MATLAB
in Augenoptik und Hörakustik

Integration von MATLAB in das Studium der Augenoptik und Hörakustik an der Hochschule Aalen durch den Aufbau einer e-Learning-Plattform für die Lehrveranstaltung Methoden der Mathematik Augenoptik und Hörakustik

Bachelorarbeit

im Studiengang Augenoptik/Augenoptik und Hörakustik

vorgelegt von:	Judith Ungewiß
Studiengang:	Augenoptik/Augenoptik und Hörakustik
Matrikelnummer:	30090
Erstgutachter:	Prof. Dr. Ulrike Paffrath
Zweitgutachter:	Prof. Dr. Jürgen Nolting
eingereicht am:	4. Oktober 2013

Abstract

Im Rahmen dieser Bachelorarbeit „MATLAB in Augenoptik und Hörakustik“ wird eine Plattform entwickelt, die die Studierenden im Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen dabei unterstützt, sich mathematische Inhalte selbst zu erarbeiten und den Umgang mit MATLAB zu erlernen. Mithilfe von MATLAB wird die Anwendung mathematischer Grundlagen auf augenoptische und hörakustische Inhalte praxisorientiert umsetzt.

Um dies erreichen zu können, werden die notwendigen Vorüberlegungen angestellt. Hier wird zunächst auf den Begriff des e-Learning eingegangen, anschließend wird in einem Vergleich festgestellt, dass andere Hochschulen mit fachlich ähnlichen Bachelor-Studiengängen der Augenoptik/Optometrie in Deutschland MATLAB bisher nicht in ihre Lehre integriert haben. Fachverwandte Bachelor-Studiengänge wie solche der Feinwerktechnik oder Optoelektronik/Lasertechnik hingegen lehren durchgängig MATLAB oder andere fachlich relevante Programmiersprachen. Zudem wird mithilfe einer Befragung der Studierenden im Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen festgestellt, dass ein grundsätzliches Interesse der Studierenden an der Thematik MATLAB vorliegt.
Nun werden die Skripte, die den Studierenden zur selbstständigen Erarbeitung der Themen „Lineare Gleichungssysteme und Matrizen“ sowie „Mehrdimensionale Integration“ zur Verfügung gestellt werden, vorgestellt. Anschließend wird dargestellt, in welcher Form die Integration der Lehre von MATLAB in die Lehrveranstaltung Methoden der Mathematik Augenoptik und Hörakustik vonstatten gehen kann und welche Möglichkeiten hinsichtlich der Umsetzung dessen bestehen.
Es wird veranschaulicht, wie MATLAB zur praxisorientierten Unterstützung der Lehre in anderen Lehrveranstaltungen genutzt werden kann. Beispielhaft werden hierbei zwei Programme gezeigt, die sich mit der Berechnung eines FIR-Filters und der Beugung an der Kreisblende befassen. Damit findet die Anwendung von MATLAB auf augenoptische und hörakustische Sachverhalte statt. Schließlich wird erläutert, wie die in dieser Arbeit konzipierte e-Learning-Plattform bei Bedarf zukünftig erweitert werden kann.

Danksagung

Zunächst bedanke ich mich bei allen Studierenden der Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen, die mich durch die Teilnahme an der durchgeführten Umfrage bei meiner Bachelorarbeit unterstützt haben. Durch sie habe ich wertvolle Hinweise zur Konzeption der vorgesehenen e-Learning-Plattform erhalten können.

Besonders danke ich Frau Prof. Dr. Paffrath, die diese Arbeit sehr geduldig betreut hat und mir bei Schwierigkeiten jederzeit zur Seite stand. Ihre Vorschläge und Ideen haben mir die Arbeit erheblich erleichtert.

Ebenso gilt mein Dank Herrn Prof. Dr. Nolting für die Übernahme des Zweitgutachtens und die damit verbundenen Mühen.

Mein Dank gilt jedoch auch meinen Eltern, welche mich das ganze Studium hindurch unterstützt haben all denen, die mir immer mit Rat und Tat zur Seite standen.

Inhaltsverzeichnis

Abstract II

Danksagung III

Abbildungsverzeichnis VI

Tabellenverzeichnis VIII

Abkürzungsverzeichnis IX

1. Einleitung 1

1.1. Problemstellung 1
1.2. Zielsetzung 2
1.3. Vorgehensweise 2

2. Vorüberlegungen und Hintergründe 4

2.1. E-Learning im Kontext der Lerntheorie 5
2.2. MATLAB® als zentrale Software im Gesamtkonzept der e-Learning-Plattform 7
2.3. MATLAB in Bachelor-Studiengängen an anderen Hochschulen 7
2.4. Das Interesse der Studierenden an MATLAB: Ergebnisse einer Umfrage . . . 9

3. Konzeption der e-Learning-Plattform 18

3.1. Selbststudium 19
3.1.1. Themenschwerpunkt Lineare Gleichungssysteme und Matrizen 19
3.1.2. Themenschwerpunkt Mehrdimensionale Integration 20
3.2. Integration von MATLAB in die Lehrveranstaltung Mathematik 20
3.2.1. Praktikum MATLAB - Teil 1: Direkte Eingabe 21
3.2.2. Praktikum MATLAB - Teil 2: Indirekte Eingabe 22
3.3. MATLAB im Gesamtkontext des Studiums: Augenoptisch und hörakustisch relevante Anwendungen 23

3.3.1. FIR-Filter als Beispiel der hörakustisch relevanten Anwendung von MATLAB 25
3.3.2. Die Beugung an der Kreisblende als Beispiel der augenoptisch relevanten Anwendung von MATLAB 26

4. Fazit **28**

5. Literatur **30**

6. Eidesstattliche Erklärung **33**

Anhang 34

A. Fragebogen zur Umfrage: MATLAB in Mathematik **34**

B. Übersicht über die Ergebnisse der Umfrage **37**

C. Skript zum Themenschwerpunkt Lineare Gleichungssysteme und Matrizen **41**

D. Skript zum Themenschwerpunkt Mehrdimensionale Integration **58**

E. Skript zum Praktikum MATLAB - Teil 1: Direkte Eingabe **67**

F. Skript zum Praktikum MATLAB - Teil 2: Indirekte Eingabe **91**

G. Skript zur MATLAB-Anwendung: FIR-Filter **111**

H. Programmcode zur MATLAB-Anwendung: FIR-Filter **115**

I. Skript zur MATLAB-Anwendung: Beugung an der Kreisblende **127**

J. Programmcode zur MATLAB-Anwendung: Beugung an der Kreisblende **132**

Abbildungsverzeichnis

2.1. Umfrage, Ergebnis zu Frage 2: Wo möchtest Du nach dem Studium am liebsten arbeiten? . . . 11
2.2. Umfrage, Ergebnis zu Frage 3: Denkst Du, dass es sinnvoll wäre, während des Studiums (mehr) mit MATLAB zu arbeiten? . . . 12
2.3. Umfrage, Ergebnis zu Frage 4: Hältst Du den Praxisbezug des Studiums für ausreichend? . . . 12
2.4. Umfrage, Ergebnis zu Frage 5: Denkst Du, dass durch die Möglichkeit, während des Studiums MATLAB zu lernen, der Praxisbezug des Studiums erhöht werden könnte? . . . 13
2.5. Umfrage, Ergebnis zu Frage 6: Fändest Du es sinnvoll, wenn bereits im ersten Semester, begleitend zur Vorlesung Mathematik, eine Einführung in MATLAB stattfinden würde? . . . 14
2.6. Umfrage, Ergebnis zu Frage 7: Würdest Du ein zusätzlich angebotenes Wahlfach, in dem MATLAB thematisiert wird, wählen? . . . 14
2.7. Umfrage, Ergebnis zu Frage 8: Wenn Du die Wahl zwischen MATLAB und anderen Programmiertools/-sprachen hättest, was würdest Du wählen? . . . 15

3.1. Programm zum Thema FIR-Filter (Bandpass): Eine hörakustisch relevante Anwendung von MATLAB . . . 26
3.2. MATLAB-Programm zum Thema Beugung an der Kreisblende: Eine augenoptisch relevante Anwendung von MATLAB . . . 27

D.1. Polarkoordinaten . . . 61
D.2. Halbkreisring . . . 61
D.3. Kugelkoordinaten . . . 62
D.4. Raumwinkelberechnung . . . 64

E.1. MATLAB-Desktop nach dem Start (Studenten-Version R 2012a) . . . 69
E.2. MATLAB-Command Window nach dem Start (Studenten-Version R 2012a) . 70

E.3. Current Folder bzw. Current Directory (Studenten-Version R 2012a) 71
E.4. MATLAB-Workspace mit den momentan verfügbaren Variablen (Studenten-Version R 2012a) 72
E.5. MATLAB-Variable Editor mit dem Inhalt der Matrix x, bestehend aus 5 Spalten und 4 Zeilen (Studenten-Version R 2012a) 73
E.6. Command History mit Wiedergabe der zuletzt eingegebenen MATLAB-Befehle (Studenten-Version R 2012a) 74

G.1. MATLAB-Programm zum Thema FIR-Filter (Bandpass) 114

I.1. MATLAB-Programm zum Thema Beugung an der Kreisblende 131

Tabellenverzeichnis

B.1. Ergebnisse der Umfrage: MATLAB in Mathematik 38

E.1. Übersicht über nützliche Kommandos 78
E.2. Übersicht über arithmetische Operationen 80
E.3. Übersicht über spezielle Werte 80
E.4. Übersicht über häufig benötigte Funktionen 81
E.5. Übersicht über Funktionen der komplexen Zahlen 82
E.6. Übersicht zur Matrizenrechnung 82
E.7. Übersicht über Befehle für Grafiken 84
E.8. Übersicht: Symbolische Rechnungen 86

F.1. Übersicht über die Eigenschaften einer Grafik 94
F.2. Übersicht über Farbenwerte, Punkt- und Linientypen 94
F.3. Übersicht über die Farbpaletten 95
F.4. Übersicht über Grafiktypen 96
F.5. Übersicht über wichtige Befehle 100

Abkürzungsverzeichnis

A	Augenoptik
Abb.	Abbildung
AH	Augenoptik & Hörakustik
B.Eng.	Bachelor of Engineering
B.Sc.	Bachelor of Science
bzw.	beziehungsweise
CAD	Computer Aided Design
cd	Candela
CL	Contactlinse
DFT	Discrete Fourier Transformation
d.h.	das heißt
DSV	Digitale Signalverarbeitung
etc.	et cetera
FFT	Fast Fourier Transformation
FIR	Finite Impulse Response
HTML	Hypertext Markup Language
Hz	Hertz
IIR	Infinite-Impulse-Response
KL	Kontaktlinse
LGS	Lineares Gleichungssystem, Lineare Gleichungssysteme
lm	Lumen
o.g.	oben genannt
ProE	ProEngineer
rad	Radiant
s.	siehe
S.	Seite
sr	Steradiant
vgl.	vergleiche

1. Einleitung

Der Lehre der Mathematik im Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen kommt die Bedeutung eines Grundsteins zu, der zu Beginn des Studiums gelegt wird, um ein Verständnis verschiedener optischer und akustischer Inhalte gewährleisten zu können. Der Erfolg der Studierenden in der Lehrveranstaltung Methoden der Mathematik Augenoptik und Hörakustik, nachfolgend der besseren Lesbarkeit der Arbeit wegen vereinfacht als Lehrveranstaltung Mathematik bezeichnet, kann sich deshalb auf ihren Erfolg in einigen weiteren Lehrveranstaltungen und sogar in ganzen Modulen auswirken. Aufgrund dieser Tatsache ist die Vermittlung der Inhalte der Lehrveranstaltung Mathematik essentiell für die Studierenden.

1.1. Problemstellung

Momentan wird die Lehre im Bezug auf die Lehrveranstaltung Mathematik in Form einer Vorlesung umgesetzt. Ergänzend werden online, über moodle, Übungsaufgaben zur Verfügung gestellt, in einem zusätzlich angebotenen Tutorium besprochen werden. Auch erarbeiten die Studierenden sich bestimmte mathematische Grundlagen eigenständig. Das Erlernen eines Programms oder einer Programmiersprache, wie beispielsweise MATLAB, die bei der Bearbeitung mathematischer Probleme hilfreich sein können, ist bisher nicht Inhalt der Lehrveranstaltung Mathematik.
Aktuell stehen zur effizienten eigenständigen Erarbeitung von Themenkomplexen der Lehrveranstaltung Mathematik keine Unterlagen zur Verfügung, die zeigen, welche Schwerpunkte für Studierende der Augenoptik/Augenoptik und Hörakustik von Bedeutung sind.

Es besteht ein großes Angebot an Literatur zu MATLAB, sowohl in deutscher, als auch in englischer Sprache, womit die Studierenden sich bei Interesse die Grundlagen des Umgangs mit MATLAB selbst erarbeiten könnten. Jedoch ist die vorliegende Literatur durchweg sehr umfangreich gestaltet und nicht auf die Bedürfnisse des Studiengangs Augenoptik/Augenoptik und Hörakustik zugeschnitten, weshalb ein effizientes Erlernen der angesprochenen Grundla-

gen nicht möglich ist. Auch Programme, die MATLAB auf augenoptische und hörakustische Sachverhalte anwenden und solche darstellen, wodurch sie Studierenden zur Übung dienen können, bestehen in diesem Sinne bislang nicht. Dies soll mit dieser Arbeit geändert werden.

1.2. Zielsetzung

Ziel dieser Arbeit ist, auf der Grundlage der bereits online zur Verfügung stehenden Übungsaufgaben zur Lehrveranstaltung Mathematik, eine e-Learning-Plattform zu dieser Lehrveranstaltung im Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen aufzubauen. Diese soll die Studierenden zum Einen dabei unterstützen, sich mathematische Inhalte eigenständig anzueignen. Zum Anderen sollen die Studierenden den Umgang mit einer Software erlernen, die ebendies unterstützen und die Lösung mathematischer und technischer Probleme vereinfachen kann. Hierzu wird in der vorliegenden Arbeit MATLAB, ein weltweit verbreitetes Programm zur Berechnung und zur grafischen Darstellung mathematischer und technischer Probleme, gewählt. Auch soll den Studierenden gezeigt werden, wie sich das so Erlernte auf augenoptische und hörakustische Inhalte anderer Lehrveranstaltungen anwenden lässt, um zu zeigen, dass MATLAB während des Studiums in vielerlei Hinsicht hilfreich sein kann.
Insgesamt soll so das Lernverhalten der Studierenden unterstützt und ihre Motivation hinsichtlich der Inhalte der Lehrveranstaltung Mathematik sowie der Lösung mathematischer und technischer Probleme mithilfe eigener Programme gesteigert werden.

1.3. Vorgehensweise

Um die gesteckten Ziele zu erreichen, wird eine e-Learning-Plattform konzipiert, die sich im Wesentlichen auf drei Säulen stützt:
Als erste Säule werden Skripte auf elektronischem Weg zur Verfügung gestellt, mit denen die Studierenden sich bestimmte Themen selbstständig erarbeiten können. Die zweite Säule besteht im Erlernen des Umgangs mit MATLAB. Dies soll zwar betreut stattfinden, allerdings sollen die Studierenden auch hier selbstständig arbeiten, weshalb entsprechende Aufgaben zur Verfügung gestellt werden. Die dritte Säule ist die Anwendung des in der Lehrveranstaltung Mathematik Gelernten in augenoptischen oder hörakustischen Sachverhalten. Auch hier können die Studierenden mithilfe von Programmen, die zur Verfügung gestellt werden, selbstständig lernen.

Zur Konzeption dieser Plattform befasst diese Arbeit sich zunächst mit dem Begriff und der Definition des e-Learning und begründet anschließend die Wahl von MATLAB als wesentlichen Bestandteil der aufzubauenden e-Learning-Plattform. Nun wird untersucht, wie andere Bachelor-Studiengänge in Deutschland, die sich mit Augenoptik, Optometrie, Hörakustik und fachlich verwandten Disziplinen befassen, MATLAB aktuell in die Lehre einbeziehen. Auch wird unter Studierenden der Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen eine Umfrage zum Thema MATLAB und dessen Integration in die Lehrveranstaltung Mathematik durchgeführt, mit der festgestellt werden soll, ob die Studierenden überhaupt Interesse an MATLAB haben. Auch werden die Studierenden hierbei gebeten, selbst Vorschläge zu machen, wie MATLAB in das Studium der Augenoptik/Augenoptik und Hörakustik eingebunden werden kann.

Anschließend werden zu den Themen „Lineare Gleichungssysteme und Matrizen“ sowie „Mehrdimensionale Integration“, welche im Rahmen der Lehrveranstaltung Mathematik von den Studierenden selbstständig erarbeitet werden sollen, Skripte erarbeitet, die den Studierenden zukünftig zur Verfügung gestellt werden.

Weiterhin wird dargelegt, wie die Studierenden den Umgang mit MATLAB erlernen werden. Das hierzu in zwei Teilen zur Verfügung gestellte Skript wird gemeinsam mit Aufgaben und zugehörigen Lösungen, mithilfe derer die Studierenden den Umgang mit MATLAB üben und vertiefen können, besprochen. Nun wenden die Studierenden die Inhalte, die in der Lehrveranstaltung Mathematik vermittelt werden, praktisch an. Dazu werden beispielhaft zwei MATLAB-Programme gezeigt. Zum Einen kann die mathematische Beschreibung eines FIR-Filters grafisch nachvollzogen werden, zum Anderen kann die entstehende Beugungsfigur bei der Beugung an einer Kreisblende in Abhängigkeit von der Größe der entsprechenden Blende begutachtet werden.
Weitere Möglichkeiten zur praktischen Anwendung mathematischer Inhalte in Augenoptik und Hörakustik werden aufgezeigt. Auch Ideen zur zukünftigen Nutzung und zum Ausbau der e-Learning-Plattform werden vorgestellt.

Schließlich werden die Vorüberlegungen und die tatsächlich erarbeitete praktische Umsetzung der e-Learning-Plattform mit ihren drei Säulen, kritisch gegenübergestellt. Dabei wird analysiert, inwiefern die Plattform bereits an dieser Stelle Erfolg verspricht und inwiefern eine Weiterentwicklung stattfinden kann.

