Abgaben 55% korrekt, von der 137 Mal auftretenden einfachen Variante dann

Geben Sie die komplexe Zahl in Polarkoordinaten an.

(Beachten Sie, dass das Argument im zulässigen Bereich $(-\pi, \pi]$ liegen muss.)

$-6 =$ $\cdot e^{i$

Abbildung 3.5: Studierendenansicht einer als leicht eingestuften Aufgabe. Eingetragen werden Radius und Winkel

60%. Die freiwillige Übung wurde nur 67 mal aufgerufen mit insgesamt 71% korrekten Ergebnissen, hier gab es nur acht Tests mit einfachen Versionen, davon waren sechs korrekt.

Jahr	Modus	bearbeitete Aufgaben	davon richtig	besonders einfache Varianten	davon richtig
2020/21	freiwillig	123	67%	19	68%
2018	Hausaufgabe	229	63%	60	54%
2019	Übung	67	71%	8	75%
2019	Hausaufgabe (4 Varianten)	546	55%	137	60%

Tabelle 3.1: Umrechnung komplexer Zahlen zu Abb. 3.5

Insgesamt lässt sich also festhalten, dass von uns im Vorfeld als einfach eingestufte Varianten den Studierenden anscheinend keine großen Vorteile bieten. Beobachtete Fehler waren beispielsweise, dass der Betrag bei Aufgabenstellungen wie in Abb. 3.5 negativ angegeben wurde und das Argument dafür mit 0 angegeben worden ist. Ansonsten traten andere Eingabefehler auf, wie z.B. dass die Eingabe von π vergessen worden ist. Hier ließe sich darüber diskutieren, ob diese vereinfachte Eingabe in zwei Felder bei einer einfachen Variante nicht sogar zu Irreführungen führt.

Eine andere Aufgabe, auf die ich eingehen möchte, ist die in Kapitel 2.1.2 besprochene Aufgabe zur Berechnung eines Eigenwerts und seiner Vielfachheit. Hier überraschen die Ergebnisse ebenfalls. Als Hausaufgabe wurde diese Aufgabe im Wintersemester 2020 insgesamt 240fach ausgegeben. Eine Bearbeitung war pro Studierenden bis zu drei mal möglich. Es gab insgesamt 20 Varianten, die im Vorfeld als unterschiedlich schwer eingestuft wurden. Beispielsweise wurde die Variante aus Abb. 3.6 von uns als eher einfach eingestuft. Die Matrix

$$A_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -5 & -6 \\ 0 & 3 & 4 \end{pmatrix}$$

besitzt $\lambda = 1$ als doppelten und $\lambda = -2$ als einfachen Eigenwert. Da hier nach dem kleinsten Eigenwert gefragt war, ist algebraische und geometrische Vielfachheit also schnell ablesbar. Nur einmal wurde $\lambda = -2$ als der kleinste Eigenwert angegeben, dann aber eine falsche Vielfachheit dazu bestimmt. Ansonsten wurde die 1 als kleinster Eigenwert gesetzt oder ein falscher Wert berechnet. In der Auswertung wurde abgebrochen, wenn nicht der gesuchte Eigenwert angegeben war, aber auch nachträglich betrachtet, sind zum Eigenwert 1 in keinem Fall algebraische und geometrische Vielfachheit richtig angegeben.

Bei einer anderen eingesetzten Variante, die 14 mal eingesetzt wurde, war die Matrix

$$A_2 = \begin{pmatrix} 33 & 78 & 90 \\ -26 & -62 & -72 \\ 10 & 24 & 28 \end{pmatrix}$$

gegeben. Diese Matrix besitzt die Eigenwerte $\lambda_1 = 0$, $\lambda_2 = 1$ und $\lambda_3 = -2$. In 50% der Fälle wurde diese Variante korrekt bearbeitet. Nur zweimal wurde die Aufgabe nicht bearbeitet, in den übrigen Fällen konnten Fehler nicht nachvollzogen werden. Hat man bei dieser Matrix die Eigenwerte gefunden, ist die Bestimmung der Vielfachheiten dafür ohne weiteren Rechenaufwand möglich, auch wenn in den Abgaben durchaus größere geometrische als algebraische Vielfachheiten auftraten.

Varianten, in denen die algebraische Vielfachheit größer als eins war, traten auch auf. Diese liefen in manchen Fällen gut, in anderen nicht. Zu der Matrix

$$A_3 = \begin{pmatrix} 1 & 1 & -7 \\ 0 & 1 & -3 \\ 0 & 0 & -2 \end{pmatrix}$$

wurde der größte Eigenwert λ_e gesucht. Zu $\lambda_e = 2$ gehören die Vielfachheiten $\mu_{alg}(\lambda_e) = 2$ und $\mu_{geo}(\lambda_e) = 1$. Von den 13 Abgaben waren sechs komplett richtig, bei vier weiteren Abgaben konnte nur der Punkt für den korrekten Eigenwert gegeben werden.

Anders sieht es bei der Matrix

$$A_4 = \begin{pmatrix} -2 & 1 & 0 \\ -4 & 2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$$

aus, bei der der größte Eigenwert $\lambda_e = 0$ gesucht wurde. Wie in A_3 ist dies ein doppelter Eigenwert und die Variante wurde als vergleichbar schwierig eingestuft. Allerdings waren von 17 Abgaben hier nur drei Antworten korrekt.

Zusammenfassend lässt sich also auch hier die Schwierigkeit der Aufgabe nicht durch einen Blick auf Matrix und Lösung erkennen.

Gegeben Sei die Matrix $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -5 & -6 \\ 0 & 3 & 4 \end{pmatrix}$.

Geben Sie den kleinsten Eigenwert λ_e und seine algebraische und geometrische Vielfachheit $\mu_{alg}(\lambda_e)$ bzw. $\mu_{geo}(\lambda_e)$ an.

$\lambda_e =$

$\mu_{alg}(\lambda_e) =$

$\mu_{geo}(\lambda_e) =$

Abbildung 3.6: Studierendenansicht einer als leicht eingestuftem Aufgabe

3.2 Rückmeldung der Studierenden

Ich möchte zwei Arten des Feedbacks von unseren Studierenden erwähnen. Zum einen wurden über zwei Semester Tests durch STACKrate ergänzt, so dass die Studierenden um direktes Feedback während der Bearbeitung der Aufgaben gebeten wurden. Zum anderen gab es immer auch Studierende, die uns direkt angesprochen oder angeschrieben haben. Auch hierauf möchte ich gerne eingehen.

3.2.1 STACKrate

Mit STACKrate können STACK-Aufgaben in moodle direkt im Test von Studierenden bewertet werden. Studierende sehen beim Prüfen der Eingabe ein Feld, auf dem sie Sterne in unterschiedlichen Kategorien vergeben können. Zusätzlich können Kommentare zu den Aufgaben abgegeben werden. Wir verwendeten hier die Version 0.1.

STACKrate wurde an der Ruhr Universität Bochum von Jonas Lache und Daniel Meissner entwickelt und ist ein JavaScript Modul, das direkt an den Feedbacktext einer STACK-Aufgabe eingebunden wird [LM22]. So kann es in Tests eingesetzt werden, die eine Überprüfung der Eingabe während des Tests erlauben. Da wir davon ausgegangen sind, dass bei Tests mit mehreren Aufgaben nicht jede Aufgabe bewertet wird, haben wir immer zwei bis vier Aufgaben auf eine Seite des Tests gelegt und am Ende der Seite dann sämtliche Fragen der Seite zusammen bewerten lassen. Die Antworten lassen sich dann bei den Ergebnissen des Test finden. Einen solchen Test zeigen wir im Anhang in (D).

Da die Tests während der Bearbeitung überprüft werden müssen, können diese Tests nicht zu Hausaufgabenzwecken herangezogen werden und die Bearbeitung ist bei uns freiwillig gewesen. Dies hat leider zur Folge, dass nur wenige dieser Tests bearbeitet wurden und die STACKrate-Fragen auch nur in sehr geringen Maß beantwortet wurden. So ist der Test zu Mengenoperationen und Aussagen (Anhang D) zum Auswertungszeitpunkt nur von 79 Studierenden bearbeitet worden. Feedback kam allerdings nur von wenigen.

berücksichtigte Tests	Feedback Seite 1	Feedback Seite 2	Feedback Seite 3
79	28	14	13

Zuerst wurde das Niveau von sehr leicht (1) über angemessen (3) bis sehr schwer (5) abgefragt.

Niveau der Fragen	Anzahl Seite 1	Anzahl Seite 2	Anzahl Seite 3
1	16	3	0
2	5	5	1
3	0	5	6
4	0	1	6
5	0	0	0

Zur Bedienfreundlichkeit wurden Angaben von sehr schlecht (1) bis sehr gut (5) abgefragt.

Usability	Anzahl Seite 1	Anzahl Seite 2	Anzahl Seite 3
1	1	0	0
2	1	0	2
3	3	1	0
4	6	4	2
5	17	9	9

Die Bearbeitung wurde von langweilig (1) bis interessant (5) bewertet.

Aufgabenbearbeitung	Anzahl Seite 1	Anzahl Seite 2	Anzahl Seite 3
1	0	1	0
2	2	2	0
3	11	3	3
4	10	5	7
5	5	3	4

Das Feedback wurde von gar nicht hilfreich (1) bis sehr hilfreich (5) angegeben.

Feedback	Anzahl Seite 1	Anzahl Seite 2	Anzahl Seite 3
1	0	0	2
2	3	1	0
3	8	3	2
4	3	7	3
5	13	3	6

In den Bemerkungen wurde hier einmal erwähnt, dass es gut gewesen wäre, wenn klargestellt würde, wie die leere Menge eingegeben werden sollte. Die andere Bemerkung war, dass bei einem Studierenden die Fragen zur Aussagenlogik tatsächlich identisch waren.

Rückblickend ist anzumerken, dass die freiwillige Abfrage keinen ausrei-

chenden Rücklauf hatte, um Schlüsse zu den Aufgaben zu ziehen.

3.2.2 Direkte Rückmeldungen

Häufiger wurden wir direkt von Studierenden kontaktiert, wenn Aufgaben nicht wie erwartet bewertet wurden. Hierbei handelte es sich meist um verpflichtende Hausaufgaben. Da Zwischenschritte in Rechnungen selten berücksichtigt werden konnten, wurden auch kleinste Rechenfehler als vollständig falsch klassifiziert. Die Ausarbeitungen, die uns teilweise vorgelegt wurden, wären bei einer händischen Korrektur nur minimal, wenn überhaupt abgewertet worden. Dies stuften Studierende als demotivierend ein. Auch geringe systematische Fehler, die wir nicht vorhergesehen hatten, wurden an uns zurückgemeldet. So wurde in einer Aufgabe zur hesseschen Normalform teilweise der Normalenvektor in die entgegengesetzte Richtung gesetzt. Damit hätten Studierende einen negativen Abstand zum Ursprung ablesen können. Die Missachtung der Konvention haben wir noch erwartet und im Feedback berücksichtigt. Dass es aber auch Studierende gab, die hier einen positiven Abstand durch Betragsbildung erzwingen, hatten wir nicht vorhergesehen. Gerade bei langen Termen, bei denen Eingabefehler zu erwarten sind, kamen Studierende auch auf uns zu und baten um eine Korrektur der Punkte, der wir teilweise auch nachkamen um die Studierenden nicht durch technische Schwierigkeiten zu demotivieren.

STACK Frage Dashboard

Liebe Studierende,

wir entwickeln die digitalen Aufgaben kontinuierlich weiter und wollen sie stets verbessern und Ihren Wünschen entsprechend anpassen. Wir sind daher sehr an Ihren Rückmeldungen interessiert und freuen uns über Ihre ehrlichen Bewertungen und Anregungen.

Um die Aufgaben, die Sie auf dieser Seite bearbeitet haben, zu evaluieren, klicken Sie bitte zunächst auf *Prüfen* unter dem Eingabefeld (bitte verändern Sie nichts im Eingabefeld). Anschließend erscheinen die Fragen zur Bewertung.

Vielen Dank für Ihre Mitarbeit!

Bewertung

Aufgabenbewertung

Aufgabenschwierigkeit
Wie haben Sie das Niveau der Aufgabe empfunden?

☆☆☆☆☆
(sehr leicht -- angemessen -- sehr schwer)

Usability
Wie gut sind Sie mit der Eingabe zurecht gekommen?

☆☆☆☆☆
(sehr schlecht -- sehr gut)

Aufgabenbearbeitung
Wie haben Sie den Bearbeitungsprozess empfunden?

☆☆☆☆☆
(langweilig -- interessant)

Aufgabenfeedback
Für wie hilfreich haben Sie das Feedback empfunden, welches Sie während der Bearbeitung der Aufgabe erhalten haben?

☆☆☆☆☆
(gar nicht hilfreich -- sehr hilfreich)

Bemerkungen:

Abbildung 3.7

Kapitel 4

Fazit und Ausblick

4.1 Grenzen der Nutzung

STACK bietet äußerst hilfreiche Tools, um Aufgaben zu erstellen. Im Rahmen meiner Arbeit interessierte mich hauptsächlich die Erstellung von Aufgabenscharen, bei denen einige Teile der Aufgaben randomisierte Komponenten enthielten, so dass mehrere Varianten einer Aufgabe auf einmal erstellt werden konnten. Dies klammert natürlich einige interessante Features von STACK aus, die mit meinen Methoden nicht umgesetzt werden können. Zudem ist es auch einfach nicht immer praktikabel alle Aspekte der mathematischen Ausbildung zu automatisieren.