2. Vorüberlegungen und Hintergründe

Um eine e-Learning-Plattform für den Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen aufbauen zu können, müssen zunächst einige Vorüberlegungen durchgeführt werden. Hierbei ist festzustellen, wie der Begriff des e-Learning überhaupt zu verstehen ist, um diesem Rechnung tragen zu können, und welche Vorteile e-Learning gegenüber konventionellen Lernmethoden haben kann.

Zudem wird MATLAB, als ausgewähltes Programm und zentraler Bestandteil des hier erarbeiteten Konzeptes, kurz vorgestellt.

Da diese Arbeit eine Integration von MATLAB in die Lehrveranstaltung Mathematik und somit auch in die Arbeit der Studierenden, die mithilfe des e-Learning gefördert werden soll, vorsieht, wird untersucht, ob andere Hochschulen in Deutschland, die ähnliche Studiengänge anbieten, MATLAB bereits in die dortige Lehre integriert haben. Nach momentanem Kenntnisstand liegen bislang keine solchen Untersuchungen und Vergleiche der verschiedenen augenoptischen, hörakustischen und fachverwandten Studiengänge in Deutschland im Hinblick auf diesen Aspekt vor. Hierbei kann festgestellt werden, ob die Integration von MATLAB in die Lehrveranstaltung Mathematik den Studierenden der Hochschule Aalen mögliche Vorteile gegenüber den Absolventen anderer, fachlich ähnlicher, Studiengänge in Deutschland bietet oder ob eine Einbindung von MATLAB sogar notwendig ist, um die Konkurrenzfähigkeit der Absolventen der Hochschule Aalen gegenüber derjenigen anderer Hochschulen, insbesondere im Hinblick auf berufliche Perspektiven nach Abschluss des Studiums, gewährleisten zu können.

Zudem werden Studierende des Studiengangs Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen mittels einer Umfrage um ihre Meinung zur Integration von MATLAB in die Lehrveranstaltung Mathematik und somit in das Studium gebeten. Hier wird zum Einen das Interesse der Studierenden an MATLAB erfragt, zum Anderen werden Erkenntnisse bezüglich der Wünsche der Studierenden, die praktische Umsetzung der Integration von MATLAB in das Studium betreffend, gewonnen.

Die so gewonnenen Erkenntnisse dienen als Grundlage zum Aufbau der e-Learning-Plattform, der Inhalt dieser Arbeit ist, und werden hierbei dementsprechend berücksichtigt.

2.1. E-Learning im Kontext der Lerntheorie

Die Studierenden sollen mithilfe der in dieser Arbeit vorgestellten Anleitungen in der Lage sein, sich die entsprechenden Themenkomplexe weitgehend eigenständig, unterstützt durch e-Learning, erarbeiten zu können. Deshalb ist zunächst zu klären, welche Vorteile ein eigenständiges und selbstgesteuertes Lernen für die Studieren überhaupt bringt. Weiterhin ist der Begriff des e-Learning dazu genauer zu definieren und zu erörtern. Auch ist festzulegen, wie eine grundsätzliche Struktur auszusehen hat, um den Studierenden die betreffende Thematik effektiv, also verständlich und nachhaltig, beizubringen.

Eine wichtige Voraussetzung für die Studierenden, um selbstgesteuert lernen zu können, ist, dass sie an der gegebenen Thematik Interesse entwickeln und zum Lernen motiviert sind. Zudem müssen die Studierenden in der Lage sein, Wissen, Fertigkeiten und Einstellungen entwickeln zu können, die auch ihr zukünftiges Lernen fördern und erleichtern. Durch die Möglichkeit des eigenständigen Lernens erhalten die Studierenden die Gelegenheit, diese Fertigkeiten und Einstellungen weiterzuentwickeln. [1]

Den Begriff des e-Learning exakt zu abzugrenzen, ist nicht möglich, da in der Literatur eine Vielzahl verschiedener Definitionen besteht. Die vorliegende Arbeit hält sich an das von Bauer und Philippi bereits 2001 dargelegte Verständnis des Begriffs.

Demnach gehört zu e-Learning jeder Lernprozess, der sich durch ein gleichzeitiges Erfüllen der folgenden vier Merkmale auszeichnet.

- Nutzung multimedialer Technologien
- Angebot von autonomem Lernen
- Möglichkeit von persönlicher Betreuung
- Nutzung von elektronischen Daten- und Kommunikationsnetzen [2]

Wesentlich bei der Nutzung multimedialer Technologien ist, dass die eingebrachten Inhalte in einer Art strukturiert und visualisiert werden, die auf anderem Weg nicht zu realisieren wäre. So sollten beispielsweise Grafiken, Texte und Animationen zu einem gelungenen e-Learning-Angebot verbunden werden. [3]

Beim e-Learning ist die Eigenschaft der Lernautonomie von besonderer Bedeutung. Autonomie heißt, dass der Lernende seinen Lernprozess selbst bestimmen und dadurch entscheiden kann, wie schnell er vorangehen möchte, welche Inhalte ihm möglicherweise bereits bekannt sind und deshalb übersprungen werden können und welche er, etwa aufgrund von Verständnisschwierigkeiten, wiederholen möchte. Auch hat der Lernende Handlungsoptionen und kann beispielweise die Abfolge verschiedener Lernschritte selbst bestimmen. Das e-Learning bietet so verschiedene Möglichkeiten, welche davon aber konkret realisiert werden, hängt vom Lernenden selbst ab. [4]

Zudem sollte das e-Learning nicht nur als technologische, sondern auch als kommunikative Möglichkeit angesehen werden. So ist das e-Learning als Verbund anzusehen, der sich sowohl aus Lernenden als auch aus einem oder mehreren Betreuer/n zusammensetzt. [5]

Die Nutzung elektronischer Daten- und Kommunikationsnetze erleichtert die Kommunikation zwischen den Lernendem und Betreuer und dienen dazu, den Lernenden die jeweils benötigten Materialien und Informationsquellen zur Verfügung zu stellen. Sie ersetzt nicht die bereits angesprochene Betreuung durch eine kompetente reale Person, unterstützen diese aber bei der Vermittlung der entsprechenden Inhalte. [6]

Wie kann man den Studierenden die vorliegende Thematik nun effektiv nahebringen? Zunächst ist davon auszugehen, dass zum Verstehen von Zusammenhängen, wie hier gefordert, das Lernen am Beispiel als didaktisch kluge Maßnahme angesehen werden kann. Allerdings muss diese auch vernünftig angewendet werden. So kann ein Verstehen der Thematik nur dann erfolgen, wenn nicht nur Beispiele, also Einzelfälle eines allgemeinen Sachverhalts, vorgestellt werden. Vielmehr muss ein Einzelfall zwar prinzipiell gegeben sein, jedoch müssen ebenso ein allgemeiner Zusammenhang und schließlich eine Beziehung, die den Einzelfall zum Beispiel für den allgemeinen Zusammenhang macht, dargelegt werden. [7]

Zusammenfassend ist also festzustellen, dass die Studierenden zum eigenständigen, aber betreuten Lernen angeleitet werden sollen. Dabei wird den oben genannten Erkenntnissen, wie die effektiv vonstatten gehen kann, ebenso Rechnung getragen wie den einzelnen Aspekten, die das e-Learning charakterisieren. Die Umsetzung eines solchen Konzeptes hinsichtlich der Thematik MATLAB innerhalb der Lehrveranstaltung Mathematik im Studiengang Augenoptik/Augenoptik und Hörakustik ist Gegenstand dieser Arbeit und wird in Kapitel 3. Konzeption der e-Learning Plattform erläutert.

2.2. MATLAB® als zentrale Software im Gesamtkonzept der e-Learning-Plattform

MATLAB® ist ein nummerisches Berechnungs- und Simulationswerkzeug. Auf die Computeralgebra-Funktionalität kann allerdings innerhalb der MATLAB-Umgebung über eine integrierte „Symbolics"-Toolbox zugegriffen werden. Die grundsätzliche Struktur, auf der alle MATLAB-Operationen beruhen, ist jedoch das nummerische Feld, also eine Matrix. Daher auch der Name MATLAB, welcher eine Abkürzung für MATrix LABoratory ist.
MATLAB bietet damit gleichzeitig die Möglichkeiten nummerischer und algebraischer Berechnungen. Zudem kann es als interaktive Berechnungsumgebung und über den Aufruf von selbst geschriebenen Programmen genutzt werden. Aufgrund dieser Möglichkeiten wird MATLAB als sehr geeignet für die Studierenden eingestuft und für den Aufbau der e-Learning-Plattform im Rahmen dieser Arbeit ausgewählt. Mit MATLAB können mathematische und technische Probleme dargestellt und gelöst werden, wodurch die Studierenden ein Verständnis der thematisierten Inhalte erreichen, welches ohne die Nutzung von MATLAB in dieser Form nicht möglich wäre.

MATLAB wurde ursprünglich aus der Sprache FORTRAN entwickelt und wird von der Firma *The MathWorks, Inc.* vertrieben. Neben der Vollversion wird eine eingeschränkte Student Version angeboten. Diese wird, mit dem Release 2012a, zur Erstellung der MATLAB-Anwendungen dieser Arbeit verwendet.
Zudem wird MATLAB®Mobile™ angeboten. Hierbei handelt es sich um eine App, mit der von mobilen Geräten aus auf MATLAB zugegriffen werden kann, sofern der Benutzer im Besitz einer entsprechenden Lizenz ist. Die App wird für die Betriebssysteme iOS 5.0 oder höher und Android angeboten.

2.3. MATLAB in Bachelor-Studiengängen an anderen Hochschulen

Nun stellt sich die Frage, ob eine Einführung in MATLAB in das Studium der Augenoptik/Augenoptik und Hörakustik notwendig ist, um eine Konkurrenzfähigkeit von Aalener Absolventen mit Studierenden anderer Fachhochschulen, an denen ähnliche Studiengänge angeboten werden, zu erreichen. Dazu wird mithilfe der Modulhandbücher ausgewählter Bachelor-Studiengänge an anderen Fachhochschulen festgestellt, ob Matlab dort bereits in das Studium eingebunden ist. Eine solche Untersuchung liegt bislang nicht vor. Es werden

sowohl die deutschlandweit angebotenen Studiengänge der Augenoptik und/oder Optometrie in Berlin, Jena, Lübeck, München und Wolfsburg, als auch fachlich verwandte Studiengänge untersucht. Ein Anspruch auf Vollständigkeit bezüglich der deutschlandweit bestehenden Studiengänge wird nicht gestellt.

Den Modulhandbüchern ist zu entnehmen, dass im Studiengang Augenoptik/Optometrie (B.Sc.) an der Beuth Hochschule für Technik Berlin [8], im Studiengang Augenoptik/Optometrie (B.Sc.) an der Ernst-Abbe-Fachhochschule Jena [9] sowie im Studiengang Augenoptik (B.Sc.) an der Ostfalia Hochschule für angewandte Wissenschaften in Wolfsburg [10] kein MATLAB gelehrt wird. An der Fachhochschule Lübeck wird im Studiengang Augenoptik/Optometrie (B.Sc.) kein Matlab gelehrt [11], im Studiengang Hörakustik (B.Sc.) [12] findet hingegen eine Lehrveranstaltung „MATLAB“, bewertet mit 4 CP, innerhalb des Moduls „Statistik und Programmieren“ im zweiten Fachsemester statt. Im Studiengang Augenoptik/Optometrie (B.Sc.) an der Hochschule für angewandte Wissenschaften München [13] besteht ein Modul „Informatik“ mit dem Inhalt „Programmiersprachen“, letztere werden jedoch nicht genauer spezifiziert, weshalb fraglich bleibt, ob MATLAB in diesem Studiengang Lehrinhalt ist.

Als fachlich verwandte Studiengänge werden Optoelektronik, Lasertechnik, Präzisionsmaschinenbau und Feinwerktechnik angesehen. Im Studiengang Optoelektronik/Lasertechnik (B.Sc.) an der Hochschule Aalen [14] wird MATLAB bereits im ersten Fachsemester gelehrt, an der Hochschule Koblenz, Studiengang Optik und Lasertechnik (B.Sc.) [15] findet ein Modul „Informatik“ im ersten und zweiten Fachsemester statt, in dem unter anderem MATLAB thematisiert wird. An der Hochschule Esslingen wird der Studiengang Mechatronik/Feinwerk- und Mikrotechnik angeboten, hier wird kein MATLAB gelehrt, allerdings ist das Erlernen von C Inhalt des Studiums[16]. Im Studiengang Präzisionsmaschinenbau (B.Sc.) an der Hochschule für angewandte Wissenschaft und Kunst Göttingen [17] wird ebenfalls kein MATLAB, jedoch ab dem zweiten Fachsemester C sowie C++ gelehrt, ebenso im Studiengang Mechatronik/Feinwerktechnik (B.Eng.) an der Hochschule für angewandte Wissenschaften München, allerdings hier im dritten Fachsemester [18]. An der Ernst-Abbe-Fachhochschule Jena werden die Studiengänge Feinwerktechnik/Precision Engineering (B.Eng.) sowie Laser- und Optotechnologien (B.Eng.) angeboten. Im erstgenannten Studiengang wird MATLAB nicht gelehrt [19], jedoch sind unter anderem MATLAB, C und C++ ab dem ersten Fachsemester Studieninhalte in letztgenanntem Studiengang [20]. Die Technische Hochschule Nürnberg bietet den Studiengang Mechatronik/Feinwerktechnik (B.Eng.) an, hier wird MATLAB nicht thematisiert, jedoch wird C gelehrt [21].

Zusammenfassend ist festzustellen, dass die in Deutschland angebotenen Studiengänge der Augenoptik/Optometrie MATLAB bisher inhaltlich nicht in das Studium integriert haben, es im Studiengang Hörakustik allerdings fester Bestandteil des Bachelor-Studiums ist. Die o.g. verwandten Studiengänge, die etwas technischer orientiert sind als der Studiengang Augenoptik/Augenoptik und Hörakustik, lehren durchgehend MATLAB oder C.

Aus diesen Feststellungen ist abzuleiten, dass diejenigen Studierenden der Augenoptik, die ein Interesse an einer zukünftigen Beschäftigung in Forschung und Industrie haben, durch die Einbindung von MATLAB in das Studium einen Vorteil gegenüber den Absolventen der Augenoptik/Optometrie an anderen Hochschulen erlangen könnten. Die Lehre von MATLAB wäre somit auch ein Alleinstellungsmerkmal für den Bachelor-Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen. Für Studierende der Augenoptik und Hörakustik, die sich insbesondere für die Hörakustik interessieren, ist die Einbindung von MATLAB in das Studium essentiell, um gegenüber den Absolventen des Bachelor-Studiengangs Hörakustik an der Fachhochschule Lübeck keine Wettbewerbsnachteile zu haben. Auch Studierende, die sich für einen weiterführenden Masterstudiengang in einem verwandten Fachgebiet interessieren, ist es wichtig, MATLAB bereits im Bachelor-Studium kennengelernt zu haben, um den im Master-Studium gestellten Anforderungen gerecht werden zu können, da hier davon ausgegangen werden muss, dass diese Master-Studiengänge die in den entsprechenden Bachelor-Studiengängen (beispielsweise Optoelektronik/Lasertechnik oder Feinwerktechnik) gelehrten Inhalte voraussetzen.

Sowohl die genannten Kriterien der Konkurrenzfähigkeit gegenüber Absolventen der Augenoptik und Hörakustik anderer Hochschulen und der Absolventen fachverwandter Studiengänge als auch die Tatsache, dass die Lehre von MATLAB im Studiengang Augenoptik/Augenoptik und Hörakustik, Studienschwerpunkt Augenoptik für die Hochschule Aalen ein Alleinstellungsmerkmal bedeuten könnte, sprechen dafür, MATLAB zukünftig in das Studium der Augenoptik/Augenoptik und Hörakustik zu integrieren.

2.4. Das Interesse der Studierenden an MATLAB: Ergebnisse einer Umfrage

Um Erkenntnisse über das grundsätzliche Interesse der Studierenden an MATLAB zu gewinnen, wird unter den Studierenden im sechsten und siebten Fachsemester eine Umfrage mit dem Titel „MATLAB in Mathematik“ durchgeführt. Hierbei werden bewusst Studierende

in höheren Fachsemestern befragt, da von diesen erwartet werden kann, die Anforderungen des Studiums bereits aus eigener Erfahrung zu kennen und so auch einschätzen zu können, inwiefern eine Einführung in MATLAB im Rahmen des Studiums von Nutzen sein kann. Um dies besser einschätzen zu können, werden die Studierenden von Durchführung der Umfrage kurz über das Programmiertool MATLAB und dessen mögliche Anwendungen informiert. Der vollständige Fragebogen befindet sich ebenso wie eine tabellarische Übersicht über die Ergebnisse der Umfrage im Anhang B. dieser Arbeit. Insgesamt werden 34 Studierende befragt. Hierbei handelt es sich um 18 Studierende des sechsten Fachsemesters (Datum der Umfrage: 12.07.2013) und 16 Studierende des siebten Fachsemesters (Datum der Umfrage: 02.07.2013), davon neun Studierende (26,47 %) mit Studienschwerpunkt Augenoptik und 25 Studierende (73,53 %) mit Studienschwerpunkt Augenoptik und Hörakustik.

Die Studierenden geben an, in welchem Bereich sie nach dem Studium gerne arbeiten möchten. Dies geschieht, um feststellen zu können, ob die Zukunftspläne der Studierenden, hier im Speziellen ein Interesse an einer späteren Beschäftigung in Industrie und Forschung, mit dem Wunsch, während des Studiums MATLAB zu lernen, korrelieren. Weiterhin soll ermittelt werden, ob der Studienschwerpunkt einen Einfluss auf den Standpunkt der Studierenden, MATLAB betreffend, hat. Außerdem wird das Interesse der Studierenden am Erlernen des Umgangs mit anderen Programmiertools/-sprachen erfragt. Abschließend machen die Studierenden eigene Vorschläge, wie MATLAB in das Studium der Augenoptik/Augenoptik und Hörakustik allgemein sowie direkt in die Lehrveranstaltung Mathematik eingebunden werden könnte.