Beweisverfahren

Ein ganz wesentlicher Bestandteil der mathematischen Ausbildung ist das Beweisen von Aussagen. Dies sehen auch die Macher von STACK und entwickeln STACK in diese Richtung weiter. So ist in der im Juli 2024 erschienenen Version ein neuer Baustein hinzugekommen, die Parson-Blocks. Hierbei werden Bausteine eines Beweises gegeben, die dann in die richtige Reihenfolge gebracht werden müssen, oft auch ergänzt durch überflüssige, teilweise falsche, Bausteine. Abgesehen von einer zufälligen Reihenfolge ist hier aber keine Randomisierung sinnvoll. Konstruktives Feedback ist natürlich möglich, allerdings unter großem Aufwand.

Eine Überprüfung von frei eingegebenen Beweisen wäre natürlich noch zielführender, dafür würde allerdings ein entsprechend trainiertes Sprachmodul benötigt.

Graphen

Nicht immer ist eine Randomisierung sinnvoll. So liefert z.B. das Programm `createTreeGraph(m,n)` oder `addToK5(n,k)` alle möglichen Graphen, die aber nicht notwendigerweise die beim Erstellen der Aufgabe angedachte Form besitzen. Hier ist mindestens eine sorgfällige Nachkontrolle der erstellten Aufgaben nötig. Natürlich lassen sich diese Algorithmen auch einfach zur Erstellung von Kandidaten nutzen, die man wie in Section 2.1.2 erwähnt, in einer externen Datei ablegt.

Ähnliches haben wir bei den Ableitungsaufgaben zur Randomisierung festgestellt, auch hier führt es zu Problemen, wenn man zuviele Möglichkeiten in einem Algorithmus unterbringen möchte. Dies führt meist dazu, dass die erstellten Aufgaben kuratiert werden wollen. In diesen Fällen bietet es sich an, verschiedenen Programme für die unterschiedlichen Typen zu schreiben und dann einen zufälligen Typ auszuwählen, ähnlich wie es in Abschnitt 2.2.3 zur Planarität beschrieben ist.

4.1.1 Einsatz der Aufgaben

Wie diskutiert, ist es kaum möglich, vorab eine Einschätzung über die Schwierigkeit einer Aufgabe zu geben, die sich im Nachgang bestätigen lässt. Dadurch ist es etwas problematisch, randomisierte Aufgaben zu Bewertungszwecken heranzuziehen, da die Frage, ob eine Vergleichbarkeit existiert, eher subjektiv beantwortet werden muss.

Durch den Einsatz von technischen Mitteln seitens der Studierenden sind Rechenaufgaben schnell gelöst, ein Einsatz in einer Klausur könnte nur in einer überwachten Umgebung stattfinden. In diesem Fall wäre eine Randomisierung nicht notwendig. Allerdings lassen sich bei randomisierten Aufgaben schnell immer neue fertige Aufgaben auswählen.

Insgesamt halte ich das Randomisieren mit STACK für eine hervorragende Möglichkeit einen brauchbaren Aufgabenpool zu Übungszwecken bereitzustellen, Aufgaben zur Bepunktung sollten allerdings relativ offen gestaltet werden.

4.2 Ausblick

Die Entwicklung von STACK wird weiter fortgesetzt und entsprechend stehen auch in Zukunft neue Möglichkeiten zur Verfügung. Beispielsweise ist angekündigt, die `let()`-Option aus Maxima zu integrieren, was vermutlich die Auswertungen in Abschnitt 2.2.1 stark verbessern wird.

Die Integration von Geogebra ist deutlich verbessert worden, hier eröffnen sich neue Möglichkeiten, grafische Zusammenhänge auch im dreidimensionalen Raum darzustellen, was momentan auch an der Universität Kassel umgesetzt wird.

Der Plan, die Graphenbibliothek aus Maxima auszubauen, existiert ebenfalls, so können weitere Veranstaltungen mit Aufgaben in STACK betreut werden. Was letztendlich Sprachmodule an Möglichkeiten liefern, lässt sich momentan nur erahnen. Sobald aber textuelle Antworten auf Inhalt und nicht nur auf identische Worte untersucht werden können, werden sich bei der Aufgabenauswertung ganz neue Optionen auftun.

Literaturverzeichnis

- [Aba] ABACUS: *Projektseite*. <https://abacus.aalto.fi/>,
- [Con] CONFERENCES, STACK: *Projektseite*. <https://stack-assessment.org/Conference/>,
- [Dig21] DIGITAL, AK M.: *AK-Mathe-Digital*. <https://ak-mathe-digital.de/index.html>, 2021
- [DJMWDD24] DOZ, Daniel ; JUMA, Zevick O. ; MOTOGNON WASTALAS D. DOGBALOU, Danilo L.: *Integrating STACK at the University of Trieste, Italy*. https://stack-assessment.org/CaseStudies/2024/Integrating_STACK_at_the_University_of_Trieste_Italy/, 2024
- [Fis14] FISCHER, Pascal R.: *Mathematische Vorkurse im Blended-Learning-Format*. Springer Spektrum Wiesbaden, 2014
- [FY48] FISCHER, Ronald A. ; YATES, Frank: *Statistical tables for biological, agricultural and medical research*. Oliver and Boyd, London, 1948
- [GVW10] GERHAUSER, Michael ; VALENTIN, Bianca ; WASSERMANN, Alfred: JSXGraph–Dynamic Mathematics with JavaScript. In: *International Journal for Technology in Mathematics Education* 17 (2010), Nr. 4, 211–215. <https://www.learntechlib.org/p/109512>. – ISSN 1744–2710
- [Haa14] HAAGER, Wilhelm: *Computeralgebra mit Maxima*. Hanser, 2014
- [HPW09] HARRISON, Martin ; PIDCOCK, Dave ; WARD, Joe: Using Technology to Help Engineers Learn Mathematics. In: *The Electronic Journal of Mathematics and Technology, Volume 3 Number 2* (2009)

- [JAOO22] JUMA, Zevick O. ; AYERE, Mildred A. ; OYENGO, Michael O. ; OSANG, Georg: Evaluating Engagement and Learning Based on a Student Categorization using STACK, Exam Data, Key Informant Interviews, and Focus Group Discussions. In: *International Journal of Emerging Technologies in Learning (iJET)* 17 (2022), Dec., Nr. 23, S. pp. 103–114. <http://dx.doi.org/10.3991/ijet.v17i23.36619>. – DOI 10.3991/ijet.v17i23.36619
- [JC DL24] JUMA, Zevick O. ; CHELLERI, Elisabetta ; DOZ, Daniel ; LEWANSKI, Danilo: Students attitudes on STACK content: A pilot study at the university of Trieste, Italy . In: *Journal of Advanced Sciences and Mathematics Education* 4 (2024), Nr. 1
- [Kal16a] KALLWEIT, Michael: CAS-unterstütztes Assessment von Mathematik. In: *Computeralgebra-Rundbrief, Ausgabe 59* (2016), S. 22–24
- [Kal16b] KALLWEIT, Michael: Der Computer als Tutor - technikbasierte Diagnostik mit Freitext- aufgaben. In: *Institut für Mathematik und Informatik Heidelberg (Hrsg.), Beiträge zum Mathematikunterricht 2016. WTM-Verlag Münster.* (2016), S. 22–24
- [Knu97] KNUTH, Donald E.: *The art of computer programming*. Addison-Wesley, Reading, Mass, 1997
- [LM22] LACHE, Jonas A. ; MEISSNER, Daniel: How to STACKrate: The Development of an Intuitive Evaluation Tool. In: *International Journal of Emerging Technologies in Learning (iJET)* 17 (2022), Dezember, Nr. 23, 7–14. <http://dx.doi.org/10.3991/ijet.v17i23.36527>. – DOI 10.3991/ijet.v17i23.36527
- [Max18] MAXIMA: *Maxima Manual 63 graphs* . http://maxima.sourceforge.net/docs/manual/maxima_285.html, 2018
- [MJO24] MIDDEKE, Johannes ; JEFFREY, David J. ; OLAGUNJU, Aishat: Applications of Rational Orthogonal Matrices. In: *Mathematics in Computer Science* 18 (2024), Nr. 14

- [Net] NETWORK, STACK P.: *Projektseite*. https://stack-assessment.org/Communities/Professional_network/,
- [NFE⁺25] NEUGEBAUER, Malte ; FALK, Sabrina ; ERLEBACH, Ralf ; HIGUCHI, Saburo ; NAKAMURA, Yasuyuki: Robust and Reliable Automated Feedback Using Tree Edit Distance for Solving Open Response Mathematical Questions INSTICC, SciTePress, 2025. – ISBN 978–989–758–746–7, S. 618–626
- [OPSS21] OYENGO, Michael O. ; PARSONS, Danny ; STERN, David ; SANGWIN, Christopher: *Providing student feedback through electronic assessment for linear algebra at Maseno University, Kenya* . <https://zenodo.org/records/5035969>, Juni 2021
- [Pre22] PREMUŠ, Mojca: *Analysis of moodle quizzes compared to grades*. <https://eams.ncl.ac.uk/sessions/2022/analysis-of-moodle-quizzes-compared-to-grades/>, 2022
- [Ras16] RASILA, Antti: E-Assessment Material Bank Abacus, 2016
- [RV17] RITTER, Steffan ; VASKO, Mikko: Beispiele für Verständnisaufgaben zur Ingenieurmathematik in digitaler Form. In: *Technische Universität Dortmund (Hrsg.), Beiträge zum Mathematikunterricht 2017*. WTM-Verlag Münster. (2017), S. 1197–1200
- [San16] SANGWIN, Chris: *ComputerAided Assessments of Mathematics*. Oxford University Press, 2016
- [SCMMBZ25] STE CROIX, Louisa de ; MACKENZIE, Malakai ; MAUDSLEY-BARTON, Hayden ; ZERVA, Konstantina: *Employ.Ed Internship - Translating the HELM workbooks to STACK Continued*. <https://stack-assessment.org/CaseStudies/2025/EmployEd%20Internship%20Translating%20HELM%20workbooks%20STACK/>, 2025
- [SD22] STACK-DEVELOPMENT: *Inclusions*. <https://docs.stack-assessment.org/en/Authoring/Inclusions/>, 2022

- [SD24] STACK-DEVELOPMENT: *Advanced JSXGraph*. https://docs.stack-assessment.org/en/Specialist_tools/JSXGraph/Advanced_JSXGraph, 2024
- [SD25] STACK-DOCS: *HELM-Materials*. https://docs.stack-assessment.org/en/STACK_question_admin/Library/HELM/#helm-workbooks-list, 2025
- [SS23] SCHMIDTPOTT-SCHULZ, Hendrikje: *Testing on Inequality*. <https://eams.ncl.ac.uk/sessions/2023/testing-on-inequality/>, 2023
- [SS24] SCHMIDTPOTT-SCHULZ, Hendrikje: *Session recording: Planarity of Graphs with jsxGraph*. <https://www.youtube.com/watch?v=1hDgF4u08B4>, 2024
- [Ste03] STEELE, John: *Setting linear algebra problems*. <https://web.maths.unsw.edu.au/~jds/Papers/linalg.pdf>, Januar 2003
- [THN25] TAKADA, Tomoki ; HIGUCHI, Saburo ; NAKAMURA, Yasuyuki: Visualization of Solution Processes to Reach the Correct Answer in Online Math Tests. In: *Proceedings of the 18th International Conference on Educational Data Mining*, International Educational Data Mining Society, Juli 2025, 635–639
- [Vas18] VASKO, Mikko: Interaktive grafische Aufgaben mit STACK und JSXGraph. In: *Technische Universität Dortmund (Hrsg.), Beiträge zum Mathematikunterricht 2018*. WTM-Verlag Münster. (2018), S. 1855–1858
- [WCH15] WLADIS, Claire ; CONWAY, Katherine M. ; HACHEY, Alyse C.: The Online STEM Classroom—Who Succeeds? An Exploration of the Impact of Ethnicity, Gender, and Non-traditional Student Characteristics in the Community College Context. In: *Community College Review* 43 (2015), Nr. 2, S. 142–164. <http://dx.doi.org/10.1177/0091552115571729>. – DOI 10.1177/0091552115571729
- [WWA24] WASSERMANN, Alfred ; WIKI-AUTHORS: *JSXGraph Documentation*. <https://jsxgraph.uni-bayreuth.de/wiki/index.php/Documentation>, 2024

- [ZB23] ZEVICK, Juma ; BORIO, Santiago: *Transforming Mathematics Education in Africa: The Impact of the INNODEMS STACK Internship Programme*. https://stack-assessment.org/CaseStudies/2023/INNODEMS_Internship, 2023

Anhang A

Codefragmente

Hier findet sich eine Sammlung interessanter Codefragmente. Diese Fragmente sind auch unter https://www.mathematik.uni-kassel.de/~heschmpo/test_include.txt verfügbar, so dass z.B. mit einer STACK-Frage auf sie zugegriffen werden kann.