Um die Aussagen der Studierenden auf mögliche Korrelationen hin überprüfen zu können, wird der ϕ-Koeffizient (Vierfelder-Korrelationskoeffizient) als Maß für die Stärke des Zusammenhangs zweier dichotomer Merkmale ermittelt. Dieser wird mit der Prüfgröße χ^2 auf Signifikanz hin überprüft. Dabei ist anzumerken, dass zur Ermittlung des ϕ-Koeffizienten grundsätzlich die Häufigkeit in den einzelnen Feldern der Vierfeldertafel ≥ 5 sein sollte, um zuverlässige Werte zu erhalten. Dem kann in der hier vorliegenden Auswertung aufgrund der geringen Fallzahl der befragten Personen jedoch nicht Rechnung getragen werden, weshalb die Ergebnisse lediglich als Anhaltspunkt dienen und nicht als in vollem Umfang verlässlich angesehen werden können.

Zu den Angaben, die Zukunftspläne der befragten Studierenden betreffend, ist anzumerken, dass hier Mehrfachnennungen erlaubt sind. Von den Befragten möchten nach dem Studium neun Personen (20,93 %) in einem augenoptischen und acht Personen (18,60 %) und ei-

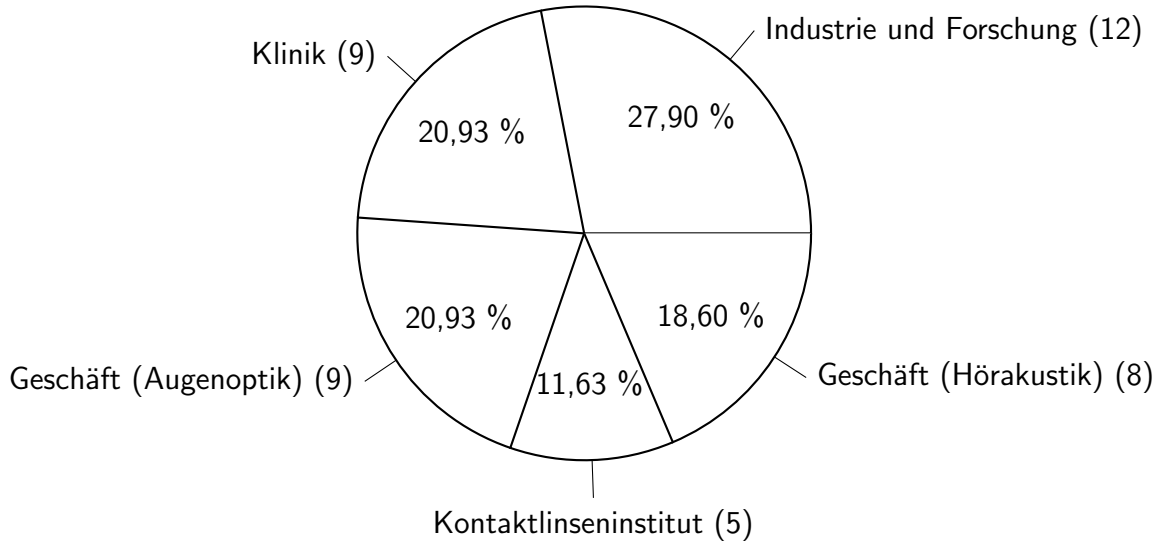


Abbildung 2.1.: Umfrage, Ergebnis zu Frage 2: Wo möchtest Du nach dem Studium am liebsten arbeiten?

nem hörakustischen Fachgeschäft arbeiten. Weitere neun Personen (20,93 %) möchten in einer Klinik arbeiten, fünf Personen (11,63 %)würden gerne eine Beschäftigung in einem Kontaktlinseninstitut aufnehmen. An einer Beschäftigung in Industrie und Forschung sind elf Personen interessiert, eine weitere Person gibt als Ziel „Forschung und Lehre“ an. Für die im Folgenden durchgeführte Auswertung werden die beiden letztgenannten Personengruppen gemeinsam als an der Industrie und Forschung interessierte Befragte mit 12 Personen (27,90 %) betrachtet (siehe Abb. 2.1).

Zunächst werden die Studierenden gefragt, ob sie es für sinnvoll halten, während des Studiums überhaupt oder in größerem Umfang als bisher mit MATLAB zu arbeiten (Frage 3). 18 Personen (52,94 %) geben an, dies für sinnvoll zu halten, vier Personen (11,76 %) finden das nicht sinnvoll. Die übrigen 12 Personen (35,29 %) antworten mit „weiß nicht“ (siehe Abb. 2.2).

Nun werden die Studierenden dahingehend befragt, ob sie den Praxisbezug des Studiums für ausreichend halten (Frage 4). Hier antworten sechs Personen (17,65 %) mit „ja“, 21 Personen (61,76 %) mit „nein“ und sieben Personen (20,59 %) mit „weiß nicht“ (siehe Abb. 2.3).

Anschließend werden die Studierenden gefragt, ob sie die Möglichkeit, den Praxisbezug des

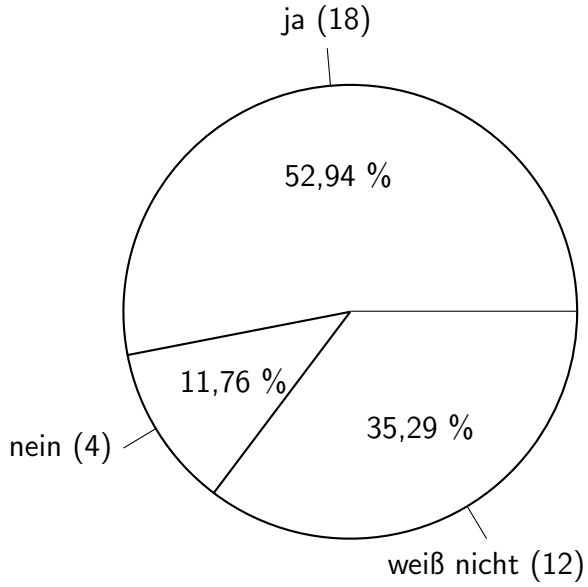


Abbildung 2.2.: Umfrage, Ergebnis zu Frage 3: Denkst Du, dass es sinnvoll wäre, während des Studiums (mehr) mit MATLAB zu arbeiten?

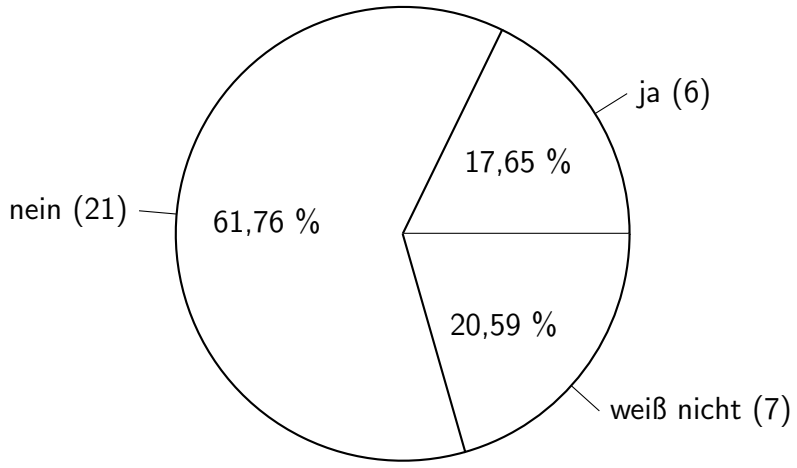


Abbildung 2.3.: Umfrage, Ergebnis zu Frage 4: Hältst Du den Praxisbezug des Studiums für ausreichend?

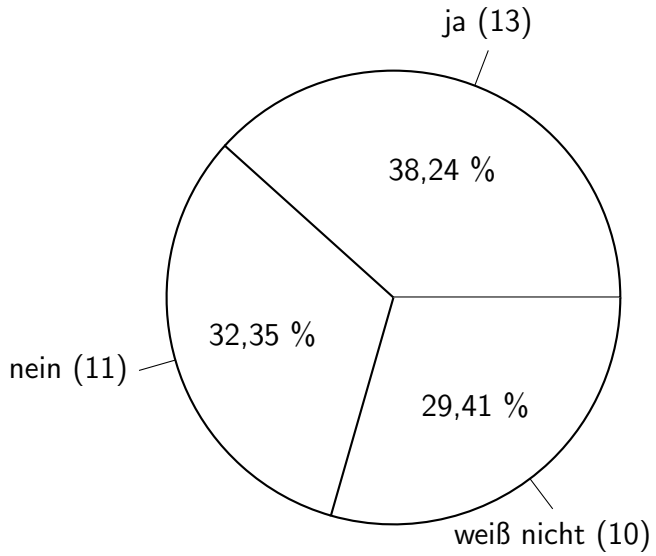


Abbildung 2.4.: Umfrage, Ergebnis zu Frage 5: Denkst Du, dass durch die Möglichkeit, während des Studiums MATLAB zu lernen, der Praxisbezug des Studiums erhöht werden könnte?

Studiums mithilfe von MATLAB zu erhöhen, sehen (Frage 5). Hierauf anworten 13 Personen (38,24 %) mit „ja“, elf Personen (32,35 %) mit „nein“ und zehn Personen (29,41 %) mit „weiß nicht“ (siehe Abb. 2.4).

Weiterhin werden die Studierenden gefragt, ob es in ihren Augen sinnvoll wäre, wenn bereits im ersten Semester, begleitend zur Vorlesung Mathematik, eine Einführung in MATLAB stattfände (Frage 6). 18 Personen (52,94 %) halten dies für sinnvoll, zehn Personen (29,41 %) sehen darin keinen Sinn. Sechs Personen (17,65 %) antworten mit „weiß nicht“ (siehe Abb. 2.5).

Ebenso werden die Studierenden gefragt, ob sie ein zusätzlich angebotenes Wahlfach, in dem MATLAB thematisiert wird, wählen würden (Frage 7). 13 Personen (38,24 %) würden dies tun, 15 Personen (44,12 %) würden ein solches Wahlfach nicht belegen, sechs Personen (17,65 %) sind unschlüssig (siehe Abb. 2.6).

Nun wird überprüft, ob die auf die Fragen 3 bis 7 gegebenen Antworten abhängig vom Studienschwerpunkt der Studierenden sind. Hierzu werden lediglich die Antwortalternativen „ja“ und „nein“ ausgewertet, die Antwortalternative „weiß nicht“ bleibt außen vor. Bei keiner dieser Fragen kann eine signifikante Korrelation zwischen Studienschwerpunkt und Antwort-

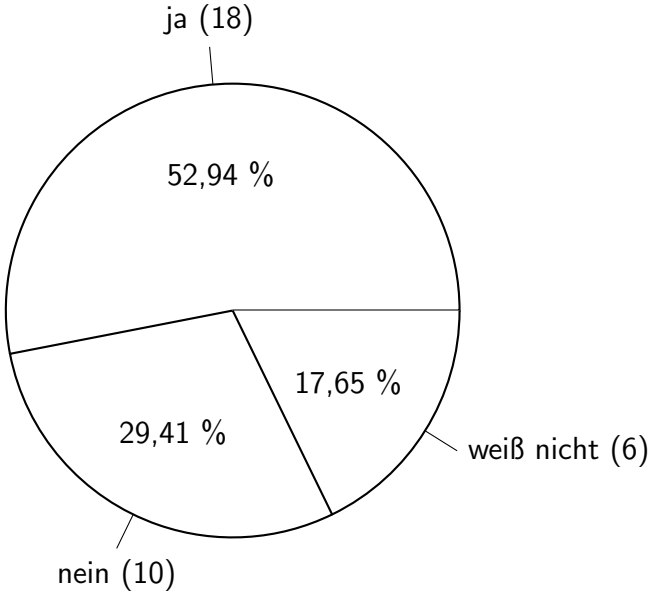


Abbildung 2.5.: Umfrage, Ergebnis zu Frage 6: Fändest Du es sinnvoll, wenn bereits im ersten Semester, begleitend zur Vorlesung Mathematik, eine Einführung in MATLAB stattfinden würde?

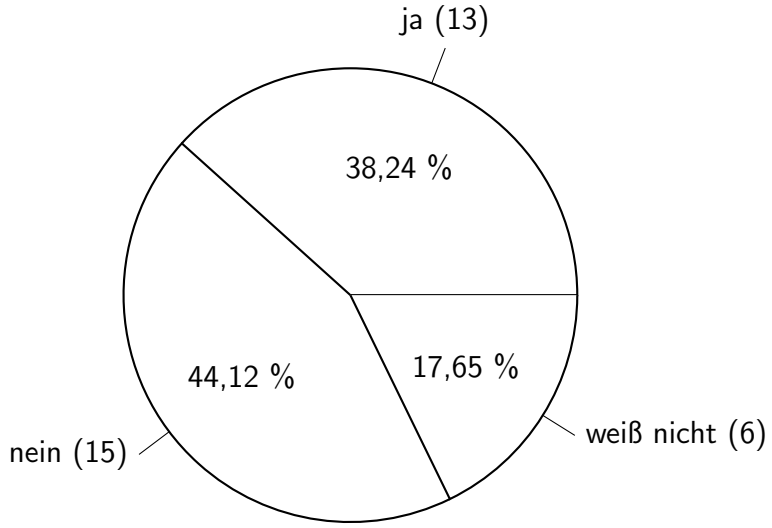


Abbildung 2.6.: Umfrage, Ergebnis zu Frage 7: Würdest Du ein zusätzlich angebotenes Wahlfach, in dem MATLAB thematisiert wird, wählen?

verhalten nachgewiesen werden. Ebenso wird überprüft, ob Studierende mit Interesse an einer zukünftigen Beschäftigung in Industrie und Forschung ein grundsätzlich höheres Interesse an MATLAB haben als Studierende mit anderen Zukunftsplänen. Auch hier kann, Bezug nehmend auf die auf die Fragen 3 bis 7 gegebenen Antworten und die Angaben derjenigen, die mit „weiß nicht“ antworten, nicht miteinbeziehend, keine signifikante Korrelation festgestellt werden. Auch die Antworten aller teilnehmenden Studierenden auf die Fragen 4 und 5 untereinander korrelieren nicht signifikant. Auch werden die Studierenden gefragt, welche

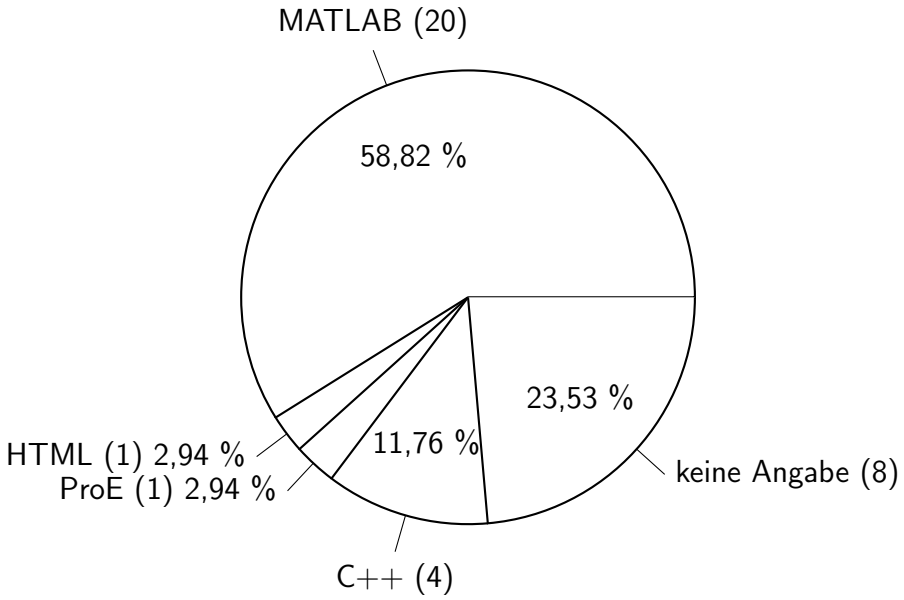


Abbildung 2.7.: Umfrage, Ergebnis zu Frage 8: Wenn Du die Wahl zwischen MATLAB und anderen Programmiertools/-sprachen hättest, was würdest Du wählen?

Programmiertools/-sprachen sie wählen würden (Frage 8). 20 Personen (58,82 %) würden MATLAB wählen, eine Person (2,94 %) möchte HTML, eine weitere Person (2,94 %) ProE lernen. Vier Personen (11,76 %) geben an, Interesse an C++ zu haben, die übrigen acht Personen (23,53 %) machen keine Angabe (siehe Abb. 2.7).

Nun wird die Frage gestellt, wie dringend die Studierenden den Bedarf an der Thematisierung von MATLAB im Studium sehen. Hierbei geben sie ihre Antwort auf einer Skala von 1 (überhaupt nicht) bis 5 (sehr dringend) an. Der Median der Angaben ist $\bar{x}_{Med} = 3$ und unterscheidet sich nicht zwischen zwei Gruppen mit Studienschwerpunkt Augenoptik bzw. Augenoptik und Hörakustik. Der Median derjenigen Studierenden, die an einer zukünftigen Beschäftigung in Industrie und Forschung interessiert sind, beträgt $\bar{x}_{Med} = 3{,}5$ und ist so-

mit gegenüber dem Median derjenigen Personengruppe, die andere Ziele hat ($\bar{x}_{Med} = 3$) geringfügig erhöht.