A.1 Matrizen

```
make_mat_zero(n,z,max):=block([m,i,line,test:false,t],
  if z>n-1 then m[i,j]:0 else
  while test=false do block(
    line:makelist(if i<z+1 then 0 else 1,i,n),
    for i:1 thru n do block([li,j],
      li:random_permutation(line),
      for j:1 thru n do if li[j]=1 then m[i,j]:
        rand_with_prohib(-max,max,[max+1]) else m[i,j]:0),
    test:true,
    for j:1 thru n do block(t:0,
      for i:1 thru n do if not(m[i,j]=0) then t:t+1,
      if t=0 then test:false)),
  genmatrix(m,n,n));
```

Codeblock A.1. $n \times n$ -Matrix mit mindestens z Nulleinträgen pro Spalte

```

perMat(n):=block([m,A,l:[],M,line:[1]],
  for i:1 thru n do line:endcons(0,line),
  A:transpose(matrix(line)),line[1]:0,
  for i:2 thru n do block(
    line[i]:1,
    A:addcol(A,transpose(line)),
    line[i]:0 ),
  for i:1 thru n do l:cons(A[i],l),
  M:random_permutation(l) ,
  A:transpose(matrix(first(M))),
  for i:2 thru n do A:addcol(A,M[i]),
  transpose(A) );

```

Codeblock A.2. Erstellt eine $n \times n$ -Matrix zum zufälligen Zeilen- bzw. Spaltentausch¹

```

perMatRank(n,r):=block([m,A,l:[],line:[1]],
  for i:2 thru n do line:endcons(0,line),
  A:transpose(matrix(line)),
  line[1]:0,
  for i:2 thru n do block(
    line[i]:1,
    A:addcol(A,transpose(line)),
    line[i]:0 ),
  for i:1 thru n do l:cons(A[i],l),
  l:rand_selection(l,r),
  for i:r+1 thru n do block([a:rand(r)+1,b],
    b:rand_with_prohib(1,r,[a]) ,
    l:endcons(l[a]+l[b],l)),
  l:random_permutation(l),
  A:transpose(matrix(first(l))),
  for i:2 thru n do A:addcol(A,l[i]),
  transpose(A) );

```

Codeblock A.3. Erstellt eine $n \times n$ -Matrix, die mit einer regulären $n \times n$ -Matrix A multipliziert werden kann um eine Matrix vom Rang r zu erzeugen.

¹In diesem Codeblock werden Maxima-Array vermieden, deswegen umständlichere Programmierung über die Zeilen der Matrix als Listenelement. Das Transponieren ist in früheren STACK-Versionen erforderlich gewesen, da der addrow-Befehl in STACK leider neu besetzt wurde.

```

make_intInv_n(n,max):=block([L,R,i,j],
L[i,j]:=0,
R[i,j]:=0,
for i:1 thru n do
block(L[i,i]:rand_with_prohib(-1,1,[0]),
R[i,i]:rand_with_prohib(-1,1,[0])),
for i:1 thru n do
for j:1 thru i do
block(R[i+1,j]:rand_with_step(-max,max,1),
L[n-i,n+1-j]:rand_with_step(-max,max,1)
),
L:genmatrix(L,n,n),
R:genmatrix(R,n,n),
L.R);

```

Codeblock A.4. Erstellt eine $n \times n$ -Matrix mit ganzzahligen Inversen zu Algorithmus 2.2.²

```

instGram(n,m,k):=block([a:makelist(rand_with_prohib(-
k,k,[0]),n),A],
A:transpose(matrix(a)),
for i:2 thru m do block([line:makelist(0,n)],
line[i]:1,
A:addcol(A,line)),
A);

```

Codeblock A.5. Erstellt eine $m \times n$ - Matrix, deren erste Zeile Einträge zwischen $-k$ und k ohne Null enthält und deren weitere Zeilen aus Einheitsvektoren bestehen

²Die Deklaration von L und R über ein Array wie in diesem Codeblock funktioniert in den bislang genutzten STACK-Versionen gut, bei einer Maxima-Installation kann es aber zu Konflikten kommen, da Arrays teilweise global agieren obwohl sie in **blocks**-Umgebungen definiert werden. Dies kann vermieden werden, indem die Matrix analog zu A.2 generiert wird.

```

gramN(b):=block([i,j,n:length(row(b,1)),m:length(b),c,sq],
  sq:[b[1].b[1]],
  for i:2 step 1 thru m do block(
    c:b[i]-sum((b[j].b[i]/sq[j])*b[j],j,1,i-1),
    ratsimp(c),
    sq:endcons(c.c,sq),
    b[i]:c),
  b);

```

Codeblock A.6. Erstellt aus einer $n \times m$ -Matrix eine Matrix, deren i -te Zeile orthogonal zu den Zeilen 1 bis $i - 1$ ist und rationale Einträge besitzt

```

makePytMat(n):=block([A:ident(n),i:rand(n)+1,j,r:rand(2)],
  j:rand_with_prohib(1,n,[i]),
  if r=1 then block(A[i,i]:rand([4/5,-4/5]),
    A[j,j]:A[i,i],
    A[i,j]:rand([3/5,-3/5]),
    A[j,i]:-A[i,j])
  else block(A[i,i]:rand([3/5,-3/5]),
    A[j,j]:A[i,i],
    A[i,j]:rand([4/5,-4/5]),
    A[j,i]:-A[i,j]),
  A);

```

Codeblock A.7. Erstellt eine $n \times n$ -Drehmatrix

```

makePytMat3(n,max):=block([A:ident(n),i:rand(n)+1,j],
  j:rand_with_prohib(1,n,[i]),
  A[i,i]:rand_with_prohib(-max,max,[0]),
  A[j,j]:A[i,i],
  A[i,j]:rand_with_prohib(-max,max,[0]),
  A[j,i]:-A[i,j],
  A);

```

Codeblock A.8. Erstellt eine $n \times n$ -Matrix, die einer Drehmatrix ähnlich ist.

A.2 Graphen

```
sortEdges(edges):=block([n:length(edges),temp],
for i:1 thru n do
if edges[i][2]<edges[i][1]
then block(temp:reverse(edges[i]),edges[i]:temp),
listify(setify(edges)));
```

Codeblock A.9. Sortiert Kanten, so dass der niedriger indizierte Knoten zuerst genannt wird

```
edgeToAdjacence(n,edges):=block([h:zeromatrix(n,n)],
for i:1 thru n do for j:1 thru n do
if elementp([i-1,j-1],edges) or elementp([j-1,i-1],
edges)
then h[i,j]:1,
h);
```

Codeblock A.10. Erstellt aus Kantenmenge `edges` und Anzahl der Knoten eine Adjazenzmatrix

```
genSummands(n,k):=block([s:0,i,liste:[],t],
for i:1 thru k-1 do block(
t:rand(n-s+1),
liste:endcons(t,liste),
s:s+t
),
liste:endcons(n-s,liste),
liste
);
```

Codeblock A.11. Findet eine Zerlegung von n in k Summanden zwischen 0 und n . Wird für `createTreeGraph(n,k)` benötigt

```

createTreeGraph(m,n):=block(
  [levels:[[0]],t:0,r:[],edges:[],points:[[0,0]]],
  for i:2 thru m-1 do block([l,li:[]],
    l:rand(n-m-t+i-1)+1,
    for i:1 thru l do li:endcons(t+i,li),
    levels:endcons(li,levels),
    t:t+l),
  for i:1 thru n-t-1 do r:endcons(t+i,r),
  levels:endcons(r,levels),
  for i:1 thru length(levels[2]) do
    edges:endcons([0,i],edges),
    t:length(levels[1])-1,
  for i:2 thru m-1 do block(
    [t2:0,l1:length(levels[i]),
    l2:length(levels[i+1]),k:0
    ,nextLevel],
    nextLevel:genSummands(l2,l1),
    for j:1 thru l1 do block(
      for i2:1 thru nextLevel[j] do
        edges:endcons([t+j,t+l1+t2+i2],edges),
        t2:t2+nextLevel[j]),
      t:t+l1),
  for i:2 thru m do block([l1:length(levels[i])],
    if evenp(l1)
    then for j:0 thru l1-1 do
      points:endcons([i-1,-l1/2+j],points)
    else for j:0 thru l1-1 do
      points:endcons([i-1,-(l1-1)/2+j],points)
    ),
  [edges,points]
);

```

Codeblock A.12. Liefert einen Baum mit n Knoten, wobei v_0 Wurzelknoten ist, von dem alle anderen Knoten maximal m Kanten entfernt liegen. Die $n-1$ Kanten sind zufällig verteilt und die Koordinaten der Knoten so platziert, dass die Kanten überschneidungsfrei dargestellt sind.

```

planGraph(vertice,edges):=block([i:0,j,ans:true], while ans=
  true and i<length(edges) do block(i:i+1,
    for j:i+1 thru length(edges) do
      if not(member(edges[i][1], edges[j]) or member(edges[
        i][2],edges[j])) then
        if edgeCrossing([vertice[edges[i][1]+1], vertice[
          edges[i][2]+1]], [vertice[edges[j][1]+1], vertice[
            edges[j][2]+1]])=true then ans:false),ans);
/*liefert true bei planarer Darstellung*/

```

Codeblock A.13. Überprüft inwiefern ein Graph planar ist

```

edgeCrossing(e,f):=block([a:first(e),b:second(e),c:first(f),
  d:second(f),ny,nz],
  if a[1]=c[1]
    then if b[1]=d[1] then ny=0 else ny:(b[1]-d[1])/(c
      [1]-a[1])
    else
      if ((c[2]-d[2])*(a[1]-b[1])=(a[2]-b[2])*(c[1]-d[1]))
        then ny:0
        else block(ny:-((b[1]-a[1])*c[2]+(a[1]-c[1])*b
          [2]+(c[1]-b[1])*a[2])/((b[1]-a[1])*d[2]+(a[1]-b
            [1])*c[2]+(c[1]-d[1])*b[2]+(d[1]-c[1])*a[2]),
            nz:(c[1]-a[1])/(b[1]-a[1])+(d[1]-c[1])/(b[1]-
              a[1])*ny),
  if 0<ny and ny<1 and 0<nz and nz<1 then true else
    false);
/*ueberprueft, ob sich zwei Kanten ausserhalb der
  Knoten kreuzen*/

```

Codeblock A.14. Überprüft ob sich zwei Strecken in der Ebene kreuzen

```

k:4
n:rand_with_step(3,9,1)
n3:n+rand([-1,1])
zr0:rand_with_prohib(-k,k,[0])
zi0:rand_with_step(-k,k,1)
z0:zr0+zi0*i
z1:rand_with_step(1,k,1)
zfr2:rand_with_step(-k,k,1)
zfi2:rand_with_prohib(-k,k,[0])
zf4:rand_with_prohib(1,k,[z1])
size_rp:max(zr0+z1,zr0+zfr2+z1,zr0+zf4)+1
size_rn:min(zr0-z1,zr0+zfr2-z1,zr0-zf4)-1
size_ip:max(zi0+z1,zi0+zfi2+z1,zi0+zf4)+1

```

```

size_in:min(zi0-z1,zi0+zfi2-z1,zi0-zf4)-1
t1:allroots((z-z0)^n-z1^n)
real1:map(realpart,map(rhs,t1))
imag1:map(imagpart,map(rhs,t1))
real2:real1+zfr2
imag2:imag1+zfi2
t3:allroots((z-z0)^n3-z1^n3)
real3:map(realpart,map(rhs,t3))
imag3:map(imagpart,map(rhs,t3))
t4:allroots((z-z0)^n-zf4^n)
real4:map(realpart,map(rhs,t4))
imag4:map(imagpart,map(rhs,t4))
plot1:plot([discrete,real1,imag1],[x,size_rn,size_rp
],[y,size_in,size_ip],[style,points],[point_type
,bullet],[color,red],[ylabel,"Im(z) i"],[xlabel,"
Re(z)"])
plot2:plot([discrete,real2,imag2],[x,size_rn,size_rp
],[y,size_in,size_ip],[style,points],[point_type
,bullet],[color,red],[ylabel,"Im(z) i"],[xlabel,"
Re(z)"])
plot3:plot([discrete,real3,imag3],[x,size_rn,size_rp
],[y,size_in,size_ip],[style,points],[point_type
,bullet],[color,red],[ylabel,"Im(z) i"],[xlabel,"
Re(z)"])
plot4:plot([discrete,real4,imag4],[x,size_rn,size_rp
],[y,size_in,size_ip],[style,points],[point_type
,bullet],[color,red],[ylabel,"Im(z) i"],[xlabel,"
Re(z)"])
plotlist:[plot1,plot2,plot3,plot4]
plotlist:random_permutation(plotlist)
ta:[[a, if first(plotlist)=plot1 then true else false
],[b, if second(plotlist)=plot1 then true else
false],[c, if third(plotlist)=plot1 then true else
false],[d, if fourth(plotlist)=plot1 then true
else false]]

```

Codeblock A.15. Erstellung von Boxplots zu komplexen Wurzeln

Anhang B

Aufgaben

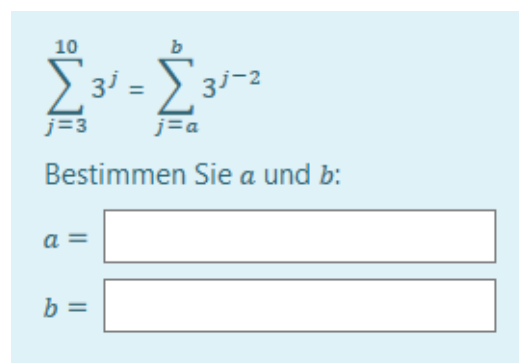
Hier finden sich unterschiedliche Aufgabentypen, die im Zuge der Arbeit implementiert worden sind.