Abschließend werden die Studierenden gebeten, Vorschläge anzubringen, wie MATLAB in das Studium der Augenoptik/Augenoptik und Hörakustik im Allgemeinen und in die Lehrveranstaltung Mathematik im Speziellen eingebunden werden sollte (Fragen 10 und 11). Die hierauf von den Studierenden angeführten Vorschläge befinden sich in vollem Umfang im Anhang B. dieser Arbeit.
Aus den Vorschlägen der Studierenden dahingehend, wie MATLAB in die Lehrveranstaltung Mathematik und in das Studium eingebunden werden könnte, ist zum Einen der Schluss zu ziehen, dass die Mehrzahl sich dafür ausspricht, MATLAB im Rahmen einer Pflichtveranstaltung vorzustellen und die Grundlagen im Umgang mit diesem Programmiertool zu vermitteln. Zum anderen sind die Studierenden mehrheitlich der Auffassung, dass ein differenzierter Umgang mit MATLAB, der über die Grundlagen hinausgeht, eher in einer Wahl- als in einer Pflichtveranstaltung gelehrt werden sollte.

Die Ergebnisse der Umfrage zusammenfassend, ist festzustellen, dass ein grundsätzliches Interesse der Studierenden am Programmiertool MATLAB nicht von der Hand zu weisen ist. Zwar ist kein gesteigertes Interesse sowohl Studierender mit Studienschwerpunkt Augenoptik und Hörakustik als auch Studierender, die später in Industrie und Forschung arbeiten möchten, zu erkennen, jedoch lässt die Gesamtverteilung der Antworten darauf schließen, dass ein großer Teil der Studierenden es für sinnvoll erachtet, MATLAB während des Studiums zu erlernen. So erachten 52,94 % der Studierenden es für sinnvoll, sich während des Studiums überhaupt oder mehr mit MATLAB zu befassen (Frage 3), ebensoviele der befragten Personen würden eine Einführung in MATLAB bereits im ersten Semester begrüßen (Frage 6). Weiterhin sind 38,24 % der Befragten der Auffassung, dass sich der Praxisbezug des Studiums dadurch steigern ließe (Frage 5), welchen aktuell 61,67 % der Befragten als nicht ausreichend ansehen. Ein zukünftig angebotenes Wahlfach, in dem MATLAB thematisiert wird, würden 38,24 % der Studierenden wählen (Frage 7), vor die Wahl zwischen MATLAB und anderen Programmiertools/-sprachen gestellt, würden sich 58,82 % für MATLAB entscheiden (Frage 8). Mit einem Median von $\bar{x}_{Med} = 3$ sprechen sich die Studierenden für einen Bedarf daran aus, dass MATLAB zukünftig gelehrt wird. Auch die von den Studierenden selbst angebrachten Vorschläge, wie die Lehre von MATLAB vonstatten gehen könnte, lassen auf ein Interesse schließen.

Die mittels der Umfrage erhobenen Daten betrachtend, bleibt abschließend zu sagen, dass

dem vorhandenen Interesse der Studierenden an MATLAB Rechnung getragen werden sollte, auch um diese auf ein mögliche zukünftige Tätigkeit in Industrie und Forschung besser vorbereiten zu können.

3. Konzeption der e-Learning-Plattform

Wie eingangs erwähnt, beruht die e-Learning-Plattform, die den Studierenden zukünftig zur Verfügung gestellt werden soll, auf drei Säulen, deren erste das Selbststudium ist. Die Studierenden lernen, sich Themenkomplexe selbstständig zu erarbeiten und ihren eigenen Erfolg zu kontrollieren. Dabei sind sie terminlich nicht gebunden, wodurch sie unabhängig und autonom lernen können. Als Kommunikationsnetz, die beispielsweise die Verfügbarkeit der notwendigen Unterlagen gewährleistet, kann zunächst auf moodle zurückgegriffen werden. Hier stehen bereits Übungsaufgaben zur Lehrveranstaltung Mathematik zur Verfügung, worauf die in dieser Arbeit konzipierte e-Learning-Plattform weiter aufbaut.

Die zweite Säule, die die Plattform trägt, beschäftigt sich mit der Vermittlung von MATLAB und der Integration dieser Thematik in die Lehrveranstaltung Mathematik. Die Studierenden lernen hier, unter Anleitung und Betreuung, den Umgang mit einem Tool zur Lösung mathematischer Probleme. Auch hier werden Skripte zur Verfügung gestellt, welche es den Studierenden ermöglichen, sich auch außerhalb der betreut stattfindenden Praktika mit MATLAB zu befassen und die Thematik so selbstständig weiter zu festigen und auch zu vertiefen. Auch die Anwendung des Gelernten auf augenoptische und hörakustische Sachverhalte findet hier statt. So stehen den Studierenden Aufgaben zur Verfügung, die sie anleiten, mathematische Probleme im Hinblick auf optische und akustische Studieninhalte zu lösen.

Die dritte und letzte Säule befasst sich mit den Auswirkungen, die die Fähigkeit zum Umgang mit MATLAB auf das gesamte Studium haben kann. So werden den Studierenden, hinsichtlich verschiedener Bereiche, Programme mit grafischen Benutzeroberflächen zur Verfügung gestellt, die ein tiefgehendes Verständnis der so thematisierten Inhalte aus anderen Lehrveranstaltungen als derjenigen der Mathematik erzeugen sollen. MATLAB wird hier als Werkzeug genutzt, das den Studierenden die Zusammenhänge vieler Studieninhalte mit der Mathematik vermittelt. So wird der Stellenwert den Mathematik im Studium der Augenoptik/Augenoptik und Hörakustik verdeutlicht und die Motivation der Studierenden im Hinblick darauf

gesteigert. Ebenso wird hier die Relevanz des Erlernens von MATLAB im Studium der Augenoptik/Augenoptik und Hörakustik deutlich: Durch MATLAB lernen die Studierenden, fachspezifische Inhalte auf mathematische Grundlagen zurückzuführen, sie so zu verstehen und Probleme lösen zu können.

3.1. Selbststudium

Die e-Learning-Plattform stellt den Studierenden Skripte zu zwei Themenschwerpunkten, die im Rahmen der Lehrveranstaltung Mathematik selbstständig zu erarbeiten sind, online zur Verfügung. Hierbei handelt es sich um die Themen „Lineare Gleichungssysteme und Matrizen“ sowie „Mehrdimensionale Integration“. Wie bereits in Kapitel 2.1. E-Learning im Kontext der Lerntheorie erläutert, kann das Verstehen einer Thematik nur erfolgen, wenn diese sowohl am Beispiel, als auch in allgemeinen Zusammenhängen erklärt wird. Die zur Verfügung gestellten Skripte berücksichtigen dies selbstverständlich. Aufgaben zur Kontrolle des Lernerfolgs sind am Ende der Skripte vorhanden, auch die Lösungen der Aufgaben werden den Studierenden zur Verfügung gestellt. Die Verfügbarkeit von Aufgaben und Lösungen sowie die Selbstständigkeit im Umgang damit fördern die Eigenverantwortung der Studierenden.

Nachfolgend werden Aufbau und Inhalt der Skripte beschrieben und begründet. Die Skripte selbst befinden sich der Übersicht wegen im Anhang dieser Arbeit (siehe Anhang C. Skript zum Themenschwerpunkt Lineare Gleichungssysteme und Matrizen sowie Anhang D. Skript zum Themenschwerpunkt Mehrdimensionale Integration) und sollen den Studierenden zukünftig online über moodle zur Verfügung gestellt werden.

3.1.1. Themenschwerpunkt Lineare Gleichungssysteme und Matrizen

Das Skript zum eingenständigen Erarbeiten des Themenschwerpunkts Lineare Gleichungssysteme und Matrizen gibt eine Übersicht über die Lösung linearer Gleichungssysteme mit verschiedenen Verfahren und erklärt den Zusammenhang zwischen linearen Gleichungssystemen und Matrizen. Auch auf verschiedene Rechenoperationen für die Rechnung mit Matrizen wird eingegangen. Eine Anwendung auf augenoptisch oder hörakustisch relevante Inhalte findet an dieser Stelle nicht statt, da diese von den Studierenden nicht selbstständig erarbeitet werden soll, sondern während der Vorlesung der Lehrveranstaltung Mathematik in Form der Matrizenoptik thematisiert wird. Auch die Lösung linearer Gleichungssysteme und Matrizen

mit MATLAB wird gezeigt, um eine enge Verbindung der einzelnen Elemente der konzipierten e-Learning-Plattform zu erreichen. Hierbei wird deutlich, dass MATLAB, wie der Name schon sagt, gerade bezüglich der Rechnung mit Matrizen für die Studierenden eine wesentliche Vereinfachung bei der Lösung mathematischer Probleme darstellt. Schließlich stehen Aufgaben zur besprochenen Thematik, ebenso wie Lösungen, auch mit MATLAB, zur Verfügung, die den Studierenden die Vertiefung der Thematik und eine Erfolgskontrolle ermöglichen.

3.1.2. Themenschwerpunkt Mehrdimensionale Integration

Das Erarbeiten des Themenschwerpunkts Mehrdimensionale Integration mit dem zur Verfügung gestellten Skript setzt Kenntnisse der Integralrechnung zur Flächenberechnung in zweidimensionalen kartesischen Koordinaten voraus. Darauf aufbauend, wird nun die Integration über mehrdimensionale Bereiche erklärt, sowohl in kartesischen Koordinaten, als auch in Polar- und Kugelkoordinaten. Als Anwendung auf augenoptisch relevante Studieninhalte wird anschließend auf die Berechnung von Raumwinkeln eingegangen, welche besondere Bedeutung für die Lichttechnik hat. In der zugehörigen Lehrveranstaltung wird dieses Thema ebenfalls besprochen, den Studierenden soll durch das hier zur Verfügung gestellt Skript allerdings die Möglichkeit gegeben werden, sich bereits im Zuge der Lehrveranstaltung Mathematik damit auseinanderzusetzen. Eine wichtige Voraussetzung zum Verständnis von Inhalten der Lichttechnik kann so bereits an dieser Stelle geschaffen werden. Auch wird den Studierenden dadurch der Zusammenhang von Mathematik und anderen Studieninhalten verdeutlicht. Anschließend wird die Lösung mehrdimensionaler Integrale mit MATLAB gezeigt, um auch hier eine enge Verbindung der einzelnen Elemente der konzipierten e-Learning-Plattform zu erreichen. Schließlich stehen Aufgaben zur besprochenen Thematik, ebenso wie Lösungen, auch mit MATLAB, zur Verfügung, die den Studierenden die Vertiefung der Thematik und eine Erfolgskontrolle ermöglichen.

3.2. Integration von MATLAB in die Lehrveranstaltung Mathematik

Wie bereits in Kapitel 2.1. E-Learning im Kontext der Lerntheorie erwähnt, kann die Nutzung elektronischer Medien und die damit verbundene Möglichkeit der besseren Darstellung und Strukturierung der thematisierten Inhalte zu einem besseren Verständnis dieser führen. Ein solches soll im Zuge dieser Arbeit durch das Erlernen des Umgangs mit MATLAB erreicht werden. Deshalb stellt die hier konzipierte e-Learning-Plattform den Studierenden weiterhin

Skripte zum genutzen Programm MATLAB zur Verfügung. Hierbei werden die zum Umgang mit MATLAB essentiellen Grundlagen, gegliedert in zwei Teile, thematisiert. Aufgaben zur Kontrolle des Lernerfolgs sind am Ende der Skripte vorhanden, auch die Lösungen der Aufgaben werden den Studierenden zur Verfügung gestellt. Die Verfügbarkeit von Aufgaben und Lösungen sowie die Selbstständigkeit im Umgang damit sollen auch hier die Eigenverantwortung der Studierenden fördern.

Das Erlernen der hier aufgeführten Grundlagen wird jedoch nicht den Studierenden allein überlassen. Vielmehr soll, begleitend zur Lehrveranstaltung Mathematik, ein Praktikum stattfinden, in dem die Inhalte vermittelt werden. Ebenso soll hier das Programm MATLAB vorgestellt werden. Die Einführung in MATLAB während des Praktikums soll sich nach den zur Verfügung gestellten Skripten richten, welche den Studierenden wiederum als „Nachschlagewerk“ dienen können. Die an die Skripte angegliederten Aufgaben sollen während der Zeit des Praktikums von den Studierenden eigenständig, aber insofern betreut gelöst werden, als dass beim Auftreten von Schwierigkeiten während der Bearbeitung der Aufgaben ein Betreuer vor Ort ist, der Hilfe anbietet.

Das Praktikum soll aufgeteilt auf zwei Termine zu je drei Stunden stattfinden. Dies erklärt die Aufspaltung der begleitenden Skripte in zwei Teile. In den Kapiteln 3.2.1. Praktikum MATLAB - Teil 1: Direkte Eingabe und 3.2.2. Praktikum MATLAB - Teil 2: Indirekte Eingabe werden Aufbau und Inhalt der Skripte beschrieben und begründet. Ebenfalls findet sich dort ein detaillierterer Vorschlag zur zeitlichen Gestaltung der zugehörigen Praktikumstermine. Eine Zeitdauer von isgesamt sechs Stunden wird als realistisch und umsetzbar angesehen. Auch im Hinblick auf den Workload der Studierenden ist ein Mehraufwand von sechs Stunden vertretbar.

Die Skripte selbst befinden sich der Übersicht wegen im Anhang dieser Arbeit (siehe Anhang E. Skript zum Praktikum MATLAB - Teil 1: Direkte Eingabe sowie Anhang F. Skript zum Praktikum MATLAB - Teil 2: Indirekte Eingabe) und sollen den Studierenden zukünftig online über moodle zur Verfügung gestellt werden.

3.2.1. Praktikum MATLAB - Teil 1: Direkte Eingabe

Der erste Teil des angebotenen MATLAB-Praktikums soll sich mit der direkten Eingabe, also der Nutzung von MATLAB als interaktiver Berechnungsumgebung, beschäftigen. Das begleitende Skript ist wie folgt aufgebaut: Zunächst wird MATLAB vorgestellt, die wesentlichen Bedienelemente werden im Zuge dessen erklärt. Anschließend wird auf die MATLAB-Syntax,

Variablendefinition und verschiedene Funktionen eingegangen. Es folgen Übersichten über wichtige Befehle. Diese dienen den Studierenden als Hilfestellung zur Lösung der an das Skript angegliederten Aufgaben. Die Auswahl dieser Aufgaben deckt die in der Lehrveranstaltung Mathemtik gelehrten Inhalte im Wesentlichen ab. Dadurch erhalten die Studierneden einen guten Einblick in die vielfältigen Nutzungsmöglichkeiten von MATLAB.

Wie bereits erwähnt, soll ein Praktikum mit einer Dauer von drei Stunden stattfinden. Diese sind wie folgt aufzuteilen: 90 Minuten sollen darauf verwendet werden, den Studierenden MATLAB zunächst vorzustellen, die einzelnen Fenster und möglichen Funktionen zu erklären sowie eine Einweisung in die Syntax durchzuführen. Hierbei können den Studierenden bereits kleine Beispiele direkt in MATLAB gezeigt werden. Dieser Abschnitt kann in Form einer Vorlesung erfolgen. Der Ablauf und die Inhalte dieser theoretischen Einführung in MATLAB sollen sich an das begleitende Skript halten.

Weitere 90 Minuten sollen den Studierenden nun zur Verfügung gestellt werden, um sich selbst mit MATLAB auseinanderzusetzen. Hier können die Studierenden verschiedene Funktionen testen oder sich mit der Lösung der im Skript enthaltenen Aufgaben beschäftigen. Dabei muss ein Betreuer anwesend sein, der den Studierenden bei eventuell aufkommenden Schwierigkeiten oder Fragen behilflich ist.

Es wird empfohlen, das Praktikum eher gegen Ende des Semesters anzubieten, da die Studierenden zu diesem Zeitpunkt bereits mit dem Großteil der Inhalte der Lehrveranstaltung Mathematik vertraut sind und so möglichst viele verschiedene Anwendungsmöglichkeiten haben. Auch macht die Lösung der zur Verfügung gestellten Aufgaben mit MATLAB erst dann Sinn, wenn die Studierenden die grundsätzlichen Inhalte bereits kennen gelernt haben.

3.2.2. Praktikum MATLAB - Teil 2: Indirekte Eingabe

Der zweite Teil des angebotenen MATLAB-Praktikums soll sich mit der indirekten Eingabe, also der Nutzung des MATLAB-Editors zur Erstellung eigener Programme, beschäftigen. Das begleitende Skript ist wie folgt aufgebaut: Zunächst wird der MATLAB-Editor vorgestellt, anschließend werden wichtige Befehle zum Erstellen von Grafiken erklärt. Diese bauen auf Teil 1 des Praktikums auf, werden allerdings deshalb erst in Teil 2 des Praktikums eingeführt, weil sie sich erst bei der Eingabe über den MATLAB-Editor effizient nutzen lassen. Auch hier folgen Übersichten über wichtige Befehle, die den Studierenden als Hilfestellung zur Lösung der an das Skript angegliederten Aufgaben dienen. Nun werden die Grundlagen der Programmierung erklärt, hierbei wird auf verschiedene Funktionen und Sprachkonstrukte eingegangen.

Wieder schließen sich Aufgaben an. Hier sollen kleinere Programme zu in der Lehrveranstaltung Mathematik thematisierten Inhalten von den Studierenden selbst geschrieben werden. Zu den Aufgaben sind verschiedene Tipps vorhanden, die die Studierenden als Hilfestellung nutzen können. Auch Lösungsmöglichkeiten werden den Studierenden zur Verfügung gestellt.

Wie bereits erwähnt, soll auch dieses Praktikum mit einer Dauer von drei Stunden stattfinden. Diese sind wie folgt aufzuteilen: 45 Minuten sollen darauf verwendet werden, den Studierenden den MATLAB-Editor zunächst vorzustellenund die Grundstrukturen der Programmierung zu erklären. Hierbei können den Studierenden bereits kleine Beispiele direkt in MATLAB gezeigt werden. Dieser Abschnitt kann in Form einer Vorlesung erfolgen. Der Ablauf und die Inhalte dieser theoretischen Einführung in MATLAB sollen sich an das begleitende Skript halten.