B.1 Indexshift

Das Summenzeichen ist für viele Studierenden eine Herausforderung. In dieser Aufgabe sollen die Summationsgrenzen einem Indexshift angepasst werden.

Die Aufgabenvariablen

Die Zufallsvariablen in dieser Aufgabe sind die Summationsgrenzen, der Shift selber, der nicht Null beträgt, und der Summand. Der Summand wird zufällig aus den Termen j^2 , $j + a$, $j - a$, $j \cdot a$, a^j mit geeignetem a .



The screenshot shows a light blue rectangular box containing a mathematical equation and input fields. The equation is $\sum_{j=3}^{10} 3^j = \sum_{j=a}^b 3^{j-2}$. Below the equation, the text "Bestimmen Sie a und b :" is displayed. Underneath this text are two input fields: the first is preceded by " $a =$ " and the second by " $b =$ ".

Abbildung B.1: Eingeben werden Summationsunter- und Obergrenze

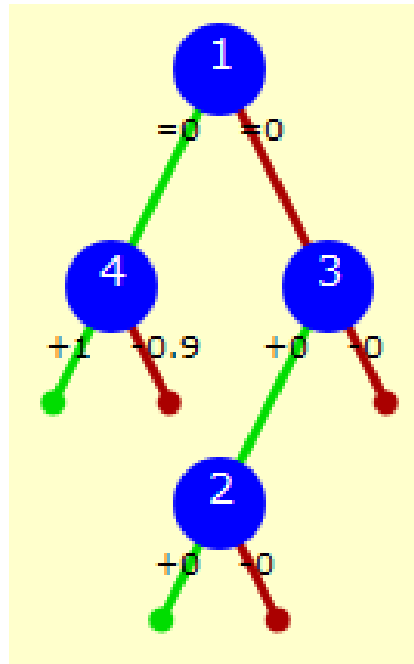


Abbildung B.2: Feedback zu den Summationsgrenzen. Stimmt die Summe, werden Punkte vergeben.

Das Feedback

Sinn der Aufgabe ist es einen Indexshift durchzuführen. Unter Umständen sind allerdings auch andere Summationsgrenzen möglich, die letztendlich dieselbe Summe ergeben. Entsprechend wird als erstes überprüft, ob das Ergebnis der Summation nach Änderung des Summanden mit der ursprünglichen Summation übereinstimmt, und danach erst, ob tatsächlich ein Indexshift durchgeführt wurde. Falls beide Seiten der Gleichung nicht übereinstimmen, wird rückgemeldet, ob beide Grenzen in die falsche Richtung verschoben wurden oder ob die Anzahl der Summanden verändert wurde.

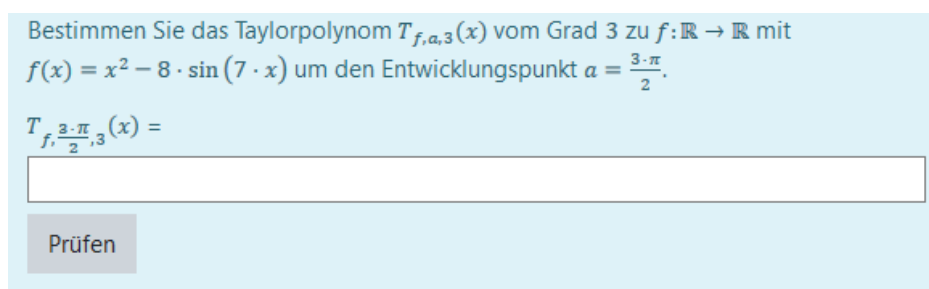
Kategorisierung

Es handelt sich um eine einfache Wiederholungsaufgabe ohne Interpretation seitens der Studierenden.

Einsatz

Die Aufgabe ist nur bei freiwilligen Tests eingesetzt worden. Die bearbeiteten Aufgaben waren in 90% der Fälle korrekt. Wenige (5%) hatten eine falsche

Anzahl an Summanden, die restlichen entweder dieselben Grenzen wie auf der linken Seite angegeben oder welche, bei denen kein System feststellbar war. Ein Kapitel der Analysis behandelt Taylorreihen und Taylorpolynome. Um Taylorpolynome zu berechnen, ist lediglich der aus der Vorlesung bekannte Algorithmus notwendig. MAXIMA selbst bietet einen Befehl an, der auch in STACK implementiert ist und so zu einer gegebenen Funktion und dem gewünschten Entwicklungspunkt einfach das Taylorpolynom eines beliebigen Gerades berechnet. Haben wir also ein Polynom gegeben, ist es einfach, die entsprechende Lösung zu finden.



Bestimmen Sie das Taylorpolynom $T_{f,a,3}(x)$ vom Grad 3 zu $f: \mathbb{R} \rightarrow \mathbb{R}$ mit $f(x) = x^2 - 8 \cdot \sin(7 \cdot x)$ um den Entwicklungspunkt $a = \frac{3 \cdot \pi}{2}$.

$T_{f, \frac{3 \cdot \pi}{2}, 3}(x) =$

Prüfen

Abbildung B.3: Darstellung der Aufgabe in moodle

B.2 Taylorpolynome

Ein Kapitel der Analysis behandelt Taylorreihen und Taylorpolynome. Um Taylorpolynome zu berechnen, ist lediglich der aus der Vorlesung bekannte Algorithmus notwendig. MAXIMA selbst bietet einen Befehl, der auch in STACK implementiert ist und so zu einer gegebenen Funktion und dem gewünschten Entwicklungspunkt einfach das Taylorpolynom eines beliebigen Gerades berechnet. Haben wir also ein Polynom gegeben, ist es einfach, die entsprechende Lösung zu finden.

Die Aufgabenvariablen

Für den Grad n des Taylorpolynoms gilt: $n \in \{3, 4\}$. Der Entwicklungspunkt hängt von der zu untersuchenden Funktion ab, um Definitionslücken zu vermeiden. Die eingesetzten Funktionen in dieser Variante finden sich in Tabelle B.1.

$f(x)$	mögliche Entwicklungspunkte a
$a \cdot x^b \sin(c \cdot x) + d \cdot x^e$	$\frac{k \cdot \pi}{2}$ mit $(k \neq 0), k \cdot \pi \pm \frac{\pi}{6}$
$a \cdot x^b \cos(c \cdot x) + d \cdot x^e$	$\frac{k \cdot \pi}{2}$ mit $(k \neq 0), k \cdot \pi \pm \frac{\pi}{3}$
$a \cdot \cos(b \cdot x) \cdot (c \cdot x)$	$\frac{k \cdot \pi}{2}$ mit $(k \neq 0)$
$a \cdot \tan(b \cdot x)$	$k \cdot \pi$ mit $(k \neq 0), k \cdot \pi \pm \frac{\pi}{4}$
$a \cdot e^{b \cdot x} \cdot \sin(c \cdot x)$	$\frac{k \cdot \pi}{2}$ mit $(k \neq 0)$
$a \cdot e^{b \cdot x} \cdot \cos(c \cdot x)$	$\frac{k \cdot \pi}{2}$ mit $(k \neq 0)$

Tabelle B.1: Funktionen und Entwicklungspunkte

Das Feedback

Das Feedback überprüft, ob das eingegebene Polynom mit der Musterlösung übereinstimmt. Andernfalls wird überprüft, ob es sich bei der Eingabe überhaupt um ein Polynom passenden Grades handelt, ob eventuell $x_0 = 0$ als Entwicklungspunkt gewählt wurde, oder ob durch Substitution von $(x - x_0)$ das gewünschte Polynom entstehen würde. Ein häufiger Fehler Studierender in Hausaufgaben ist, dass im Prinzip richtig gerechnet wird, aber die Verschiebung schlicht vergessen wird.

Kategorisierung

Die Aufgabe ist ein reines Training. Computeralgebrasysteme können mit einem Befehl die Antwort liefern.

B.2.1 Varianten

In einer weiteren Aufgabe wird nach dem Taylorpolynom eines Polynoms gefragt. Für den Entwicklungspunkt $x = 0$ ist dies trivial. Diese Erkenntnis sollte aber auch festgestellt werden.

Erweiterungen

Weitere Funktionentypen wären denkbar, allerdings muss gewährleistet werden, dass die Funktion und alle ihre Ableitungen an der Stelle x_0 definiert sind. Aus diesem Grund ist es zwingend erforderlich, dass pro Funktion eine Liste mit möglichen Entwicklungspunkten angegeben wird. Andernfalls müsste man eine Routine finden, die beim Erstellen der Aufgabe die unmöglichen Kombinationen herausfiltert.

B.3 Bild und Kern (ohne Rechnen)

Die Studierenden werden aufgefordert Dimension von Kern und Bild einer Matrix anzugeben, daraus eine Aussage über die Injektivität und Surjektivität zu treffen sowie eine Basis beider Räume zu finden.

Sei $\mathcal{L}_A$ eine lineare Abbildung mit $\mathcal{L}_A: \mathbb{R}^3 \Rightarrow \mathbb{R}^4$ mit der darstellenden Matrix $A = \begin{pmatrix} 3 & 3 & 1 \\ 0 & 18 & -9 \\ 0 & 0 & -15 \\ 0 & 0 & 0 \end{pmatrix}$.

Berechnen Sie die Dimensionen von Bild und Kern der Abbildung.

$\dim(\text{Bild}(L)) =$

$\dim(\text{Kern}(L)) =$

Es gilt:

Geben Sie eine Basis $\mathcal{B} = (b^{(1)}, \dots)$ des Bildes an:

$b^{(1)} =$

$b^{(2)} =$

$b^{(3)} =$

$b^{(4)} =$

Geben Sie eine Basis $\mathcal{C} = (c^{(1)}, \dots)$ des Kerns an:

$c^{(1)} =$

$c^{(2)} =$

$c^{(3)} =$

$c^{(4)} =$

Abbildung B.4: Interpretation der Abbildungsmatrix

Die Aufgabenvariablen

Zeilen- und Spaltenrang n_b bzw. n_c der Matrix A ist jeweils eine Zufallszahl aus der Menge $\{2, 3, 4\}$. Die Matrix A ist eine zufällige Matrix M aus M^{n_b, n_c} , die auf Zeilenstufenform gebracht wurde. Die Einträge der ursprünglichen Matrix M liegen zwischen $-n$ und n , in diesem Beispiel wurde $n = 5$ gewählt.

In den Variablen wird ebenfalls die Musterlösung hinterlegt. Die Dimensionen werden über den Befehl `rank(A)` gefunden, die Basis des Kern mit

```
append(args(nullspace(A)), [" existiert nicht " , ..., "
    existiert nicht " ]),
```

die des Bildes analog als Liste der Pivotspalten, die mit "existiert nicht" aufgefüllt wird. Die Funktion `pivot(A)` sammelt hierbei zeilenweise das erste von Null verschiedene Element in einer Liste.

Weiterhin wird die Auswahlliste zur Injektivität mit den Wahlmöglichkeiten $p_1, \dots, p_4$ aus den berechneten Dimensionen generiert mit

```
[[p1,r=nc and nb-r=0,"Die Abbildung ist bijektiv."],
 [p2,r=nc and nb>r,"Die Abbildung ist surjektiv aber nicht
   injektiv."],
 [p3,r<nc and nb-r=0,"Die Abbildung ist injektiv aber nicht
   surjektiv."],
 [p4,r<nc and nb>r, "Die Abbildung ist weder injektiv noch
   surjektiv."]],
```

so dass abhängig vom zufälligen Zeilen- und Spaltenrang der Matrix A jeder Aussage ein Wahrheitswert zugewiesen wird.

Das Feedback

Das Feedback ist in drei Teile gegliedert. Jeder dieser drei Teile ist 1 P wert. Im ersten Teil werden Dimension von Bild und Kern sowie die Injektivität und Surjektivität untersucht. Im ersten Knoten des Feedbackbaums aus Abbildung B.5 wird überprüft, ob die Antwort des Studierenden mit dem Rang der Matrix übereinstimmt, in diesem Fall werden 0,3 P vergeben. Der zweite Knoten überprüft, ob die Dimension des Kerns korrekt ist, was ebenfalls 0,3 P bedeutet. Falls dieser Wert nicht mit der Musterlösung übereinstimmt, wird in Knoten Drei überprüft, ob das Ergebnis zur angegebenen Dimension des Bildes passt, was auch mit 0,3 P gewertet wird. Knoten Vier prüft außerdem, ob anstelle des Ranges von A die Anzahl der Zeilen verwendet wurde. Die beiden ersten Antworten werden in Knoten Fünf in den Dimensionssatz eingesetzt, was 0,1 P liefert. Die letzten 0,3 P werden in Knoten Sechs für die richtige Aussage zu Injektivität und Surjektivität vergeben.

Die beiden anderen Feedbackbäume aus Abbildung B.6 sind im Aufbau identisch. Zunächst wird überprüft, ob die Anzahl der Vektoren zur angegebenen Dimension passt, nur dann wird der Feedbackbaum weiter durchlaufen. Ist diese Dimension Null, wird der Baum im zweiten Schritt gleich mit 1 P verlassen. Andernfalls werden die angegebenen Basisvektoren auf ihre Dimension geprüft, sollte ein Vektor der falschen Dimension enthalten sein, wird der Baum verlassen. Sind die Dimensionen oben richtig gewesen, ist der Rest einfach. Aus den angegebenen Vektoren wird eine Matrix gebildet. Ist der Rang dieser Matrix korrekt und liegt jeder dieser Vektoren im Bild bzw. Kern (Knoten Drei und Vier) kann der Baum erfolgreich mit 1 P verlassen

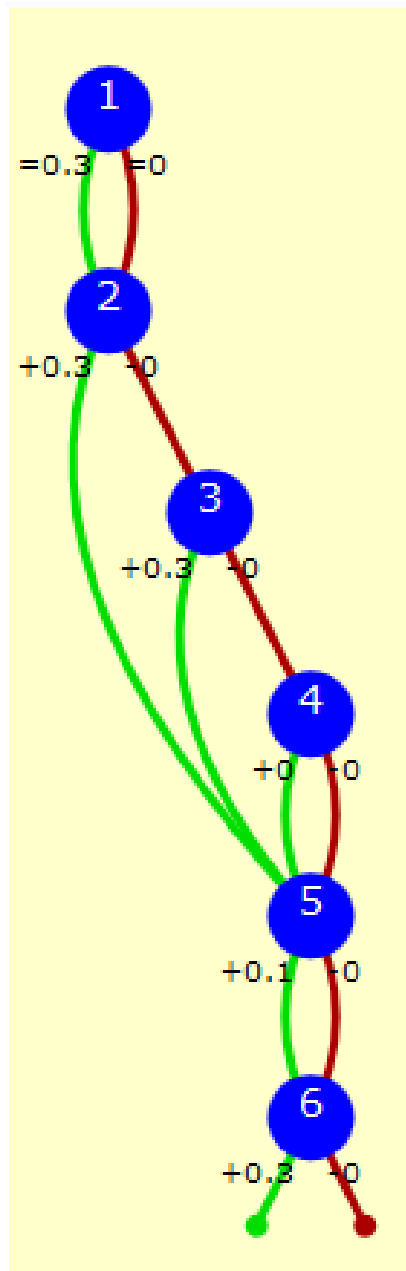


Abbildung B.5: Feedback zur Dimension von Bild und Kern

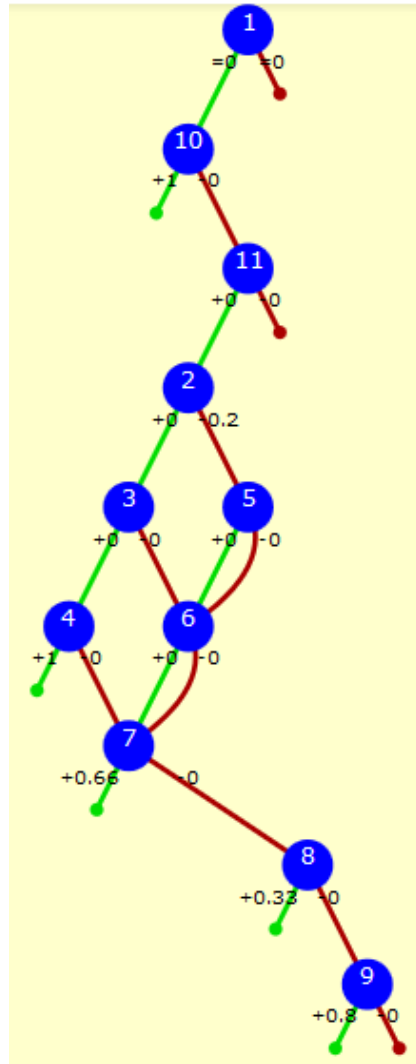


Abbildung B.6: Feedback zur Basis von Kern bz. Bild

werden. Knoten Fünf und Sechs liefern analoges Feedback, führen am Ende aber zu Knoten Sieben, der dann mit Knoten Acht und Neun anhand der Anzahl an fehlerhaften Angaben Punkte verteilt.

Kategorisierung

Diese Variante benötigt kaum Berechnungen seitens der Studierenden, sie setzt stattdessen die Fähigkeit ein Ergebnis zu interpretieren voraus. Lediglich der Kern der Matrix kann für $\dim(\text{Kern}(A)) > 0$ nicht direkt abgelesen werden.

Varianten

Diese Aufgabe existiert in einer Variante, in der die Matrix A nicht in Zeilen-Stufen-Form vorliegt.

Erweiterungen

Die momentane Variante erstellt die Matrizen absolut zufällig, die Wahrscheinlichkeit eine bestimmte Dimension des Kerns/Bildes zu erhalten hängt von der Dimension der Matrix und der Wahl der Koeffizientengröße ab.

B.3.1 Kern und Bild (3x3, neue Einstellungen, flexibler Rang),

Idee

Studierende sollen die lineare Abbildung, die durch eine 3×3 - Matrix A beschrieben wird, untersuchen und Basen von Bild und Kern bestimmen.

Elternaufgaben

- gleiche Aufgabenstellung mit umständlicher Eingabe
- Aufgabenstellung ohne Baseneingabe

Weiterentwicklung der Aufgabe

- Kern und Bild (2×3)
- Kern und Bild (4×4)
- Basis und Dimension im Polynomraum

Randomisierung

- ganzzahlige Matrixeinträge der Matrix A über L-R-Zerlegung
- Rang über Auswahl aus Matrix A oder über Multiplikation von A mit einer Matrix vom Rang 2

eingebaute Funktionen

- Matrixerstellung über
`genmatrix()`
- Berechnung der Antwortenliste zur Erstellung des Dropdownmenüs mithilfe des Rangs von A
- Bestimmung der Pivotelemente über
`triangularize()`
- Erstellung und Aufbereitung der Basisvektoren des Bildes über Pivotelemente und ergänzen von
`EMPTYANSWER`
für überzählige Eingabefelder

- Erstellung und Aufbereitung der Basisvektoren des Kerns mit

`nullspace(A)`

und

`EMPTYANSWER`

allgemeines Feedback nicht vorgesehen
Eingabefelder

- Dimension von Kern und Bild als Zahlenwert (**dimBild**, **dimKern**)
- Dropdownmenü zu nur injektiv/nur surjektiv/bijektiv/weder injektiv noch surjektiv (**typ**)
- 4 Felder für Basisvektoren des Bildes als frei einzugebene Matrix, überzählige Felder dürfen leer gelassen werden (**vb1, vb2, vb3, vb4**)
- analog 4 Felder für Basisvektoren des Kerns (**vk1, vk2, vk3, vk4**)

Feedbackbäume

- **dimension**
 - Wert 2/7
 - triggert bei **dimBild**, **dimKern**, **typ**
 - zusätzliche Funktionen: Übersetzung der Dropdownantwort
 - Feedback-Baum
 - * Ist die Dimension des Bildes korrekt? (0.25/0)
 - * Ist die Dimension des Kerns korrekt? (+0.25/+0)
 - * Falls **dimKern** falsch, passt sie zu **dimBild** (+0.25/+0)
 - * Falls auch das nicht, und Start/Ziel-Raum unterschiedlich: Wurde der Zielraum für Dimensionssatz verwendet? (+0/+0)
 - * Ist der Dimensionssatz erfüllt? (+0.25/+0)
 - * Passen die Ergebnisse zu der Dropdowneingabe **typ**? (+0.25/+0)
- **basisbild**
 - Wert 3/7
 - triggert bei **dimBild**
 - verwendet **dimBild**, **vb1**, **vb2**, **vb3**, **vk4**

- zusätzliche Funktionen:
 - * Antwortbereinigung
 - * Erstellung der Antwortmatrix aus **vb1, ..., vb4**
 - * Vergleich der Bildräume von A mit den Vektoren der Studierenden
 - * Vergleich zur behaupteten Dimension **dimKern**
- Feedbackbaum
 - * Passt Anzahl der Basisvektoren zu **dimBild** (0/0 ENDE)
 - * Dimension 0 und korrekt? (+1 ENDE/+0)
 - * Basisvektoren richtige Länge? *intern nötig um Matrix zu erstellen*(+0/+0)
 - * passender Rang? (nur Feedback)
 - * Menge falscher Vektoren leer? (+0.5/+0)
 - * Menge fehlender Vektoren leer? (+0.5/+0)
- **basiskern**
 - Wert 3/7
 - triggert bei **dimKern**
 - verwendet bei **dimKern, vk1, vk2, vk3, vk4**
 - zusätzliche Funktionen:
 - * Antwortbereinigung
 - * Erstellung der Antwortmatrix aus **vk1, ..., vk4**
 - * Vergleich des Kerns von A mit den Vektoren der Studierenden
 - * Vergleich zur behaupteten Dimension **dimKern**
 - Feedbackbaum
 - * Passt Anzahl der Basisvektoren zu **dimKern** (0/0 ENDE)
 - * Dimension 0 und korrekt? (+1 ENDE/+0)
 - * Basisvektoren richtige Länge? *intern nötig um Matrix zu erstellen*(+0/+0)
 - * passender Rang? (nur Feedback)
 - * Menge falscher Vektoren leer? (+0.5/+0)
 - * Menge fehlender Vektoren leer? (+0.5/+0)

STACK Frage Dashboard

Sei L_A die lineare Abbildung $L_A: \mathbb{R}^3 \rightarrow \mathbb{R}^3$ mit der darstellenden Matrix $A = \begin{pmatrix} 13 & 5 & 11 \\ 8 & -2 & 4 \\ 17 & -7 & 7 \end{pmatrix}$.

Berechnen Sie die Dimensionen von Bild und Kern der Abbildung.

$\dim(\text{Bild}(L)) =$

$\dim(\text{Kern}(L)) =$

Es gilt:

Geben Sie eine Basis $\mathcal{B} = (b^{(1)}, \dots)$ des Bildes $\text{Bild}(L)$ an:

$b^{(1)} =$

$b^{(2)} =$

$b^{(3)} =$

$b^{(4)} =$

Geben Sie eine Basis $\mathcal{C} = (c^{(1)}, \dots)$ des Kerns $\text{Kern}(L)$ an:

$c^{(1)} =$

$c^{(2)} =$

$c^{(3)} =$

$c^{(4)} =$

Geben Sie die Basisvektoren als Matrix ein. Verwenden sie Leerzeichen für eine neue Spalte und Return für eine neue Zeile. Überzählige Vektoren dürfen leer gelassen werden.