Weitere 135 Minuten sollen den Studierenden nun zur Verfügung gestellt werden, um sich selbst mit MATLAB auseinanderzusetzen. Hier sollen die Studierenden die im Skript enthaltenen Aufgaben bearbeiten. Dabei muss ebenfalls ein Betreuer anwesend sein, der den Studierenden bei eventuell aufkommenden Schwierigkeiten oder Fragen behilflich ist.

Auch hier wird selbstverständlich empfohlen, das Praktikum eher gegen Ende des Semesters anzubieten, da die Studierenden zu diesem Zeitpunkt bereits mit dem Großteil der Inhalte der Lehrveranstaltung Mathematik vertraut sind. Zudem ist es unerlässlich, dass die Studierenden das Praktikum zur direkten Eingabe in MATLAB (Teil 1) bereits absolviert haben.

3.3. MATLAB im Gesamtkontext des Studiums: Augenoptisch und hörakustisch relevante Anwendungen

Neben den Möglichkeiten, MATLAB zur direkten oder indirekten Eingabe zu nutzen, besteht zusätzlich die Möglichkeit, grafische Oberflächen zu programmieren. Die Programmierung dieser erfordert allerdings, ebenso wie die Einarbeitung in dieses Thema, deutlich mehr Zeit als die in das vorgesehene Praktikum integrierten Möglichkeiten der Nutzung von MATLAB und ist deshalb, bereits aus zeitlichen Gründen, nicht ohne weiteres in die Lehrveranstaltung Mathematik einzubinden. Allerdings kann die Erstellung von Programmen mit grafischer Benutzeroberfläche zur hilfreichen Anwendung von MATLAB in Augenoptik und Hörakustik dienen, da über solche Programme fachlich relevante Inhalte dargestellt werden und den Stu-

dierenden, die diese Programme dann nutzen, beim Verständnis der jeweils aufgegriffenen Thematik helfen können. Eine Aufnahme solcher Programme in das Konzept der e-Learning-Plattform für die Lehrveranstaltung Mathematik ist deshalb, wie auch aus dem Grund, dass nur so deutlich werden kann, welche Rolle die Mathematik und der Umgang mit MATLAB im Studium der Augenoptik/Augenoptik und Hörakustik spielen können, unerlässlich.

Es stellt sich nun also die Frage, wie diesem Problem begegnet werden kann. Hier bestehen grundsätzlich zwei verschiedene Lösungsansätze. Zum Einen ist es möglich, die Erstellung von MATLAB-Programmen mit grafischen Benutzeroberflächen während des vorgesehenen Praktikums zur freiwilligen Bearbeitung, auch außerhalb der Termine des Praktikums, vorzustellen. Besonders an MATLAB interessierte Studierende könnten sich dann freiwillig in die Programmierung grafischer Oberflächen einarbeiten und fachspezifische Programme schreiben.
Zum Anderen besteht die Möglichkeit, die Integration von MATLAB in das Studium der Augenoptik/Augenoptik und Hörakustik über die Grenzen der Lehrveranstaltung Mathematik hinaus auszuweiten. So könnte beispielsweise ein Wahlfach geschaffen werden, in welchem insbesondere die Programmierung grafischer Oberflächen mit MATLAB gelehrt wird. Hier könnte den Studierenden die Aufgabe gestellt werden, zur einem selbst gewählten Thema ein Programm zu entwickeln. Beispielhaft seien hier nur einige Bereiche genannt, zu denen sich MATLAB-Programme erstellen ließen:

- Die Berechnung einkurviger Kontaktlinsen zur Vorbereitung auf Inhalte, die in der Lehrveranstaltung Kontaktlinsen-Technik gelehrt werden.
- Die Visualisierung von Strahlendurchgängen durch verschiedene optische Systeme aus dem Bereich der Geometischen Optik.
- Im Bereich der Wellenlehre Programme zur Beugung an anderen als der in Kapitel 3.3.2. Die Beugung an der Kreisblende als Beispiel der augenoptisch relevanten Anwendung von MATLAB Form der Blendenöffnung.
- Im Bereich der Psychophysik die Berechnung und grafische Darstellung der psychometrischen Messfunktion nach Eingabe zuvor gemessener Werte.
- Im Bereich der Digitalen Signalverarbeitung bestehen zahlreiche Möglichkeiten der Anwendung von MATLAB, zum Beispiel die Darstellung verschiedener Signale, Filter und vieles mehr.

Bereits in Kapitel 2.4. Das Interesse der Studierenden an MATLAB: Ergebnisse einer Umfrage wird deutlich, dass viele Studierende ein solches Angebot begrüßen und gerne wahrnehmen würden. Im Rahmen eines solchen Wahlfachs könnte dann eine Vielzahl an verschiedenen Programmen mit grafischen Benutzeroberflächen entstehen, die die in dieser Arbeit konzipierte e-Learning-Plattform schrittweise erweitern würden, um diese für die Studierenden noch hilfreicher werden zu lassen.

Exemplarisch werden den Studierenden an dieser Stelle zwei Programme mit grafischer Benutzeroberfläche zur Verfügung gestellt, jeweils eines für einen hörakustisch und eines für einen augenoptisch relevanten Sachverhalt. Auf diese wird in den Kapiteln 3.3.1. FIR-Filter als Beispiel der hörakustisch relevanten Anwendung von MATLAB und 3.3.2. Die Beugung an der Kreisblende als Beispiel der augenoptisch relevanten Anwendung von MATLAB näher eingegangen. Zu jedem Programm besteht ein kurz gehaltenes Skript, das den Studierenden die grundsätzlichen Zusammenhänge zum Verständnis der jeweiligen Thematik, die im entsprechenden Programm aufgegriffen wird, näherbringt. Auch diese Programme und Skripte sollen den Studierenden zukünftig online über moodle zur Verfügung gestellt werden.

3.3.1. FIR-Filter als Beispiel der hörakustisch relevanten Anwendung von MATLAB

Das als Beispiel der hörakustisch relevanten Anwendung erstellte Programm, welches mathematische Hintergründe bei der Berechnung eines FIR-Bandpass-Filters grafisch darstellt, wird in Abb. 3.1 gezeigt. Das Programm bietet die Möglichkeit, die untere und obere Grenzfrequenz eines Bandpasses mithilfe der Schieberegler zu verändern. Dabei sind für die untere Grenzfrequenz Werte zwischen 0 und 5000 Hz und für die obere Grenzfrequenz Werte zwischen 5001 und 10000 Hz verfügbar. Beim Start des Programms ist die untere Grenzfrequenz auf 4000 Hz und die obere Grenzfrequenz auf 6000 Hz eingestellt. Die Lage der Nullstellen sind im Programm ebenso sichtbar wie die einzelnen Filterkoeffizienten, welche von MATLAB selbst berechnet werden und der Amplitudengang des jeweiligen Bandpassfilters in linearer und logarithmischer Darstellung.

Dieses Programm beschäftigt sich also mit Inhalten aus dem Bereich der digitalen Signalverarbeitung und kann somit in der entsprechenden Lehrveranstaltung eingesetzt werden, um den Studierenden das Verständnis der Zusammenhänge zu vereinfachen.

Das zugehörige Skript liefert eine kurze Übersicht über die Bedeutung verschiedener Filter für die digitale Signalverarbeitung und die Hörakustik sowie deren Funktionsweise und ermöglicht

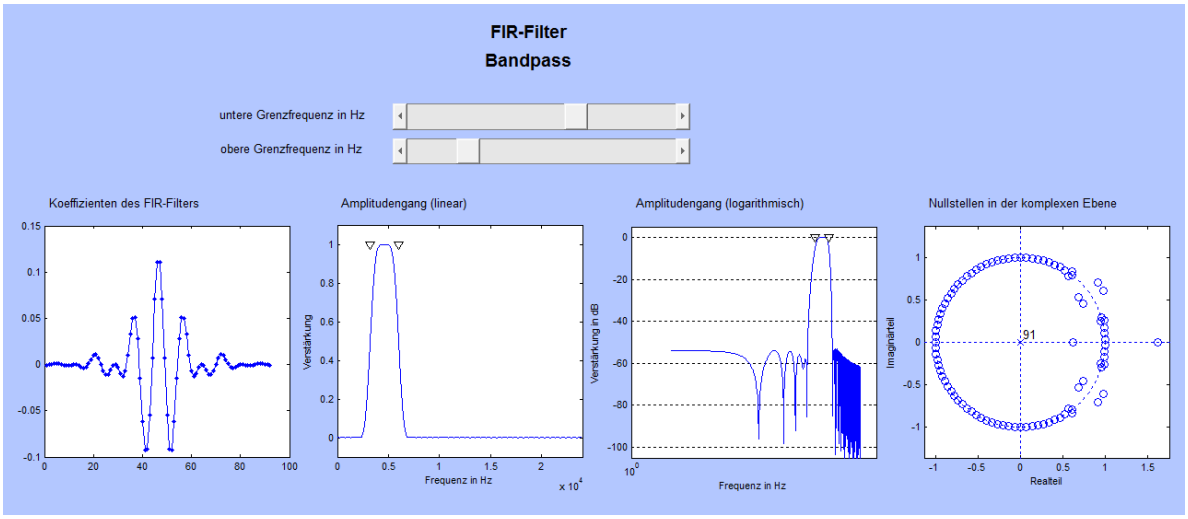


Abbildung 3.1.: Programm zum Thema FIR-Filter (Bandpass): Eine hörakustisch relevante Anwendung von MATLAB

den Studierenden somit, das vorgestellte Programm in die Gesamtthematik der Filter in der Hörakustik einzuordnen. Das Skript schließt mit zwei Fragen, die die Studierenden mithilfe des zur Verfügung gestellten Programms beantworten können, um das Erlernte zu festigen.

Der Übersicht wegen befindet sich das Skript zu dieser Anwendung in Anhang G. Skript zur MATLAB-Anwendung: FIR-Filter, der zugehörige Programmcode ist in Anhang H. Programmcode zur MATLAB-Anwendung: FIR-Filter dieser Arbeit zu finden.

3.3.2. Die Beugung an der Kreisblende als Beispiel der augenoptisch relevanten Anwendung von MATLAB

Das als Beispiel der augenoptisch relevanten Anwendung erstellte Programm, welches mathematische Hintergründe der Beugung an der Kreisblende grafisch darstellt, wird in Abb. 3.2 gezeigt. Dieses Programm verdeutlicht, wie sich das Beugungsmuster ganz allgemein in Abhängigkeit vom Radius der Kreisblende ändert. Dazu wird zunächst die Kreisblende an sich dargestellt. Anschließend wird das Beugungsmuster gezeigt. Hierbei kann neben der Beugungsstruktur an sich auch das Größenverhältnis der Struktur, beispielsweise der Maxima, die im Beugungsbild ersichtlich sind, mit der Größe der Blende verglichen werden. Schließlich wird das Beugungsmuster in logarithmierter Form dargestellt. Dies bietet den Vorteil, dass nun in den Regionen, in denen die Amplituden klein sind, Details sichtbar werden.

Dieses Programm beschäftigt sich also mit Inhalten aus dem Bereich der Wellenlehre und der

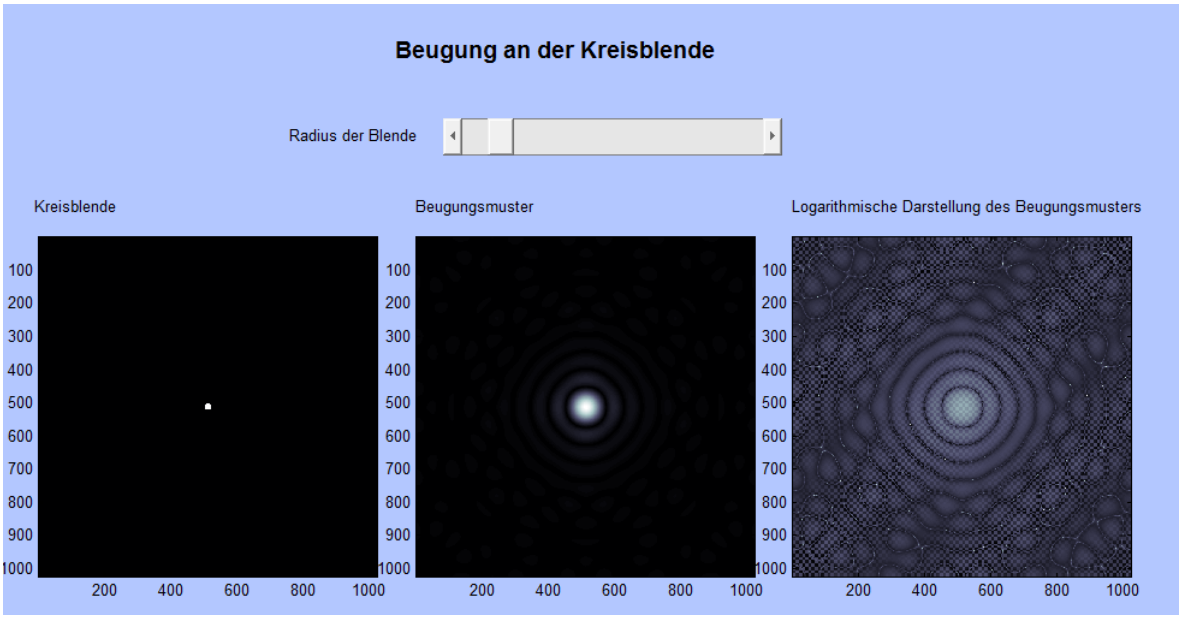


Abbildung 3.2.: MATLAB-Programm zum Thema Beugung an der Kreisblende: Eine augenoptisch relevante Anwendung von MATLAB

physikalischen Optik und kann somit in den entsprechenden Lehrveranstaltungen eingesetzt werden, um den Studierenden das Verständnis der Zusammenhänge zu vereinfachen.

Das zugehörige Skript erklärt, in verkürzter Form, den Zusammenhang zwischen Fourier-Transformation und Beugung, um den Studierenden die mathematische Grundlage zur Entstehung von Beugungsmustern deutlich zu machen. Außerdem wird kurz und knapp auf die Fast Fourier Transformation, welche in MATLAB genutzt werden kann, eingegangen, um deren Grundgedanken deutlich zu machen. Das Programm ermöglicht den Studierenden gemeinsam mit dem begleitenden Skript so, die Entstehung von Beugungsmustern zu begreifen und die in den Lehrveranstaltungen Wellenlehre und Physical Optics thematisierten Inhalte zu vertiefen. Auch dieses Skript schließt mit zwei Fragen, die die Studierenden mithilfe des zur Verfügung gestellten Programms beantworten können, um das Erlernte zu festigen.

Der Übersicht wegen befindet sich das Skript zu dieser Anwendung in Anhang I. Skript zur MATLAB-Anwendung: Beugung an der Kreisblende, der zugehörige Programmcode ist in Anhang J. Programmcode zur MATLAB-Anwendung: Beugung an der Kreisblende dieser Arbeit zu finden.

4. Fazit

Schließlich stellt sich die Frage, inwieweit das zu Beginn festgelegte Ziel, auf der Grundlage der bereits online zur Verfügung stehenden Übungsaufgaben zur Lehrveranstaltung Mathematik, eine e-Learning-Plattform zu dieser Lehrveranstaltung im Studiengang Augenoptik/Augenoptik und Hörakustik an der Hochschule Aalen aufzubauen, mit dieser Arbeit erreicht wird.

Zunächst besteht hier die Festlegung, dass die konzipierte Plattform die Studierenden dabei unterstützen soll, sich mathematische Inhalte eigenständig anzueignen. Hierzu werden in dieser Arbeit zwei Skripte zu den Themenschwerpunkten Lineare Gleichungssysteme und Matrizen sowie Mehrdimensionale Integration vorgeltellt (siehe Kapitel 3.1. Selbststudium). Diese Skripte stellen die Themen zunächst allgemein vor, zeigen anschließend an Beispielen die konkrete Umsetzung und schließen mit Aufgaben und Lösungen. Bereits an dieser Stelle werden, zusätzlich zu den analytischen Lösungen, Lösungen der Aufgaben mit MATLAB vorgestellt, um eine Verzahnung der Inhalte, die die e-Learning-Plattform enthält, zu erreichen. Die Studierenden können mithilfe dieser Skripte selbstständig und eigenverantwortlich lernen und ihren Lernerfolg kontrollieren.

Weiterhin sollen die Studierenden den Umgang mit MATLAB erlernen. Dies soll das Verständnis der Mathematik verteifen und zeigen, dass die Fähigkeit zur Bedienung eines solchen Berechnungstools die Aufgaben des Studiums erleichtern kann. Das hierzu konzipierte Praktikum (siehe Kapitel 3.2. Integration von MATLAB in die Lehrveranstaltung Mathematik) mit den zugehörigen Skripten ermöglicht den Studierenden einen Einstieg, zum Einen in MATLAB, zum Anderen in die Programmierung im Allgemeinen. Aufgrund der in der Umfrage MATLAB in Mathematik erhobenen Daten (siehe Kapitel 2.4. Das Interesse der Studierenden an MATLAB: Ergebnisse einer Umfrage) ist davon auszugehen, dass die Studierenden dies begrüßen. Weiterhin kann sich der Studiengang Augenoptik/Augenoptik und Hörakustik hierdurch ein Alleinstellungsmerkmal gegenüber anderen fachverwandten Studiengängen in Deutschland erarbeiten, was den zukünftigen Absolventen zugute kommen wird.

Auch soll den Studierenden gezeigt werden, wie sich MATLAB-Kenntnisse und die Fähigkeit

des Programmierens auf augenoptische und hörakustische Inhalte anderer Lehrveranstaltungen und somit auf große Teile des gesamten Studiums anwenden lassen. Hier soll besonders die Relevanz des Anwendung von MATLAB im Bereich der Augenoptik und Hörakustik deutlich werden. Dies wird zum Einen dadurch erreicht, dass den Studierenden zwei Programme mit grafischer Benutzeroberfläche zur Verfügung gestellt werden (siehe Kapitel 3.3.1. FIR-Filter als Beispiel der hörakustisch relevanten Anwendung von MATLAB und Kapitel 3.3.2. Die Beugung an der Kreisblende als Beispiel der augenoptisch relevanten Anwendung von MATLAB), die das Angebot der e-Learning-Plattform, welche zukünftig jedoch weiter ausgebaut werden kann, abrunden.