Abbildung B.7: Bild und Kern einer Linearen Abbildung

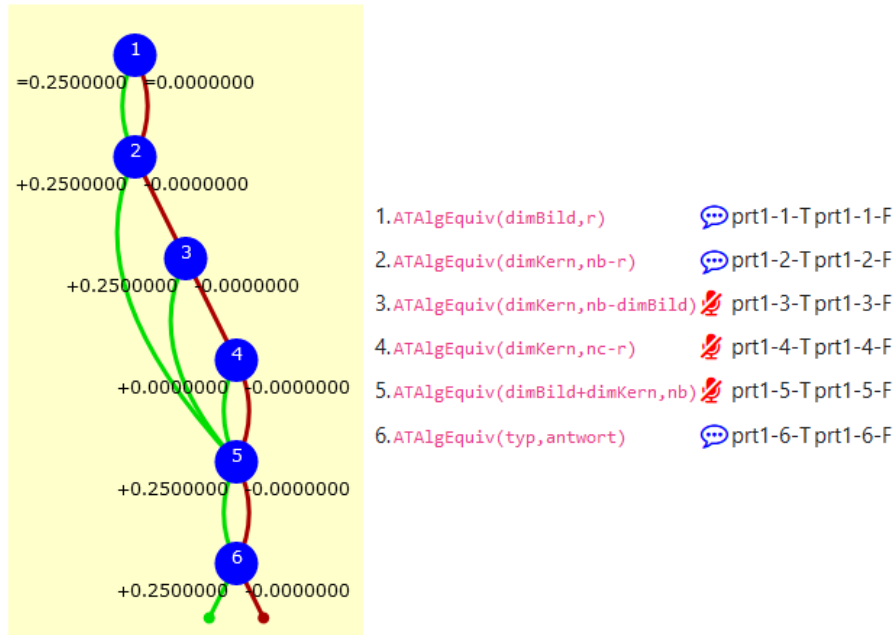


Abbildung B.8: Feedbackbaum zur Dimension

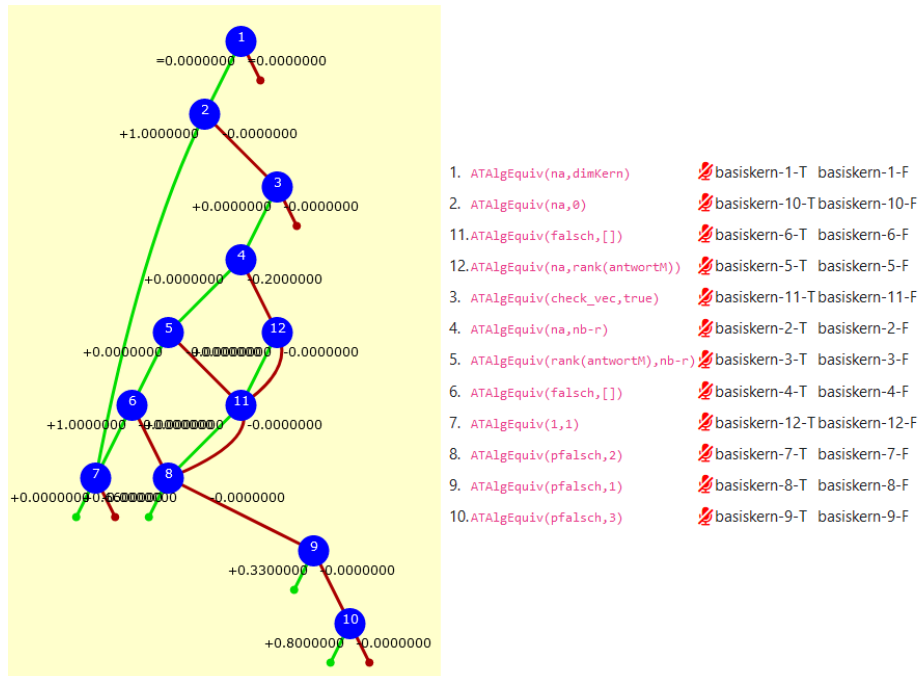


Abbildung B.9: Feedbackbaum zur Basis des Kerns

Anhang C

Dynamische Grafiken mit JSXGraph

JSXGraph eine JavaScript-Bibliothek, mit deren Hilfe mathematische Objekte in Webbrowsern dargestellt werden können. Verfügt man bereits über JavaScript-Kenntnisse wird einem der Einstieg vermutlich leicht fallen. Andernfalls soll dieser Teil eine kurze Einführung zum Erstellen von JSXGraph-Applets liefern. Hier möchte ich zeigen, wie geometrische Formen oder Funktionenplots in einem Applet bereit gestellt werden und auch durch den Nutzer des Applets bearbeitet werden können. Ähnlich wie in einem Geogebra-Applet lassen sich so Punkte, Geraden oder Figuren verschieben oder konstruieren, abhängig davon, was der Autor des Applets vorgesehen hat. Ich beschränke mich hier auf die Nutzung innerhalb von STACK, abgesehen von der Variablenübergabe in den JSXGraph-Block und den Aufruf desselben, sind die Befehle aber direkt übertragbar. Zusätzliche Dokumentation findet sich unter [WWA24] wo auch diverse Beispiele zu finden sind.

STACK bindet JSXGraph-Anwendungen direkt in Fragetext oder Feedback ein, dies geschieht über einen Questionblock

```
[[jsxgraph]]...[[/jsxgraph]].
```

Bei Aufruf dieses Blocks kann unter anderem auch die Größe des Applets auf der Seite festgelegt werden. Dazu ergänzen wir `[[jsxgraph width="x" height="y"]]` erreicht, wobei der Defaultwert `width="200px" height="200px"` beträgt.

In moodle kann der User einen Editor wählen, der zur Erstellung des Aufgabentexts und Feedback verwendet wird. Einige dieser Editoren erstellen HTML-Code beispielsweise bei Zeilenumbrüchen, der aber nicht angezeigt wird. Dies kann den Code, der zur Erstellung des Applets eingegeben wird,

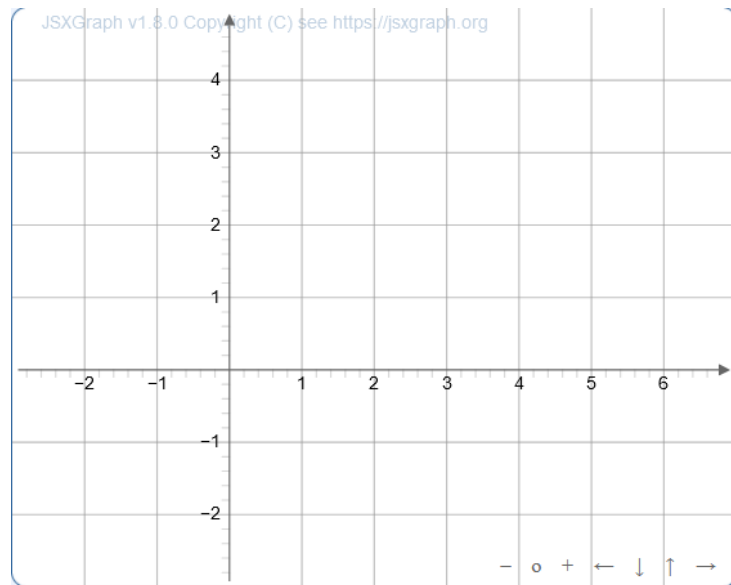


Abbildung C.1: Board mit Startkonfiguration `{boundingbox: [-3,5,7,-3], axis:true, grid:true}`

verfälschen. Entsprechend sollte bei Erstellung oder Bearbeitung grundsätzlich die Wahl eines passenden Editors beachtet werden.

JavaScript ist eine objektorientierte Sprache. Wir haben es in JSXGraph mit unterschiedlichen Objekten und deren Attributen und Methoden zu tun, angefangen mit dem Objekt `board`, das unsere Zeichenebene ist und mit

```
var board = JXG.JSXGraph.initBoard();
```

initialisiert wird. An dieser Stelle können bestimmte Eigenschaften dieser Zeichenebene gesetzt werden, wie z.B. ob ein Gitter oder Koordinatenachsen angezeigt werden sollen. Dies geschieht indem wir der `initBoard()`-Methode folgende Attribute mitgeben:

```
divid, {boundingbox: [left,top,right,bottom], axis:true, grid: true}
```

und wir erhalten eine Zeichenebene, die durch `left`, `top`, `right` und `bottom` beschränkt wird. Der angezeigte Ausschnitt im Applet ist während der Nutzung veränderbar.

Abbildung C.1 zeigt eine Zeichenebene mit standardmäßigem Copyright und Navigationselementen. Diese lassen sich optional aber auch abschalten.

Punkte

Um ein Objekt wie einen Punkt p_1 zu erstellen, verwenden wir auf unserem `board` die `create()`-Methode. Wir ergänzen den Code zur Erstellung der Ebene durch:

```
var p1 = board.create('point',[x,y]);
```

Damit ist p_1 ein Objekt des Typs `point` und den Koordinaten (x, y) . Standardmäßig werden Punkte dem Alphabet nach mit Buchstaben benannt, und sind frei bewegbar in der Ebene. Dies lässt sich durch zusätzliche Attribute aber ändern. Beispielsweise würde

```
var p1 = board.create('point',[x,y],{name:'p1',face:'[]',  
    strokecolor:'blue',fillcolor:'blue',trace:true});
```

einen blauen, quadratischen Punkt mit dem Namen `P1` erstellen, der an zunächst bei den Koordinaten (x, y) liegt, aber bewegt werden kann und dabei seine Spur in der Zeichenebene hinterlässt. Ein Punkt kann direkt durch seine Koordinaten gegeben werden, aber auch abhängig von anderen Objekten sein.

```
var p2 = board.create('point',['X(P1)','Y(P1)-1'],  
    {name:'P2', face:'>', size:3});
```

erstellt einen zweiten Punkt p_2 namens `P2`, dessen y -Koordinate gegenüber p_1 verschoben ist, wie in Abbildung C.2 zu sehen. Auch nachträglich lassen sich Attribute verändern:

```
p2.setAttribute()
```

weist dem Punkt p_2 alle in der Klammer aufgezählten Attribute zu. Eventuell wird ein `fixed:true` an dieser Stelle allerdings ignoriert, falls das zu bearbeitende Objekt von anderen Objekten abhängig ist. In unserem Beispiel würde der Punkt p_2 immer noch bewegt werden, sobald p_1 bewegt wird.

Geraden und Kreise

Weitere geometrische Standardobjekte sind Geraden und Kreise. Diese werden jeweils durch zwei Punkte definiert. Eine Gerade erhalten wir durch

```
l1:board.create('line',[p1,p2]).
```

Dies erstellt eine Gerade durch die Punkte p_1 und p_2 , für die erst einmal mit all ihren Attributen definiert und auf der Koordinatenebene plziert werden. Setzt man anstelle eines im Vorfeld definierten Punkts nur Koordinaten, so

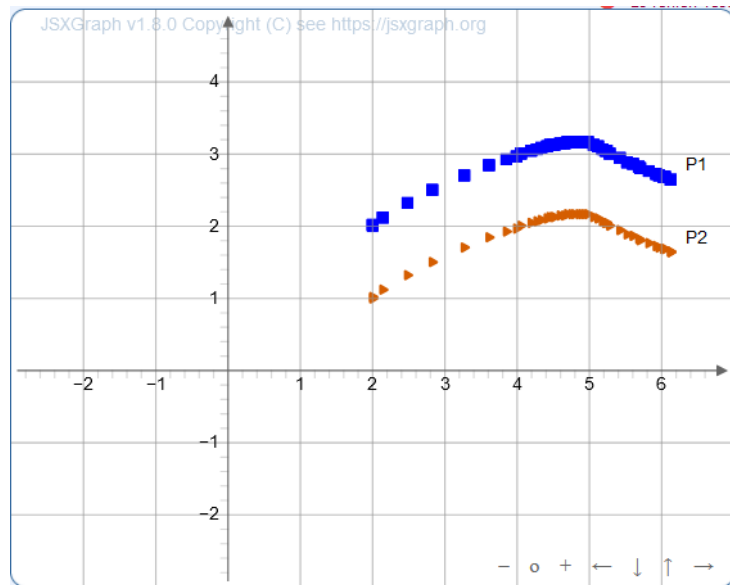


Abbildung C.2: Die Punkte P1 und P2, die bereits verschoben wurden.

sind dies lediglich die Startwerte und sie werden nicht als Punkte auf der Gerade angezeigt. Die Gerade lässt sich, sofern nicht durch `fixed:true` anders definiert, verschieben. Auch lassen sich verschiedene Attribute einer Gerade zuweisen. Mit `straightFirst:false` bzw. `straightLast:false` wird die Gerade vor dem ersten bzw. nach dem zweiten Punkt nicht weitergezeichnet, womit sich Strecken oder Halbgeraden realisieren lassen. Wie für Punkte können wir die Farben oder Strichbreiten anpassen oder sie unsichtbar machen. Wir können sie auf verschiedenste Arten gestrichelt zeichnen oder an den Enden Pfeilspitzen einfügen lassen. Mit

```
var l1 = b2.create('line',[p11,p12],{shadow:true});
var l2 = b2.create('line',[p21,p22],{straightFirst:false,
    straightLast:false, strokeWidth:5, dash:2});
var l3 = b2.create('line',[p31,p32],{straightLast:false,
    lastArrow:true, strokeWidth:2});
```

erstellen wir die Geraden-Objekte zu Abbildung C.3 auf der Zeichenebene `b2` aus zuvor erstellten Punkten.

Auch ein Kreis kann ähnlich definiert werden. Hier verwenden wir ebenfalls zwei Punkte, der erste gibt den Mittelpunkt des Kreises an, der zweite einen Punkt der auf dem Kreis liegt. So würde

```
var c1 = board.create('circle',[p1,p2],{fillColor:'yellow'});
```

uns auf der Ebene `board` einen gelbgefüllten Kreis liefern, mit Mittelpunkt p_1 und Radius $|p_2 - p_1|$. Analog zu Punkten und Geraden sind auch wie-

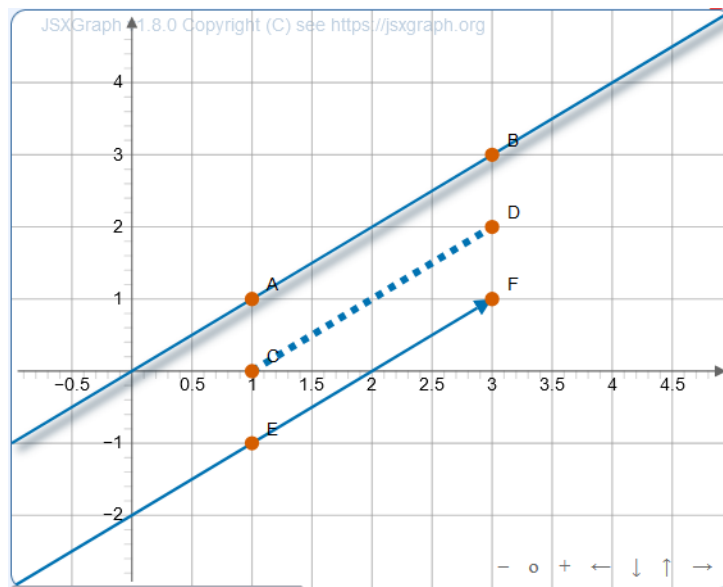


Abbildung C.3: Unterschiedliche Attribute von Objekten des Typs line

der diverse Attribute möglich, die Kreislinie kann gestrichelt, mit Pfeilspitze versehen oder von unterschiedlicher Stärke sein, die Fläche kann eingefärbt werden, dabei kann die Transparenz der Färbung angegeben werden. Die Farben können, wie hier, mit Farbname aufgerufen werden, der Hexwert der Farbe ist aber auch zulässig.

Hinzu kommen weitere geometrische Objekte, wie z.B. Kreisbogenabschnitte 'arc', die durch drei Punkte beschrieben werden, Polygone, die durch entsprechende Punkte definiert sind oder auch beliebig zusammengesetzte Objekte. Einige häufig verwendete Objekte sind bereits vordefiniert, selbst wenn diese von den Usern per Hand erstellt werden könnten, wie z.B. 'angle'. Zur Verdeutlichung definieren wir passend zu Abbildung C.4.

```
var p1 = board.create('point',[2,2],{name:"P1"});
var p2 = board.create('point',["X(P1)","Y(P1)-1"]);
var p3 = board.create('point',["X(P1)+2","Y(P1)-1"]);
var p4 = board.create('point',["X(P1)+2","Y(P1)+1"]);
var c1 = board.create('circle',[p1,p3]);
var a1 = board.create('arc',[p1,p3,p4],
    {strokeWidth:4,lastArrow:true});
var a2 = board.create('angle',[p1,p2,p3])
```

Dem Punkt p1 geben wir den Namen p1. Auf diese Koordinaten greifen wir mit diesem Namen zur Erstellung der anderen Punkte zu. Alle weiteren Punkte werden nun einfach alphabetisch benannt. An dieser Stelle nutzen

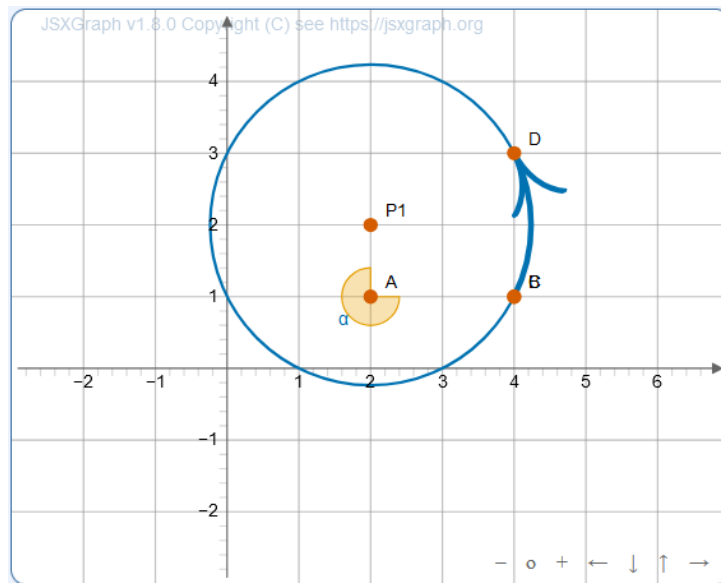


Abbildung C.4: Kreis, Bogen, Winkel

wir GEONExT-Syntax. Etwas länger als der GEONExT-Syntax wäre es, JavaScript-Syntax zu verwenden. So wird aus " $x(p1)$ " nun ein `function()` `{return p1.x() }`. Hierbei rufen wir das Objekt `p1` allerdings direkt auf und greifen nicht über seinen Namen auf es zu. Der Übersicht wegen könnte dies bei einer variablen Anzahl Objekte durchaus sinnvoll sein.

Der Kreis `c1` wird nun einfach durch die Punkte `p1` und `p3` erstellt, in Abbildung C.4 dargestellt als $P1$ und B . Das Kreissegment `a1` von B zu C erstellen wir durch den Mittelpunkt des Kreises `p1` und den Punkt `p3` auf dem Kreisbogen. Um die Bogenlänge zu bestimmen, verwenden wir einen weiteren Punkt `p4` (im Bild C). Wir haben `p4` so gewählt, dass er auf dem Kreisbogen liegt, andernfalls endet der Bogen an dem Schnittpunkt des Kreises mit der Gerade zwischen `p1` und `p4`. Zusätzlich setzten wir hier `lastArrow:true`. Der Winkel `a2` wird durch drei Punkte `[p1,p2,p3]` beschrieben, so wie aus der Geometrie gewohnt, mit dem mittleren der Punkte als dem Mittelpunkt des Winkels zwischen den Punkten `p1` und `p3`.

Kontrollstrukturen

Wir können innerhalb des JSX-Blocks `for`-Schleifen oder `if`-Abfragen nutzen. So können wir beispielsweise eine Liste aus Punkten anlegen. In Kapitel 2.2.3 verwenden wir dies und lassen uns in den STACK-Aufgabenvariablen eine Liste von Koordinaten und Kanten erstellen, auf die wir in dem JSX-Block zugreifen. Hier wird dann aus jeder Koordinate ein Punkt erstellt und aus jeder Kante ein Geradenabschnitt. Letztendlich funktioniert die `for`-Schleife folgendermaßen:

```
for (Initialisierung; Test; Schritt)
    {Anweisungsblock}
```

und die `if`-Anweisung:

```
if (Bedingung) {Anweisungsblock}
else {Anweisungsblock}
```

In der oben erwähnten Aufgabe wurden an die Variable `vPos` die oben definierten Koordinaten übergeben, mit `var vPos = {#vertices#}`. Mit `{#...#}` können wir in STACK auf eine Variable, die ausserhalb des JSX-Blocks definiert ist, zugreifen. Bei `vPos` handelt es sich nun um eine Liste, die die Koordinaten der Knoten als Paare in Listenform enthält:

`vPos := [[x1, y1], [x2, y2], ..., [xn, yn]]`. Möchten wir nun aus diesen Koordinaten eine Liste aus Punkt-Objekten erstellen, so können wir das mit Hilfe einer `for`-Schleifen erreichen. Hierzu definieren wir `var v = []` als momentan leere Liste der Punkte und definieren jeden Punkt `v[i]` in der `for`-Schleife mit

```
for(var i=0; i< vPos.length; i++){v[i]=board.create('point',
    vPos[i]);}
```

Hierbei wird mit `vPos.length` die Zahl n bestimmt, die die Länge der Liste `vPos` angibt. Anders als in den oben mit STACK angelegten Variablen, deren Koordinaten `vertice` mit den Indizes $1..n$ versehen waren, laufen unsere Punkte nun von `v[0]` bis `v[n-1]`. Die einzelnen Kanten können wir ebenfalls über eine Schleife jeweils als `'line'` anlegen lassen, bei der wir die beiden definierenden Punkte der Kante aus den zuvor angelegten Punkten `v[i]` und `v[j]` wählen.

```
var e={#edges#};
for(var i=0; i< e.length; i++)
    { board.create('line', [v[e[i][0]],v[e[i][1]]],
      {straightFirst:false, straightLast:false, lastArrow:
        true, strokeWidth:1.5});
    };
```

liefert beispielsweise Pfeile als gerichtete Kanten. Die Variable `e` wird durch die ausserhalb des JSX-Block definierten Kanten gesetzt die durch die Indizes der an einer Kante beteiligten Knoten bestehen, sodass nun `e` von der

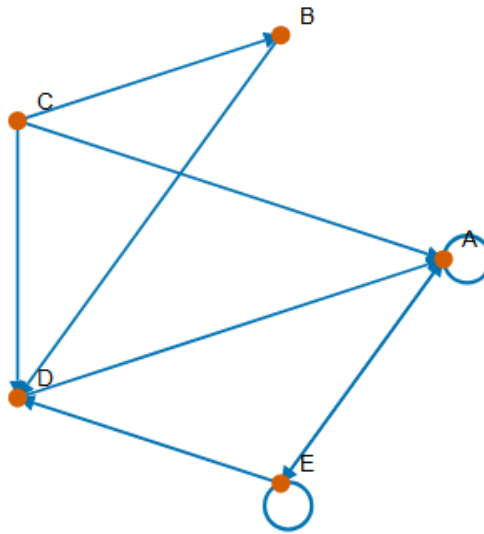


Abbildung C.5: gerichteter Graph mit Schleifen

Form $e = [[i_{1,1}, i_{1,2}], [i_{2,1}, i_{2,2}], \dots, [i_{m,1}, i_{m,2}]]$ ist.

Falls wir einen Graphen zeichnen lassen möchten, der auch Schleifen enthalten kann, so können wir uns das `if`-Statement zu Nutze machen. Falls Anfangs- und Endknoten der Kante übereinstimmt, so zeichnen wir einen Kreis oder Kreisbogen an den Knoten, wie in Abbildung C.5. So unterscheiden wir, ob sich die Indizes des ersten und zweiten Knotens der Kante unterscheiden oder nicht:

```
for(var i=0; e.length; i++)
  {if (e[i][0]!=e[i][1])
    {board.create('line',[v[e[i][0]],v[e[i][1]]],
      {straightFirst:false, straightLast:false,
        lastArrow:true, strokeWidth:1.5}
    );
  }
  else
    {var t=e[i][0];
      board.create('circle',
        [[v[t].X()+0.5*Math.cos(Math.PI * 2*t/{#n#}),
          v[t].Y()+0.5*Math.sin(Math.PI * 2*t/{#n#})],
          v[t]],
        {strokeWidth:2})
    }
  }
```

Hier sind die Knoten auf einem Kreis um den Ursprung (0,0) verteilt, die

Koordinaten sind entsprechend berechnet worden und der Mittelpunkt einer Schleife wird durch

```
[v[t].X()+0.5*Math.cos(Math.PI * 2*t/{#n#}),
 v[t].Y()+0.5*Math.sin(Math.PI * 2*t/{#n#})],
```

abhängig von der Position des Knotens im Kreis, die hier mit `t=e[i][0]`, bestimmt wird.

Eine Funktion haben wir weiter oben bereits einmal verwendet, um beim Anlegen eines abhängigen Punkts die x - und y -Koordinaten des Elternknotens auszulesen. Wir können auch allgemein Funktionen definieren um den Code übersichtlicher zu gestalten. So versehen wir die Funktion mit einem Namen und definieren sie folgendermaßen:

```
function Name(Argumente)
{Anweisungen;
 return Ausgabe
}
```

Wir könnten also die x -Koordinate des Mittelpunkts unseres oben angelegten Kreises für die Schleife im Graphen bestimmen lassen:

```
function MPX(x,t,n)
{return x+v[t].X()+0.5*Math.cos(Math.PI * 2*t/n)};.
```

Mit `MPX(v[t].X(),t,{#n#})` rufen wir diese bei Bedarf auf. In dieser neu definierten Funktion wird bereits die JavaScript-Funktion `Math.cos()` aufgerufen. Neben den in JavaScript enthaltenen Funktionen sind auch ein paar weitere wie etwa `JXG.Math.factorial(n)` in der JSXGraph Bibliothek enthalten, auf die zugegriffen werden kann.

Funktionsgraphen

Wir können mit Hilfe von JSXGraph auch Funktionsgraphen zeichnen lassen. Hierbei erstellen wir ein Objekt `'functiongraph'`. Dies liefert den Plot einer reelwertigen Funktion, die durch $f(x)$ beschrieben wird. Eine Sinuskurve zwischen $x = -2$ und $x = 2$ erhalten wir z.B. mit

```
var f1 = board.create('functiongraph',[function(x){return
Math.sin(x)};,-2, 2])
```

wobei die seitlichen Begrenzungen optional und defaultmäßig bei ± 10 liegen. Anstelle der Zahlen dürfte hier auch eine Funktion, die einen numerischen Wert liefert, stehen.

Alternativ können wir mit `'curve'` arbeiten. Ein `'curve'`-Objekt benötigt bis zu vier Einträge: `[x,y, min, max]` Hierbei können x und y die jeweiligen

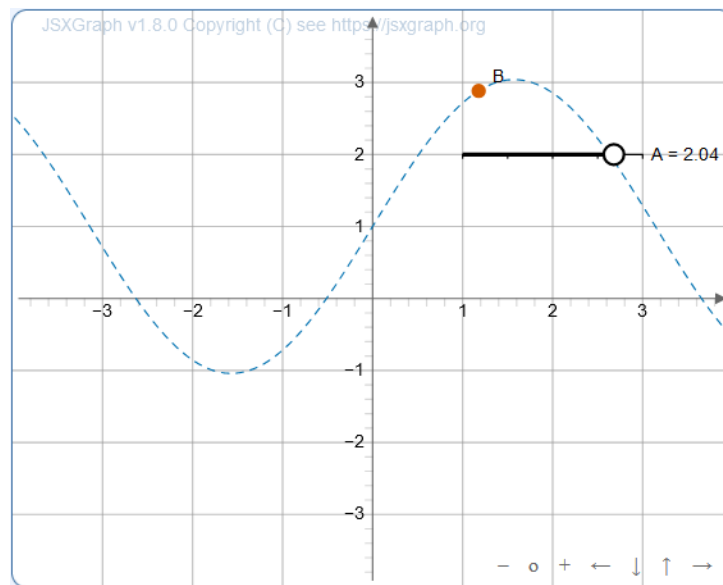


Abbildung C.6: Funktionsplot von $a * \sin(x) + 1$

Komponenten einer parametrisierten Funktion sein, aber auch Listen mit den x - bzw. y -Werten einer Datenmenge und `min` und `max` geben die Grenzen der eingesetzten Parameter an. Für beide Varianten existieren wieder diverse optionale Attribute. Wie beim Beispiel zu Geraden oder Kreisen können wir mit `{...}` einen Block mit Attributen anhängen, so liefert beispielsweise `{dash:2}` eine gestrichelte Kurve.