Insgesamt ist festzustellen, dass die konzipierte Plattform sowohl auf die Wünsche der Studierenden eingeht, als auch eine Erweiterung der momentanen Inhalte der Lehrveranstaltung Mathematik vornimmt. Auf Effizienz und Relevanz der hierzu genutzten Methoden legt die konzipierte e-Learning-Plattform mit moodle als zu verwendendem Kommunikationsnetz besonderen Wert. Auch auf die Umsetzbarkeit des Konzeptes, unter anderem im Hinblick auf zeitliche Aspekte, wird Rücksicht genommen.

Mit den in Kapitel 3.3. MATLAB im Gesamtkontext des Studiums: Augenoptisch und hörakustisch relevante Anwendungen vorgestellten Möglichkeiten zur zukünftigen Erweiterung bietet das im Rahmen dieser Arbeit erstellte Konzept einer e-Learning-Plattform eine durchaus Erfolg versprechende Grundlage zur Unterstützung der Studierenden hinsichtlich mathematischer und technischer Studieninhalte und zur Integration von MATLAB in die Lehrveranstaltung Mathematik im Studiengang Augenoptik und Hörakustik an der Hochschule Aalen.

5. Literatur

[1] **Konrad K., Traub S.:** Kooperatives Lernen. Theorie un Praxis in Schule, Hochschule und Erwachsenenbildung. 2. Aufl. Hohengehren: Schneider Verlag 2005, S. 24

[2] **Bauer R., Philippi T.:** Einstieg ins E-Learning. Nürnberg: BW Bildung und Wissen Verlag 2001, S. 108

[3] **Bauer R., Philippi T.:** Einstieg ins E-Learning. Nürnberg: BW Bildung und Wissen Verlag 2001, S. 97 f.

[4] **Bauer R., Philippi T.:** Einstieg ins E-Learning. Nürnberg: BW Bildung und Wissen Verlag 2001, S. 99 - 101

[5] **Bauer R., Philippi T.:** Einstieg ins E-Learning. Nürnberg: BW Bildung und Wissen Verlag 2001, S. 102

[6] **Bauer R., Philippi T.:** Einstieg ins E-Learning. Nürnberg: BW Bildung und Wissen Verlag 2001, S. 103 f.

[7] **Buth M.:** Lerntheorien. In: Texte zur pädagogischen Forschung und Lehre, Vol. 3. Bad Salzdetfurth: Verlag Franzbecker 1995, S. 119

[8] **Beuth Hochschule für Technik Berlin** Studiengang Augenoptik/Optometrie: Modulhandbuch. Berlin 2010

[9] **Ernst-Abbe-Fachhochschule Jena** Studiengang Augenoptik/Optometrie: ECTS-Informationsbroschüre. Jena 2013

[10] **Ostfalia Hochschule für angewandte Wissenschaften** Studiengang Augenoptik: Modulhandbuch. Wolfsburg 2010

[11] **Fachhochschule Lübeck** Studiengang Augenoptik/Optometrie: Modulhandbuch. Lübeck 2008

[12] **Fachhochschule Lübeck** Studiengang Hörakustik: Modulhandbuch und detaillierte Beschreibung der Lehrveranstaltungen. Lübeck 2013, S.25

[13] **Hochschule für angewandte Wissenschaften München** Studiengang Augenoptik/Optometrie: Studieninhalte. Online verfügbar unter http://www.fb06.fh-muenchen.de/fb/index.php/de/bachelorstudium/aob/studieninhalte.html, eingesehen am 22.09.2013

[14] **Hochschule Aalen Technik und Wirtschaft** Studiengang Optoelektronik/Lasertechnik: Modulhandbuch. Aalen 2011, S.4

[15] **Fachhochschule Koblenz** Studiengang Optik und Lasertechnik: Modulhandbuch. Koblenz 2012, S. 8

[16] **Hochschule Esslingen** Studiengang Mechatronik/Feinwerk- und Mikrotechnik: Modulhandbuch. Esslingen 2013, S.25

[17] **HAWK Hochschule für angewandte Wissenschaft und Kunst Fachhochschule Hildesheim/Holzminden/Göttingen** Studiengang Präzisionsmaschinenbau: Modulhandbuch. Göttingen 2012, S. 21

[18] **Hochschule für angewandte Wissenschaften München** Studiengang Mechatronik/Feinwerktechnik: Studieninhalte. Online verfügbar unter http://www.fb06.fh-muenchen.de/fb/index.php/de/bachelorstudium/mfb/ studieninhalte.html?ItemID=604&id=82&ids=939, eingesehen am 22.09.2013

[19] **Ernst-Abbe-Fachhochschule Jena** Studiengang Feinwerktechnik/Precision Engineering: ECTS-Informationsbroschüre. Jena 2010

[20] **Ernst-Abbe-Fachhochschule Jena** Studiengang Laser- und Optotechnologien: ECTS-Informationsbroschüre. Jena 2012, S. 30

[21] **Georg-Simon-Ohm Hochschule Nürnberg** Studiengang Mechatronik/Feinwerktechnik: Modulhandbuch. Nürnberg 2012, S. 7

[22] **Hoffmann E., Ulrich J.:** Hörakustik 2.0. Theorie und Praxis. 2. Aufl. Heidelberg: DOZ-Verlag 2011, S. 700

[23] **Hoffmann E., Ulrich J.:** Hörakustik 2.0. Theorie und Praxis. 2. Aufl. Heidelberg: DOZ-Verlag 2011, S. 701

[24] **Hoffmann E., Ulrich J.:** Hörakustik 2.0. Theorie und Praxis. 2. Aufl. Heidelberg: DOZ-Verlag 2011, S. 721

[25] **Hoffmann E., Ulrich J.:** Hörakustik 2.0. Theorie und Praxis. 2. Aufl. Heidelberg: DOZ-Verlag 2011, S. 722 - 724

[26] **Hoffmann E., Ulrich J.:** Hörakustik 2.0. Theorie und Praxis. 2. Aufl. Heidelberg: DOZ-Verlag 2011, S. 724

[27] **Demtröder W.:** Experimentalphysik 2. Elektrizität und Optik. 4. Aufl. Berlin: Springer 2006, S. 336 - 338

[28] **Lang C., Pucker N.:** Mathematische Methoden in der Physik. 2. Aufl. München: Spektrum Akademischer Verlag 2005, S. 411 f.

6. Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel verfasst habe. Alle Passagen, die ich wörtlich oder inhaltlichen aus der Literatur oder anderen Quellen übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht. Ich erkläre weiterhin, dass die vorliegende Arbeit, in gleicher oder ähnlicher Form, bei keiner anderen Prüfungsbehörde eingereicht wurde.

Aalen, im Oktober 2013

Judith Ungewiß

A. Fragebogen zur Umfrage: MATLAB in Mathematik

Der Fragebogen, mit welchem die Umfrage MATLAB in Mathematik durchgeführt wurde, befindet sich auf den folgenden Seiten.

Umfrage
Matlab in Mathematik

Matlab ist ein Programmiertool, das in der Augenoptik und Hörakustik zur Programmierung mathematischer Aufgabenstellungen und anderer Sachverhalte mit mathematischen Hintergründen wie beispielsweise der Berechnung von Contactlinsen und Inhalten der Digitalen Signalverarbeitung angewendet werden kann.

1. **Was studierst Du?**
   ☐ A ☐ AH

2. **Wo möchtest Du nach dem Studium am liebsten arbeiten?**
   ☐ Geschäft (Augenoptik)
   ☐ Geschäft (Hörakustik)
   ☐ Klinik
   ☐ Industrie/Forschung
   ☐ Sonstiges: _____

3. **Denkst Du, dass es sinnvoll wäre, während des Studiums (mehr) mit Matlab zu arbeiten?**
   ☐ ja ☐ nein ☐ weiß nicht

4. **Hältst Du den Praxisbezug des Studiums für ausreichend?**
   ☐ ja ☐ nein ☐ weiß nicht

5. **Denkst Du, dass durch die Möglichkeit, während des Studiums Matlab zu lernen, der Praxisbezug des Studiums erhöht werden könnte?**
   ☐ ja ☐ nein ☐ weiß nicht

6. **Fändest Du es sinnvoll, wenn bereits im ersten Semester, begleitend zur Vorlesung Mathematik, eine Einführung in Matlab stattfinden würde?**
   ☐ ja ☐ nein ☐ weiß nicht

7. **Würdest Du ein zusätzlich angebotenes Wahlfach, in dem Matlab thematisiert wird, wählen?**
   ☐ ja ☐ nein ☐ weiß nicht

8. **Wenn Du die Wahl zwischen Matlab und anderen Programmiertools/-sprachen hättest, was würdest Du wählen?**
   ☐ Matlab
   ☐ Andere, nämlich: _____

9. **Wie dringend siehst Du den Bedarf, dass Matlab im Studiengang A/AH gelehrt wird?**
   überhaupt nicht ☐ ☐ ☐ ☐ ☐ sehr dringend
   1 2 3 4 5

10. **Welche Vorschläge hast Du dahingehend, in welcher Form Matlab in das Studium A/AH eingebunden werden sollte?**

    _____

    _____

    _____

11. **Welche Vorschläge hast Du, wie Matlab in die Lehrveranstaltung Mathematik eingebunden werden könnte?**

    _____

    _____

    _____

B. Übersicht über die Ergebnisse der Umfrage

Die Ergebnisse der Umfrage MATLAB in Mathematik werden auf den folgenden Seiten gesammelt dargestellt.

Tabelle B.1.: Ergebnisse der Umfrage: MATLAB in Mathematik

Nr.	Ergebnis Frage 1	Ergebnis Frage 2	Ergebnis Frage 3	Ergebnis Frage 4	Ergebnis Frage 5	Ergebnis Frage 6	Ergebnis Frage 7	Ergebnis Frage 8	Ergebnis Frage 9
1	AH	Industrie/Forschung	ja	nein	ja	ja	ja	MATLAB	4
2	AH	Geschäft Augenoptik, Geschäft Hörakustik	ja	nein	weiß nicht	ja	nein	MATLAB	3
3	AH	Geschäft Hörakustik	weiß nicht	nein	weiß nicht	ja	w	HTML	3
4	AH	CL-Institut	ja	nein	ja	ja	ja	MATLAB	4
5	AH	Klinik	ja	nein	ja	nein	nein	Matlab	4
6	A	Industrie/Forschung	ja	nein	ja	ja	nein	ProE	4
7	AH	Industrie/Forschung	ja	nein	weiß nicht	nein	ja	-	4
8	A	Klinik	weiß nicht	ja	weiß nicht	nein	nein	MATLAB	2
9	AH	Geschäft Hörakustik	weiß nicht	nein	ja	ja	ja	MATLAB	4
10	AH	Geschäft Hörakustik, CL-Institut	weiß nicht	weiß nicht	nein	ja	nein	MATLAB	3
11	AH	CL-Institut	weiß nicht	nein	nein	weiß nicht	nein	MATLAB	4
12	AH	Geschäft Augenoptik, Geschäft Hörakustik	ja	weiß nicht	ja	ja	weiß nicht	-	4
13	AH	Geschäft Hörakustik	ja	nein	ja	ja	ja	MATLAB	4
14	AH	Industrie/Forschung	ja	nein	weiß nicht	weiß nicht	ja	MATLAB	4
15	AH	Klinik	ja	weiß nicht	ja	ja	ja	MATLAB	4
16	AH	Industrie/Forschung	ja	ja	ja	ja	ja	MATLAB	2
17	AH	Geschäft Augenoptik, Geschäft Hörakustik	nein	nein	nein	nein	nein	C++	1
18	AH	Geschäft Augenoptik, Geschäft Hörakustik, Industrie/Forschung	ja	weiß nicht	weiß nicht	ja	weiß nicht	MATLAB	3
19	AH	Klinik	ja	weiß nicht	nein	ja	ja	MATLAB	2
20	AH	Klinik	ja	ja	nein	ja	ja	MATLAB	4
21	A	Industrie/Forschung	weiß nicht	ja	weiß nicht	weiß nicht	weiß nicht	-	3
22	A	Industrie/Forschung	weiß nicht	ja	ja	weiß nicht	ja	MATLAB	2
23	A	Industrie/Forschung	ja	nein	ja	ja	ja	MATLAB	5
24	AH	Klinik	weiß nicht	nein	nein	nein	nein	MATLAB	3
25	AH	Geschäft Augenoptik	nein	nein	nein	nein	nein	-	1
26	A	Industrie/Forschung	ja	nein	ja	ja	weiß nicht	MATLAB	4
27	A	Industrie/Forschung	ja	weiß nicht	ja	weiß nicht	ja	MATLAB	2
28	A	Geschäft Augenoptik	weiß nicht	ja	weiß nicht	nein	nein	C++	1
29	AH	Geschäft Augenoptik, Klinik	weiß nicht	nein	nein	nein	nein	-	2
30	AH	Geschäft Augenoptik	weiß nicht	nein	nein	weiß nicht	weiß nicht	C++	3
31	AH	Klinik	ja	weiß nicht	weiß nicht	ja	nein	-	3
32	AH	Klinik, CL-Institut	nein	nein	nein	nein	nein	-	2
33	A	Industrie/Forschung, CL-Institut	weiß nicht	nein	weiß nicht	ja	nein	C++	3
34	AH	Geschäft Augenoptik	nein	nein	nein	nein	nein	-	4

Antworten auf Frage 10

- im 2. oder 3. Semester als Pflichtfach, um grundsätzlich darauf aufbauen zu können
- es sollte als Pflichtmodul eingeführt werden
- als Wahlfach ausführlich, als Pflicht grob umrissen
- Wahlfach
- Pflichtfach für Projekte wie CL-Drehen
- theoretische Einführung und dann selbst damit arbeiten, siehe CAD-Kurs (Semesterferien)
- KL-Konstruktion, Sklerallinsen (Wahlfach)
- DSV (Pflichtfach)
- als Wahlfach zur Vertiefung
- separates Praktikum
- weiterhin in KL-Technik, aber mit besserer Einführung (evtl. kleiner Kurs vorher)
- Das Programm sollte auf jeden Fall mal vorgestellt werden! Dann macht auch KL-Technik mehr Sinn!
- Praktikum Programmieren (neben Informatik)
- als Wahlfach
- Wahlfach
- Studium Generale oder als Wahlfach, kein Pflichtfach
- vertiefendes Wahlfach
- Wahlfach, statt Visual Basic
- schwierig, bei Hörgeräteoperationen eventuell
- Studium Generale oder Wahlfach
- Wahlfach! Würd ich sofort wählen

- wenn, dann nur als Wahlfach
- Informatik Modul
- Wahlfach für diejenigen, die in Industrie/Forschung gehen wollen

Antworten auf Frage 11

- Rechensysteme für Psychoakustik
- Formelgestaltungen in Hausarbeiten
- begleitend zum Lehrstoff Mathematik als Praktikum
- Praktikum Wellenphysik, mehr Versuche mit Matlab
- Kurvendiskussion
- Tutorien anbieten, ich denke dass mit Matlab größere Verständnisprobleme bei den Studierenden auftreten können
- einfache Grundlagen in Lehrveranstaltung Mathematik
- nicht von Fr. Paffrath
- Mathe 2 einführen und Matlab in Ruhe, Schritt für Schritt durchgehen und beibringen
- wöchentliche Übungsaufgaben

C. Skript zum Themenschwerpunkt Lineare Gleichungssysteme und Matrizen

Das Skript und die Aufgaben, welche den Studierenden zur Unterstützung beim eigenständigen Erarbeiten der Thematik Lineare Gleichungssysteme und Matrizen zur Verfügung gestellt werden sollen, befinden sich auf den folgenden Seiten.
Da dieses Skript lediglich dem Verständnis der Studierenden dienen soll und eine durchgehende Nummerierung der Gleichungen für ein solches Verständnis, wie auch für die übersichtliche Darstellung der Inhalte, nicht notwendig ist, wird darauf verzichtet.

Lineare Gleichungssysteme und Matrizen

1. Lineare Gleichungssysteme

Ein lineares Gleichungssystem (LGS) besteht aus einem Satz von Gleichungen. Allgemein lässt sich ein LGS mit m Gleichungen und n Unbekannten wie folgt darstellen:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \quad (1)$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \quad (2)$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \quad (m)$$

Dabei nennt man $a_{11}, \ldots a_{mn}$ Koeffizienten, $x_1, \ldots x_n$sind die Variablen oder Unbekannten.

Grundsätzlich kann ein LGS keine, eine oder unendlich viele Lösungen haben, die auf unterschiedliche Arten dargestellt werden können. Hierauf wird nachfolgend ebenso eingegangen, wie auf die verschiedenen Lösungsverfahren für LGS.

1.1. Lösbarkeit

Bei der Lösung von LGS treten drei verschiedene Fälle auf:

- Das LGS hat keine Lösung.
  Dieser Fall tritt immer dann ein, wenn mehrere Gleichungen im Widerspruch zueinander stehen. Besonders, wenn ein LGS mehr Gleichungen als Unbekannte hat, hat es oft keine Lösung, weil die Variablen nicht alle geforderten Kriterien erfüllen können.

Beispiel

$$x_1 = 3 \tag{1}$$
$$2x_1 = 3 \tag{2}$$

Hier ist keine Variable x zu bestimmen, die für beide Gleichungen zu einer richtigen Aussage führt. Das LGS hat somit keine Lösung.

- Das LGS hat genau eine Lösung.
  Genau eine Lösung ist immer dann vorhanden, wenn die Anzahl der Gleichungen m der Anzahl der Variablen n entspricht und die Variablen alle geforderten Kriterien erfüllen. Dies ist der Normalfall, der auch auf das in Kapitel 1.2 angegebene Beispiel zutrifft.