Nicht nur für Funktionsgraphen, auch für andere Objekte, die konstruiert werden können, ist das `'slider'`-Objekt interessant. Mit diesem Objekt weisen wir einer Variablen einen Wert zu, der durch den Benutzer an einem Schieberegler eingestellt wird. Wir nutzen

```
var a = board.create('slider',
    [[x1,y1],[x2,y2],[min,initial,max]],
    {name:'A'});
```

um einen Schieberegler `a` zu erstellen, der allerdings im angezeigten Fenster den Namen `A` trägt. Die ersten beiden Koordinatenpaare geben hierbei die Lage des Schiebereglers auf der Koordinatenebene an, `min` und `max` die minimalen und maximalen Werte, die `a` annehmen kann. Zu Beginn wird der Regler auf `initial` gesetzt. Optional kann außer dem angezeigten Namen auch eine Schrittweite `t` eingestellt werden, dies geschieht mit `snapWidth:t`. Zugegriffen wird dann mit `a.Value()`.

Neben den Schiebereglern gibt es auch die Option einen Punkt zu defi-

nieren, der auf bestimmten Objekten wie Funktionsgraphen, Geraden oder Kreisen lebt. Dies sind Glider, wir erstellen sie mit

```
var g = board.create('glider',[f1]);.
```

Dies wäre ein Glider g , der sich auf der oben definierten Kurve f_1 bewegt. Möchten wir bestimmte Startkoordinaten für den Punkt g vorgeben, so können wir $[f_1]$ durch $[x_1, y_1, f_1]$ ersetzen. Sofern der Punkt $g = (x_1, y_1)$ auf dem Objekt liegt, befindet sich der Glider beim Öffnen dieser Stelle. Andernfalls wird er auf das Objekt f_1 projiziert. Auch hier gibt es wieder unterschiedliche Attribute, ähnlich zu denen eines Punktes. So können wir einen Glider auch mit `visible:false` verstecken, sollte er lediglich zur Konstruktion von anderen Objekten dienen. Die Objekte zu Abbildung C.6 können wir demnach wie folgt konstruieren:

```
var a = board.create('slider',[[1,2],[3,2],[-3,1,3]], {name:'
    A',snapWidth:0.1});
var f1 = board.create('functiongraph', [function(x){return a.
    Value()*Math.sin(x)+1;}],{dash:2});
var g1 = board.create('glider',[0,1,f1]);
```

Zusammenspiel mit STACK

Wir haben bereits Werte an von STACK an Variable im JSXGraph-Block gesendet, wir können aber auch Eingaben aus dem JSXGraph-Applet heraus an Eingabe-Variablen schicken. Dazu müssen wir mit Beginn des JSXGraph-Blocks direkt Angeben, welche Input-Felder dieser Block ausfüllen soll. Hier sei dies das Feld *ans1*. Wir beginnen den Block also folgendermaßen:

```
[[jsxgraph input-ref-ans1="ans1Ref"]]
```

und beenden wie gewohnt mit `[[/jsxgraph]]`. Um den Wert dann festzulegen verwenden wir verschiedene `bind`-Befehle:

```
stack_jxg.bind_list_of(ans1Ref, list)
```

speichert beispielsweise `list` in der STACK-Variablen *ans1*. Daneben gibt es auch die Möglichkeit einen einzelnen Punkt oder `'slider'`-Wert zuzuweisen. Prinzipiell ist sogar ein selbst definierter `bind`-Befehl möglich. Hierzu sei auf die Dokumentation von STACK [SD24] verwiesen. Um Studierende nicht mit möglicherweise kryptischen Eingaben zu verunsichern, können wir das `input`-Feld verbergen, indem wir um das Eingabefeld einen `div`-Container setzen.

```
<div style="display:none">
    [[input:ans1]] [[validation:ans1]]
</div>.
```

Ausserdem sollten wir unter den Antwortoptionen an die Validierung der Antwort abschalten, die ansonsten auch für Verwirrung sorgen könnte. Zudem sollten wir beachten, dass, das JSXGraph-Objekt, dass an diese Eingabe gebunden ist, bewegt worden sein muss, um eine Eingabe an ein Input-Feld zu senden. Sollte auf ein nicht bewegtes Objekt `ob1` zugegriffen werden können, etwa weil die Eingabe optional ist, so ist das mit

```
stack_jxg.starts_moved(ob1)
```

möglich.

Neben den hier vorgestellten Möglichkeiten, finden sich in der Dokumentation zu JSXGraph noch viele weitere Anwendungen. So lassen sich Konstruktionen auch abspielen oder Steuerelemente hinzufügen. Bei weiterführendem Interesse lohnt sich ein Blick in die dort vorgestellten Applets.

Anhang D

STACKrate

Hier wird ein Test mit STACKrate exemplarisch vorgestellt.

Der Test zu Mengenoperationen und Aussagen wird zu Beginn der Veranstaltung Lineare Algebra für Elektrotechnik- und Informatikstudierende zu Übungszwecken gestellt. Die Studierenden können den Test beliebig oft wiederholen. Der Übersicht halber wurden drei Testseiten mit je einer Aufgabe und einem STACKrateblock erstellt.

Gegeben Sind die Mengen $M_1 = \{2, 3, 13, 14, 18, 19\}$ und $M_2 = \{1, 8, 9, 11, 14\}$. Geben Sie die Differenzmenge $M_1 \setminus M_2$ an:

Hinweis: Eine Menge, die a , b und c enthält wird mit **{a,b,c}** angegeben. Die leere Menge mit {}.

Prüfen

Abbildung D.1: Frage 1 zu Mengenoperationen

In der ersten Frage (D.1) muss die Differenzmenge $M_1 \setminus M_2$ zweier Mengen M_1 und M_2 bestimmt werden. Beide Mengen enthalten maximal sechs natürliche Zahlen zwischen 1 und 20. Überprüft wird lediglich, ob die Differenzmenge richtig ist, oder ob versehentlich $M_2 \setminus M_1$ gebildet wurde.

Gegeben Sind die Mengen $M_1 = \{2, 3, 9, 12\}$ und $M_2 = \{4, 6, 7, 8, 10, 16\}$. Geben Sie die Vereinigungsmenge $M_1 \cup M_2$ an:

Hinweis: Eine Menge, die a , b und c enthält wird mit **{a,b,c}** angegeben. Die leere Menge mit {}.

Prüfen

Abbildung D.2: Frage 2 zu Mengenoperationen

In der zweiten (D.2) Frage zu Mengenoperationen wird nach der Vereinigungsmenge $M_1 \cup M_2$ gefragt. Die Mengen sind wie in Frage 1 gewählt und hier wird neben der Korrektheit des Ergebnisses überprüft, ob Vereinigung und Schnitt vertauscht wurden.

Am Ende der Seite erscheint eine weitere STACKfrage, die mit 0 Punkten gewertet wird und in deren Feedback sich die STACKrate-Bewertung verbirgt.

Der Aufbau von Seite 2 ist identisch. Hier wird zunächst nach dem Schnitt zweier Mengen gefragt (D.4).

Nachdem die Grundoperationen bearbeitet wurden, wird in Frage 4 (D.5) eine Menge abgefragt, in der drei Mengen M_1 , M_2 und M_3 zufällig miteinander verknüpft werden. Hierbei werden zwei Verknüpfungen aus den obigen zufällig ausgewählt und auch die Klammerung zufällig gesetzt. Entsprechend können an dieser Stelle sehr unterschiedliche Ausdrücke angezeigt werden (D.6). Um die verschiedenen Varianten sauber anzeigen zu können, benötigen wir hier QuestionBlocks im Fragetext.

Am Ende der Seite folgt wieder eine STACKrate-Frage zu den beiden obigen Aufgaben.

Die nächste Seite beinhalten zwei Fragen zu Aussagenlogik.

Bei der Frage 6 (D.8) handelt es sich um eine Kopie von Frage 5 (D.7).

Dadurch, dass die Werte für die natürlichen Zahlen $p, q \in \mathbb{N}_{\leq 100}$ und $n \in \mathbb{Z}$ mit $-100 \leq n \leq 100$ zufällig gewählt wird, sind die Wahrheitswerte der Aussagen $T(n), G(n), Q(n), P(n)$ zu bestimmen und dann die entsprechenden Verknüpfungen zu bilden. Im Hintergrund werden nun drei Aussagen zufällig ausgewählt und teilweise negiert. Die Klammerung wird gesetzt und die Verknüpfungen jeweils aus der Liste $[\wedge, \vee, \setminus, \dot{\vee}, \Rightarrow, \Leftrightarrow]$ ausgewählt. So sind bei Kopien der Aufgabe jedesmal andere Wahrheitswerte zu erwarten. Zur Anzeige sind hier auch wieder viele Fallunterscheidungen im Fragetext nötig.

Auch hier wird wieder die STACKrate-Frage ans Ende der Seite gestellt.

STACK Frage Dashboard

Liebe Studierende,

wir entwickeln die digitalen Aufgaben kontinuierlich weiter und wollen sie stets verbessern und Ihren Wünschen entsprechend anpassen. Wir sind daher sehr an Ihren Rückmeldungen interessiert und freuen uns über Ihre ehrlichen Bewertungen und Anregungen.

Um die Aufgaben, die Sie auf dieser Seite bearbeitet haben, zu evaluieren, klicken Sie bitte zunächst auf *Prüfen* unter dem Eingabefeld (bitte verändern Sie nichts im Eingabefeld). Anschließend erscheinen die Fragen zur Bewertung.

Vielen Dank für Ihre Mitarbeit!

Bewertung

Aufgabenbewertung

Aufgabenschwierigkeit
Wie haben Sie das Niveau der Aufgabe empfunden?

☆☆☆☆☆
(sehr leicht -- angemessen -- sehr schwer)

Usability
Wie gut sind Sie mit der Eingabe zurecht gekommen?

☆☆☆☆☆
(sehr schlecht -- sehr gut)

Aufgabenbearbeitung
Wie haben Sie den Bearbeitungsprozess empfunden?

☆☆☆☆☆
(langweilig -- interessant)

Aufgabenfeedback
Für wie hilfreich haben Sie das Feedback empfunden, welches Sie während der Bearbeitung der Aufgabe erhalten haben?

☆☆☆☆☆
(gar nicht hilfreich -- sehr hilfreich)

Bemerkungen:

Abbildung D.3: Bitte um Rückmeldung der Studierenden

Gegeben Sind die Mengen $M_1 = \{1, 4, 6, 19\}$ und $M_2 = \{2, 5, 8, 14, 15, 19\}$. Geben Sie den Schnitt $M_1 \cap M_2$ an:

Hinweis: Eine Menge, die a , b und c enthält wird mit **(a,b,c)** angegeben. Die leere Menge mit {}.

Prüfen

Abbildung D.4: Frage 3 zu Mengenoperationen

Gegeben sind die Mengen $M_1 = \{3, 6, 10, 13, 16, 19\}$, $M_2 = \{1, 2, 7, 9, 12, 19\}$ und $M_3 = \{1, 12, 13, 14, 18, 19\}$. Geben Sie $(M_1 \cap M_2) \cap M_3$ an.

Hinweis: Eine Menge, die a , b und c enthält wird mit **{a,b,c}** angegeben. Die leere Menge mit $\emptyset$.

Prüfen

Abbildung D.5: Frage 4 zu Mengenoperationen

Frage-Tests und eingesetzte Varianten

Gegeben sind die Mengen $M_1 = \{1, 8, 14, 15, 17\}$, $M_2 = \{3, 5, 6, 7, 10\}$ und $M_3 = \{3, 7, 13, 14, 19, 20\}$. Geben Sie $(M_1 \cup M_2) \setminus M_3$ an.

Hinweis: Eine Menge, die a , b und c enthält wird mit **{a,b,c}** angegeben. Die leere Menge mit $\emptyset$.

Abbildung D.6: Eine zweite Variante der Frage 4

Gegeben sind folgende Aussageformen:

$T(p, q)$: p teilt q
 $G(p, q)$: p ist größer als q
 $Q(n)$: n ist Quadratzahl
 $P(n)$: n ist positiv

Bestimmen Sie die Wahrheitsgehalt der Aussage

$$A(n, p, q) = (\neg P(n) \vee G(p, q)) \Rightarrow \neg T(p, q)$$

und

$$B(n, p, q) = \neg Q(n) \Rightarrow (G(p, q) \vee T(p, q))$$

für die Werte $n = 25$, $p = 75$ und $q = 56$:

Die Aussage $A(n, p, q)$ ist:

Die Aussage $B(n, p, q)$ ist:

Prüfen

Abbildung D.7: Frage 5 zu Aussagenlogik

Gegeben sind folgende Aussageformen:

$T(p, q)$: p teilt q

$G(p, q)$: p ist größer als q

$Q(n)$: n ist Quadratzahl

$P(n)$: n ist positiv

Bestimmen Sie die Wahrheitsgehalt der Aussage

$$A(n, p, q) = (\neg Q(n) \Rightarrow \neg G(p, q)) \Rightarrow \neg P(n)$$

und

$$B(n, p, q) = (T(p, q) \Rightarrow \neg P(n)) \Rightarrow \neg G(p, q)$$

für die Werte $n = 73$, $p = 29$ und $q = 74$:

Die Aussage $A(n, p, q)$ ist:

Die Aussage $B(n, p, q)$ ist:

Prüfen

Abbildung D.8: Frage 6 zu Aussagenlogik