- Das LGS hat unendlich viele Lösungen.
  Ein LGS hat unendlich viele Lösungen, wenn weniger verschiedene Gleichungen als Variablen vorhanden sind.
  Beispiel

$$x_1 + x_2 = 6 \tag{1}$$

Hier könnte zum Beispiel $x_1 = 2$ und $x_2 = 4$; $x_1 = 1$ und $x_2 = 5$ etc. sein, somit ist das LGS nicht eindeutig lösbar, es bestehen unendlich viele Lösungen. Soll trotzdem eine allgemeingültige Lösung angegeben werden, so wird für eine frei wählbare Unbekannte x_i eine Variable gewählt, anhand derer das LGS dann weiter aufgelöst wird.
Wählt man also exemplarisch $x_2 = t$ und setzt dies in Gleichung (1), so folgt: $x_1 + t = 6$ und damit $x_1 = 6 - t$.
Die Lösung des LGS wäre also: $x_1 = 6 - t$ und $x_2 = t$.

1.2. Lösungsverfahren

Ein LGS kann mittels verschiedener Verfahren gelöst werden, die hier erklärt werden. Zur Verdeutlichung wird folgendes LGS mit jeder der vorgestellten Methoden beispielhaft nach x_1 und x_2 aufgelöst. Es bestehen grundsätzlich immer mehrere Möglichkeiten der Lösung des

LGS. Die hier vorgestellten Lösungswege sind somit lediglich als Lösungsbeispiel anzusehen.

$$2x_1 + x_2 = 9 \tag{1}$$
$$x_1 - x_2 = -3 \tag{2}$$

1.2.1. Einsetzungsverfahren

Beim Einsetzungsverfahren wird eine Gleichung des LGS nach einer Variablen aufgelöst. Diese Variable wird dann in die anderen Gleichungen eingesetzt, wodurch eine der Variablen eliminiert wird, also ein LGS mit $n-1$ Gleichungen entsteht. Durch Wiederholung dessen kann auch ein LGS mit mehr als zwei Variablen auf eine Gleichung mit einer Variablen reduziert werden, die dann berechnet und eingesetzt werden kann, um das LGS zu lösen.

(2) wird nach x_1 umgestellt:	$x_1 = x_2 - 3$	(2a)
(2a) wird in (1) eingesetzt:	$2 \cdot (x_2 - 3) + x_2 = 9$	(1a)
(1a) kann nun aufgelöst werden:	$2x_2 - 6 + x_2 = 9$	
	$3x_2 - 6+ = 9$	
	$3x_2 = 15$	
	$x_2 = 5$	
x_2 wird in (2) eingesetzt:	$x_1 - 5 = -3$	
(2b) kann nun aufgelöst werden:	$x_1 = 2$	

1.2.2. Gleichsetzungsverfahren

Beim Gleichsetzungsverfahren werden zwei Gleichungen so umgestellt, dass ihre jeweils linke Seite gleich ist und nur eine Variable enthält. Diese Variable darf auf der rechten Seite der Gleichung nicht vorhanden sein. Durch Gleichsetzen der beiden rechten Seiten, hängt die dabei entstehende Gleichung von einer Variablen weniger ab.

(1) wird nach x_1 umgestellt:	$x_2 = 9 - 2x_1$	(1a)
(2) wird nach x_2 umgestellt:	$x_2 = x_1 + 3$	(2a)

Die beiden linken Seiten sind identisch, weshalb auch die beiden rechten Seiten identisch sein müssen und gleichgesetzt werden:

$$9 - 2x_1 = x_1 + 3 \tag{3}$$

(3) wird aufgelöst:

$$\begin{aligned} x_1 + 2x_1 &= 6 \\ 3x_1 &= 6 \\ x_1 &= 2 \end{aligned}$$

Diese Lösung kann nun in eine der ursprünglichen Gleichungen eingesetzt werden, um die übrige Variable zu berechnen:

x_1 wird in (2) eingesetzt: $2 - x_2 = -3$ (2b)

(2b) wird nach x_2 aufgelöst: $x_2 = 5$

1.2.3. Additionsverfahren

Beim Additionsverfahren werden Gleichungen addiert (oder subtrahiert), so dass eine oder mehrere Variablen dadurch entfernt werden und das LGS somit eine Variable weniger enthält. So kann das LGS auf eine Gleichung mir einer Variablen reduziert werden, die dann berechnet und in die ursprünglichen Gleichungen eingesetzt werden kann, um die übrigen Variablen zu berechnen.

(1) + (2):

$$\begin{aligned} 2x_1 + x_2 &= 9 \\ x_1 - x_2 &= -3 \\ 3x_1 &= 6 \qquad (3) \end{aligned}$$

(3) kann aufgelöst werden: $x_1 = 2$

x_1 kann in (2) eingesetzt werden: $2 - x_2 = -3$ (2a)

(2a) kann aufgelöst werden: $x_2 = 5$

1.3. Lösungsmenge und Lösungsvektor

Die Lösung eines LGS kann als Lösungsmenge oder als Lösungsvektor angegeben werden. Die Lösungsmenge bzw. der Lösungsvektor besteht aus allen x_i, für die alle Gleichungen des LGS erfüllt sind.

Die Lösungen werden folgendermaßen dargestellt:

- als Lösungsmenge: $L = \{x_1, x_2, \ldots, x_n\}$
- als Lösungsvektor: $\vec{L} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$

Für das in Kapitel 1.2 beispielhaft gelöste LGS ist die Lösung $x_1 = 2$ und $x_2 = 5$. Diese Lösung wird wie folgt dargestellt:

- als Lösungsmenge: $L = \{2, 5\}$
- als Lösungsvektor: $\vec{L} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$

Hat ein LGS keine Lösung, so ist die Lösungsmenge leer. Eine Darstellung als Lösungsvektor ist dabei nicht möglich, weshalb die nur als Lösungsmenge dargestellt wird: $L = \{\}$

2. Matrizen

Eine Matrix ist eine Tabelle, d.h eine rechteckige Anordnung mathematischer Objekte wie Zahlen. Mit diesen Objekten kann auf bestimmte Art gerechnet werden. Hierbei können mehrere Matrizen addiert (oder subtrahiert) und multipliziert werden, worauf im Verlauf dieses Kapitels eingegangen wird.
Insbesondere werden Matrizen benutzt, um LGS darzustellen und zu lösen.

Die Elemente einer Matrix werden in Zeilen und Spalten angeordnet, welche in eckigen oder runden Klammern dargestellt werden können. Gebräuchlicher ist die Darstellung in runden Klammern, weshalb diese hier genutzt wird. Eine Matrix wird üblicherweise mit einem Großbuchstaben benannt.
Eine Matrix M_1 mit m Zeilen und n Spalten ist wie folgt zu notieren:

$$M_1 = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

2.1. Zusammenhang zwischen LGS und Matrizen

LGS können mithilfe von Matrizen gelöst werden. Die entsprechende Darstellung ist vor allem bei großen LGS bzw. Matrizen mit vielen Zeilen und Spalten sinnvoll, da so eine bessere Übersicht gegeben ist.
Zur Lösung werden die Koeffizienten des LGS in einer Matrix dargestellt. Die jeweiligen Lösungen der einzelnen Gleichungen werden dann durch einen senkrechten Strich von den Koeffizienten getrennt. Die dem in Kapitel 1.2 vorgestellten LGS zugehörige Matrix sieht folgendermaßen aus:

$$M_2 = \left(\begin{array}{cc|c} 2 & 1 & 9 \\ 1 & -1 & -3 \end{array}\right)$$

Das LGS kann mit dem Gauß-Algorithmus (Gaußsches Eliminationsverfahren) gelöst werden. Die Matrizendarstellung ist auch hierbei vor allem aufgrund der besseren Übersichtlichkeit zu nutzen.
Das Ziel des Verfahrens ist, das LGS auf eine Stufenform zu bringen, aus der die Lösung des LGS durch sukzessives Einsetzen der bereits ermittelten Variablen einfach zu berechnen ist. Stufenform bedeutet, dass pro Zeile mindestens eine Variable wegfällt. Ein LGS mit drei Gleichungen sieht in Stufenform folgendermaßen aus:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \quad (1)$$
$$a_{22}x_2 + a_{23}x_3 = b_2 \quad (2)$$
$$a_{33}x_3 = b_3 \quad (3)$$

Die zugehörige Matrix ist dann:

$$M_3 = \left(\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a_{22} & a_{23} & b_2 \\ 0 & 0 & a_{33} & b_3 \end{array}\right)$$

Um die Stufenform zu erreichen, werden elementare Zeilenumformungen vorgenommen. Dadurch entsteht ein neues LGS bzw. eine neue Matrix mit derselben Lösungsmenge.
Folgende Unformungen sind erlaubt:

- Vertauschen von Zeilen
- Multiplizieren oder Dividieren einer kompletten Zeile mit einem Skalar, d.h einer ein-

fachen Zahl

- eine Zeile zu einer anderen Zeile addieren oder von einer anderen Zeile subtrahieren

Das Gauß-Verfahren wird an einem Beispiel verdeutlicht:

Beispiel 1 Folgendes LGS soll gelöst werden:

$$2x_1 + x_2 + x_3 = 3 \tag{1}$$
$$x_1 - 3x_2 + 2x_3 = -7 \tag{2}$$
$$x_1 + x_2 - 2x_3 = 5 \tag{3}$$

Zunächst wird das LGS der besseren Übersicht wegen als Matrix D dargestellt:

$$D = \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 1 & -3 & 2 & -7 \\ 1 & 1 & -2 & 5 \end{array}\right)$$

Diese Matrix wird nun auf die Stufenform gebracht. Die durchzuführenden Umformungen werden dabei auf den Pfeilen angegeben, die jeweils umgeformte/n Zeile/n wird in eckigen Klammern dargestellt.

Dies ist nur eine von vielen möglichen Darstellungsformen. Die Matrizen können beispielsweise auch untereinander geschrieben werden, die jeweils durchgeführte Umformung wird dann, wie bei der Äquivalenzumformung, neben der Matrix angegeben. Wichtig ist bei jeder Darstellungsform lediglich die Nachvollziehbarkeit der durchgeführten Umformungen.

$$D = \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 1 & -3 & 2 & -7 \\ 1 & 1 & -2 & 5 \end{array}\right) \overset{[2]-[3]}{\rightarrow} \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 0 & -4 & 4 & -12 \\ 1 & 1 & -2 & 5 \end{array}\right) \overset{2\cdot[3]}{\rightarrow} \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 0 & -4 & 4 & -12 \\ 2 & 2 & -4 & 10 \end{array}\right)$$

$$\overset{[3]-[1]}{\rightarrow} \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 0 & -4 & 4 & -12 \\ 0 & 1 & -5 & 7 \end{array}\right) \overset{\frac{[2]}{4}}{\rightarrow} \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 0 & -1 & 1 & -3 \\ 0 & 1 & -5 & 7 \end{array}\right) \overset{[3]+[2]}{\rightarrow} \left(\begin{array}{ccc|c} 2 & 1 & 1 & 3 \\ 0 & -1 & 1 & -3 \\ 0 & 0 & -4 & 4 \end{array}\right)$$

Zeile 3 ergibt nun eine Gleichung, die nur noch von einer Variablen abhängt und sich auflösen

lässt:

$$-4x_3 = 4 \tag{3a}$$
$$x_3 = -1$$

Zeile 2 ergibt eine Gleichung, die von zwei Variablen abhängt. Da x_3 bereits ermittelt wurde, kann der Wert für x_3 in diese Gleichung eingesetzt werden, wodurch die Gleichung von nur noch einer Variablen abhängig ist und aufgelöst werden kann:

$$-x_2 + x_3 = 3 \tag{2a}$$
x_3 in (2a) einsetzen: $$-x_2 - 1 = -3 \tag{2b}$$
(2b) kann aufgelöst werden: $$-x_2 = -2$$
$$x_2 = 2$$

Die Gleichung in Zeile 1 der Matrix ist von drei Variablen abhängig, von denen zwei nun bereits ermittelt wurden. Werden diese eingesetzt, so lässt sich auch die übrige Gleichung lösen:

$$2x_1 + x_2 + x_3 = 3 \tag{1a}$$
x_2 und x_3 in (1a) einsetzen: $$2x_1 + 2 - 1 = 3 \tag{1b}$$
(1b) kann aufgelöst werden: $$2x_1 + 1 = 3$$
$$2x_1 = 2$$
$$x_1 = 1$$

Die Lösungsmenge des LGS ist somit $L = \{1, 2, -1\}$.

Der Lösungsvektor ist $\vec{L} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$.

2.2. Rechenoperationen

Die folgenden Rechenoperationen werden anhand der Matrizen

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 3 & 5 \\ 1 & 0 & 1 \end{pmatrix} \text{ und } B = \begin{pmatrix} 2 & 2 & 5 \\ 3 & 1 & 1 \\ 0 & 2 & 2 \end{pmatrix}$$

beispielhaft vorgestellt.

2.2.1. Matrizenaddition

Zwei Matrizen können immer dann addiert bzw. subtrahiert werden, wenn sie dieselbe Anzahl Zeilen und dieselbe Anzahl Spalten haben. Die Summe bzw. Differenz zweier Matrizen wird berechnet, indem jeweils die Einträge an derselben Stelle der Matrizen addiert nzw. subtrahiert werden.

Beispiel 2

$$A + B = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 3 & 5 \\ 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 & 5 \\ 3 & 1 & 1 \\ 0 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 1+2 & 2+2 & 1+5 \\ 0+3 & 3+1 & 5+1 \\ 1+0 & 0+2 & 1+2 \end{pmatrix} = \begin{pmatrix} 3 & 4 & 6 \\ 3 & 4 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

Beispiel 3

$$A - B = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 3 & 5 \\ 1 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 2 & 2 & 5 \\ 3 & 1 & 1 \\ 0 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 1-2 & 2-2 & 1-5 \\ 0-3 & 3-1 & 5-1 \\ 1-0 & 0-2 & 1-2 \end{pmatrix} = \begin{pmatrix} -1 & 0 & -4 \\ -3 & 2 & 4 \\ 1 & -2 & -1 \end{pmatrix}$$

2.2.2. Skalarmultiplikation

Eine Matrix kann mit einem Skalar, also einer einfachen Zahl, multipliziert werden, indem alle Eintäge der Matrix mit dem Skalar multipliziert werden.

Beispiel 4

$$5 \cdot A = 5 \cdot \begin{pmatrix} 1 & 2 & 1 \\ 0 & 3 & 5 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 5 \cdot 1 & 5 \cdot 2 & 5 \cdot 1 \\ 5 \cdot 0 & 5 \cdot 3 & 5 \cdot 5 \\ 5 \cdot 1 & 5 \cdot 0 & 5 \cdot 1 \end{pmatrix} = \begin{pmatrix} 5 & 10 & 5 \\ 0 & 15 & 25 \\ 5 & 0 & 5 \end{pmatrix}$$

2.2.3. Matrizenmultiplikation

Zwei Matrizen können miteinander multipliziert werden, wenn die Spaltenanzahl der linken mit der Zeilenanzahl der rechten Matrix übereinstimmt. Für die Berechnung gilt die Regel „Zeile mal Spalte“. Das bedeutet, dass die Einträge der ersten Zeile der linken Matrix mit den Einträgen der ersten Spalte der rechten Matrix multipliziert und anschließend addiert werden, um den ersten Eintrag (a_{11}) der entstehenden Matrix zu erhalten. Die Einträge der

ersten Zeile der linken Matrix werden dann mit den Einträgen der zweiten Spalte der rechten Matrix multipliziert und anschließend addiert, um den zweiten Eintrag (a_{12}) der entstehenden Matrix zu erhalten etc.

Beispiel 5

$$A \cdot B = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 3 & 5 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 2 & 5 \\ 3 & 1 & 1 \\ 0 & 2 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \cdot 2 + 2 \cdot 3 + 1 \cdot 0 & 1 \cdot 2 + 2 \cdot 1 + 1 \cdot 2 & 1 \cdot 5 + 2 \cdot 1 + 1 \cdot 2 \\ 0 \cdot 2 + 3 \cdot 3 + 5 \cdot 0 & 0 \cdot 2 + 3 \cdot 1 + 5 \cdot 2 & 0 \cdot 5 + 3 \cdot 1 + 5 \cdot 2 \\ 1 \cdot 2 + 0 \cdot 3 + 1 \cdot 0 & 1 \cdot 2 + 0 \cdot 1 + 1 \cdot 2 & 1 \cdot 5 + 0 \cdot 1 + 1 \cdot 2 \end{pmatrix}$$

$$= \begin{pmatrix} 2 + 6 + 0 & 2 + 2 + 2 & 5 + 2 + 2 \\ 0 + 9 + 0 & 0 + 3 + 10 & 0 + 3 + 10 \\ 2 + 0 + 0 & 1 + 0 + 3 & 5 + 0 + 2 \end{pmatrix} = \begin{pmatrix} 8 & 6 & 9 \\ 9 & 13 & 13 \\ 2 & 4 & 7 \end{pmatrix}$$

Die Matrizenmultiplikation ist

- nicht kommutativ, d.h. es gilt $A \cdot B \neq B \cdot A$. Es muss also immer auf die richtige Reihenfolge geachtet werden.
- assoziativ, d.h. Klammern dürfen getauscht werden: $(A \cdot B) \cdot C = A \cdot (B \cdot C)$.

Matrizenmultiplikation und -addition genügen zudem dem Distributivgesetz, d.h. es darf ausmultipliziert werden: $A \cdot (B + C) = A \cdot B + A \cdot C$.

2.2.4. Die transponierte Matrix

Die Transponierte einer Matrix entsteht, indem die Spalten und Zeilen einer Matrix getauscht werden. Die erste Zeile wird zur ersten Spalte, die zweite Zeile wird zur zweiten Spalte usw. Die transponierte Matrix wird gegenüber der Originalmatrix mit einem hochgestellten T gekennzeichnet. Allgemein ist die transponierte Matrix wie folgt darzustellen:

$$M_1^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix}$$

Beispiel 6 Die Transponierte der oben angegebenen Matrix A ist:

$$A^T = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 0 \\ 1 & 5 & 1 \end{pmatrix}$$

3. Lineare Gleichungssysteme und Matrizen mit MATLAB

Matrizen können mit MATLAB besonders gut berechnet werden. Für die in Kapitel 2 berechneten Beispiele werden hier exemplarisch die Musterlösungen bezüglich der Eingaben in das Command Window dargestellt.

Beispiel 1
Hier bestehen mehrere Möglichkeiten der Lösung. Zum Einen können lineare Gleichungssysteme mit dem `solve`-Befehl gelöst werden, zum Anderen kann das Gleichungssystem in eine Matrix und einen Lösungsvektor umgeschrieben und so gelöst werden. Sind beide Lösungen möglich, so werden auch beide Lösungen angegeben.

Lösung als LGS:

```
>> syms x1 x2 x3
>> S=solve(2*x1+x2+x3==3,x1-3*x2+2*x3==-7,x1+x2-2*x3==5);
>> S=[S.x1 S.x2 S.x3]
```

Lösung als Matrix:

```
>> A=[2 1 1;1 -3 2; 1 1 -2];
>> b=[3;-7;5];
>> x=A\b
```

Beispiel 2

```
>> A=[1 2 1;0 3 5;1 0 1];
>> B=[2 2 5;3 1 1;0 2 2];
>> A+B
```

Beispiel 3

```
>> A=[1 2 1;0 3 5;1 0 1];
>> B=[2 2 5;3 1 1;0 2 2];
```

```
>> A-B
```

Beispiel 4

```
>> A=[1 2 1;0 3 5;1 0 1];
>> B=[2 2 5;3 1 1;0 2 2];
>> 5*A
```

Beispiel 5

```
>> A=[1 2 1;0 3 5;1 0 1];
>> B=[2 2 5;3 1 1;0 2 2];
>> A*B
```

Beispiel 6

```
>> A=[1 2 1;0 3 5;1 0 1];
>> A'
```

4. Aufgaben

Die folgenden Aufgaben sind denjenigen, die bereits online verfügbar sind, entnommen.

Aufgabe 1

Berechnen Sie alle Lösungen des folgenden LGS:

$$-3x_1 + 2x_2 - 3x_3 = 6 \tag{1}$$
$$9x_1 - 2x_2 + 10x_3 = -10 \tag{2}$$
$$6x_1 + 8x_2 + 14x_3 = 22 \tag{3}$$

Aufgabe 2

Gegeben sind folgende Matrizen:

$$A = \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 3 \\ 2 & -1 \end{pmatrix} \qquad C = \begin{pmatrix} 1 & 0 & -4 \\ 3 & 2 & 0 \\ 6 & -2 & 5 \end{pmatrix} \qquad D = \begin{pmatrix} -4 & 0 & 3 \\ -1 & 4 & 7 \end{pmatrix}$$

Berechnen Sie:

a) $A + 3A$

b) $A - 3D^T$

c) $B \cdot D$

d) $D^T \cdot B$

e) $C \cdot D^T - 2A$

5. Lösungen

Lösung 1

$$A_1 = \left(\begin{array}{ccc|c} -3 & 2 & -3 & 6 \\ 9 & -2 & 10 & -10 \\ 6 & 8 & 14 & 22 \end{array}\right) \overset{[2]+[1]}{\rightarrow} \left(\begin{array}{ccc|c} -3 & 2 & -3 & 6 \\ 6 & 0 & 7 & -4 \\ 6 & 8 & 14 & 22 \end{array}\right) \overset{[3]+[2]}{\rightarrow} \left(\begin{array}{ccc|c} -3 & 2 & -3 & 6 \\ 6 & 0 & 7 & -4 \\ 0 & 8 & 7 & 26 \end{array}\right)$$

$$\overset{\frac{[2]}{2}}{\rightarrow} \left(\begin{array}{ccc|c} -3 & 2 & -3 & 6 \\ 3 & 0 & \frac{7}{2} & -2 \\ 0 & 8 & 7 & 26 \end{array}\right) \overset{[1]+[2]}{\rightarrow} \left(\begin{array}{ccc|c} 0 & 2 & \frac{1}{2} & 4 \\ 3 & 0 & \frac{7}{2} & -2 \\ 0 & 8 & 7 & 26 \end{array}\right) \overset{[1]\cdot 4}{\rightarrow} \left(\begin{array}{ccc|c} 0 & 8 & 2 & 16 \\ 3 & 0 & \frac{7}{2} & -2 \\ 0 & 8 & 7 & 26 \end{array}\right)$$

$$\overset{[3]-[1]}{\rightarrow} \left(\begin{array}{ccc|c} 0 & 8 & 2 & 16 \\ 3 & 0 & \frac{7}{2} & -2 \\ 0 & 0 & 5 & 10 \end{array}\right)$$

$$5x_3 = 10 \tag{3a}$$
$$x_3 = 2$$

Aus (1) und (2) folgt:

$$A_2 = \left(\begin{array}{cc|c} -3 & 2 & 12 \\ 9 & -2 & -30 \end{array}\right) \overset{[1]+[2]}{\rightarrow} \left(\begin{array}{cc|c} 6 & 0 & -18 \\ 9 & -2 & 30 \end{array}\right)$$

$$6x_1 = -18 \tag{1a}$$
$$x_1 = -3$$

$$2x_2 = 3 \tag{2a}$$
$$x_2 = \frac{3}{2}$$

Lösung mit MATLAB:

```
>> syms x1 x2 x3
>> S=solve(-3*x1+2*x2-3*x3==6,9*x1-2*x2+10*x3==-10,6*x1+8*x2+14*x3==22);
>> S=[S.x1 S.x2 S.x3]
```

Alternative:

```
>> A=[-3 2 -3;9 -2 10; 6 8 14];
>> b=[6;-10;22];
>> x=A\b
```

Lösung 2

a) $A + 3 \cdot A = \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix} + 3 \cdot \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix} = \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix} + \begin{pmatrix} -9 & 6 \\ 3 & 9 \\ 15 & -12 \end{pmatrix}$

$= \begin{pmatrix} -3-9 & 2+6 \\ 1+3 & 3+9 \\ 5+15 & -4-12 \end{pmatrix} = \begin{pmatrix} -12 & 8 \\ 4 & 12 \\ 20 & -16 \end{pmatrix}$

b) $A - 3 \cdot D^T = \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix} - 3 \cdot \begin{pmatrix} -4 & -1 \\ 0 & 4 \\ 3 & 7 \end{pmatrix} = \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix} - \begin{pmatrix} -12 & -3 \\ 0 & 12 \\ 9 & 21 \end{pmatrix}$

$= \begin{pmatrix} -3+12 & 2+3 \\ 1-0 & 3-12 \\ 5-9 & -4-21 \end{pmatrix} = \begin{pmatrix} 9 & 5 \\ 1 & -9 \\ -4 & -25 \end{pmatrix}$

c) $B \cdot D = \begin{pmatrix} 1 & 3 \\ 2 & -1 \end{pmatrix} \cdot \begin{pmatrix} -4 & 0 & 3 \\ -1 & 4 & 7 \end{pmatrix}$

$= \begin{pmatrix} 1 \cdot (-4) + 3 \cdot (-1) & 1 \cdot 0 + 3 \cdot 4 & 1 \cdot 3 + 3 \cdot 7 \\ 2 \cdot (-4) + (-1) \cdot (-1) & 2 \cdot 0 + (-1) \cdot 4 & 2 \cdot 3 + (-1) \cdot 7 \end{pmatrix}$

$$= \begin{pmatrix} (-4)+(-3) & 0+12 & 3+21 \\ (-8)+1 & 0+(-4) & 6+(-7) \end{pmatrix} = \begin{pmatrix} -7 & 12 & 24 \\ -7 & -4 & -1 \end{pmatrix}$$

d) $D^T \cdot B = \begin{pmatrix} -4 & -1 \\ 0 & 4 \\ 3 & 7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 3 \\ 2 & -1 \end{pmatrix}$

$$= \begin{pmatrix} (-4)\cdot 1+(-1)\cdot 2 & (-4)\cdot 3+(-1)\cdot(-1) \\ 0\cdot 1+4\cdot 2 & 0\cdot 3+4\cdot(-1) \\ 3\cdot 1+7\cdot 2 & 3\cdot 3+7\cdot(-1) \end{pmatrix}$$

$$= \begin{pmatrix} (-4)+(-2) & (-12)+1 \\ 0+8 & 0+(-4) \\ 3+14 & 9+(-7) \end{pmatrix} = \begin{pmatrix} -6 & -11 \\ 8 & -4 \\ 17 & 2 \end{pmatrix}$$

e) $C \cdot D^T - 2 \cdot A = \begin{pmatrix} 1 & 0 & -4 \\ 3 & 2 & 0 \\ 6 & -2 & 5 \end{pmatrix} \cdot \begin{pmatrix} -4 & -1 \\ 0 & 4 \\ 3 & 7 \end{pmatrix} - 2 \cdot \begin{pmatrix} -3 & 2 \\ 1 & 3 \\ 5 & -4 \end{pmatrix}$

$$= \begin{pmatrix} 1 & 0 & -4 \\ 3 & 2 & 0 \\ 6 & -2 & 5 \end{pmatrix} \cdot \begin{pmatrix} -4 & -1 \\ 0 & 4 \\ 3 & 7 \end{pmatrix} - \begin{pmatrix} -6 & 4 \\ 2 & 6 \\ 10 & -8 \end{pmatrix}$$

$$= \begin{pmatrix} 1\cdot(-4)+0\cdot 0+(-4)\cdot 3 & 1\cdot(-1)+0\cdot 4+(-4)\cdot 7 \\ 3\cdot(-4)+2\cdot 0+0\cdot 3 & 3\cdot(-1)+2\cdot 4+0\cdot 7 \\ 6\cdot(-4)+(-2)\cdot 0+5\cdot 3 & 6\cdot(-1)+(-2)\cdot 4+5\cdot 7 \end{pmatrix} - \begin{pmatrix} -6 & 4 \\ 2 & 6 \\ 10 & -8 \end{pmatrix}$$

$$= \begin{pmatrix} (-4)+0+(-12) & (-1)+0+(-28) \\ (-12)+0+0 & (-3)+8+0 \\ (-24)+0+15 & (-6)+(-8)+35 \end{pmatrix} - \begin{pmatrix} -6 & 4 \\ 2 & 6 \\ 10 & -8 \end{pmatrix}$$

$$= \begin{pmatrix} -16 & -29 \\ -12 & 5 \\ -9 & 21 \end{pmatrix} - \begin{pmatrix} -6 & 4 \\ 2 & 6 \\ 10 & -8 \end{pmatrix} = \begin{pmatrix} -10 & -33 \\ -14 & -1 \\ -19 & 29 \end{pmatrix}$$

Lösung mit MATLAB:

a)
```
>> A=[-3 2;1 3;5 -4];
>> A+3*A
```

b)
```
>> A=[-3 2;1 3;5 -4];
>> D=[-4 0 3;-1 4 7];
>> A-3*D'
```

c)
```
>> B=[1 3;2 -1];
>> D=[-4 0 3;-1 4 7];
>> B*D
```

d)
```
>> B=[1 3;2 -1];
>> D=[-4 0 3;-1 4 7];
>> D'*B
```

e)
```
>> A=[-3 2;1 3;5 -4];
>> C=[1 0 -4;3 2 0;6 -2 5];
>> D=[-4 0 3;-1 4 7];
>> C*D'-2*A
```

D. Skript zum Themenschwerpunkt Mehrdimensionale Integration

Das Skript und die Aufgaben, welche den Studierenden zur Unterstützung beim eigenständigen Erarbeiten der Thematik Mehrdimensionale Integration zur Verfügung gestellt werden sollen, befinden sich auf den folgenden Seiten.
Da dieses Skript lediglich dem Verständnis der Studierenden dienen soll und eine durchgehende Nummerierung der Gleichungen für ein solches Verständnis, wie auch für die übersichtliche Darstellung der Inhalte, nicht notwendig ist, wird darauf verzichtet.

Mehrdimensionale Integration

Nachfolgend wird auf die Integration über mehrere Dimensionen, beispielsweise zur Volumenberechnung, sowie die Integration in Polar- und Kugelkoordinaten eingegangen. Vorausgesetzt werden dazu Kenntnisse der Integralrechnung zur Flächenberechnung in zweidimensionalen kartesischen Koordinaten.

1. Integration über mehrdimensionale Bereiche

Der Integralbegriff kann derart verallgemeinert werden, dass die Definitionsmenge nicht mehr nur eine Dimension im Sinne einer Zahlengeraden $\mathbb{R}$, sondern mehrere Dimensionen umfasst. So kann dann über Flächen oder Körper integriert werden, wodurch die Integration beispielsweise zur Volumenberechnung genutzt werden kann.

Mehrdimensionale Integrale kann man berechnen, indem man sie in beliebiger Reihenfolge in Integrale über die einzelnen Koordinaten aufspaltet. Die einzelnen Integrale über die jeweiligen Koordinaten werden dann folgendermaßen nacheinander, immer von innen nach außen, berechnet:

$$\iiint f(x,y,z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = \left(\int\left(\int\left(\int f(x,y,z)\,\mathrm{d}z\right)\mathrm{d}y\right)\mathrm{d}x\right)$$

Dabei werden die jeweils übrigen Variablen als Konstanten behandelt. Die Integrationsgrenzen sind aus den Begrenzungen der Fläche bzw. des Volumens zu bestimmen. Diese Art der Integration kann nicht nur bei Rechnung in kartesischen, sondern auch in Polar- oder Kugelkoordinaten angewandt werden.

Ist der Integrand ein Produkt aus Funktionen, wobei in jedem Faktor nur jeweils eine Variable vorkommt, so muss das Integral nicht zwingend von innen nach außen berechnet werden, sondern kann folgendermaßen zerlegt und vereinfacht berechnet werden:

$$\int_{x=a}^{b}\int_{y=c}^{d} f(x)g(y)\mathrm{d}(x,y) = \left(\int_{x=a}^{b} f(x)\mathrm{d}x\right)\cdot\left(\int_{y=c}^{d} g(y)\mathrm{d}y\right)$$

1.1. Kartesische Koordinaten

Bei der Integration in kartesischen Koordinaten wird stets auf das Flächenelement $\mathrm{d}x\mathrm{d}y$ zurückgegriffen.

1.1.1. Beispiel 1

Zu berechnen ist das folgende Integral: $\int\limits_G x^2 cosy \,\mathrm{d}(x,y)$

Dabei ist *G* das Rechteck $|x| \leq 1, \quad 0 \leq y \leq \frac{\pi}{2}$.

G ist das Gebiet, über dem integriert werden soll.

Das Integral wird deshalb zunächst nach den angegebenen Grenzen des Rechtecks parametrisiert, also die Grenzen der x- und y-Koordinaten werden auf zwei Integrale aufgeteilt, die später einzeln berechnet werden können:

$\int\limits_G x^2 cosy \,\mathrm{d}(x,y) = \int\limits_{x=-1}^{1} \int\limits_{y=0}^{\frac{\pi}{2}} x^2 cosy \,\mathrm{d}y\mathrm{d}x$

Nun wird das Integral aufgespalten, die entstehenden Teil-Integrale werden einzeln berechnet:

$\int\limits_{x=-1}^{1} \int\limits_{y=0}^{\frac{\pi}{2}} x^2 cosy \,\mathrm{d}y\mathrm{d}x = \int\limits_{x=-1}^{1} x^2 \,\mathrm{d}x \cdot \int\limits_{y=0}^{\frac{\pi}{2}} cosy \,\mathrm{d}y = \frac{1}{3}x^3\big|_{-1}^{1} \cdot siny\big|_{0}^{\frac{\pi}{2}}$

$= \left[\frac{1}{3}(1)^3 - \frac{1}{3}(-1)^3\right] \cdot \left[sin(\frac{\pi}{2}) - sin(0)\right] = \left(\frac{1}{3} - \left(-\frac{1}{3}\right)\right) \cdot \left(1 - 0\right) = \frac{2}{3} \cdot 1 = \frac{2}{3}$

1.2. Polarkoordinaten

Statt in kartesischen Koordinaten können Strecken im zweidimensionalen Raum auch durch Polarkoordinaten beschrieben werden. Dabei werden dann nicht zwei Punkte angegeben, sondern die Länge der Strecke *r* vom Ursprung aus und der Winkel φ zur x-Achse (vgl. Darstellung komplexer Zahlen in Polarkoordinaten). Für den Wertebereich von φ gilt daher: $0 \leq \varphi \leq 2\pi$ (beruhend auf dem Kreisumfang des Einheitskreises).

1.2.1. Beispiel 2

Darstellung einer Strecke mit $r = 5$ und $\varphi = 30°$

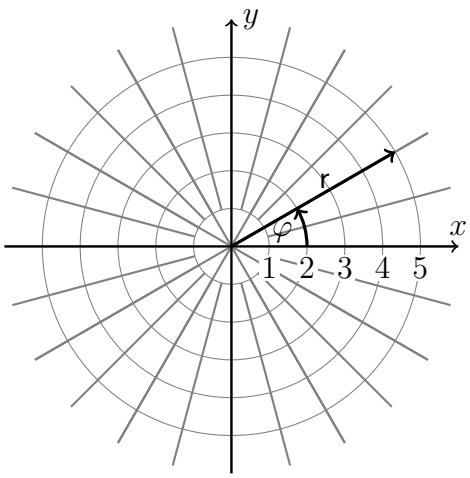


Abbildung D.1.: Polarkoordinaten

Für den Zusammenhang zwischen kartesischen und Polarkoordinaten gilt:

$x = r \cdot cos(\varphi) \quad r \geq 0$

$y = r \cdot sin(\varphi) \quad 0 \leq \varphi \leq 2\pi$

Bei der Integration in Polarkoordinaten wird stets auf das Flächenelement $r\mathrm{d}r\mathrm{d}\varphi$ zurückgegriffen.

1.2.2. Beispiel 3

Zu berechnen ist die Fläche eines Halbkreisrings *R*: $4 \leq x^2 + y^2 \leq 9, y \geq 0.$

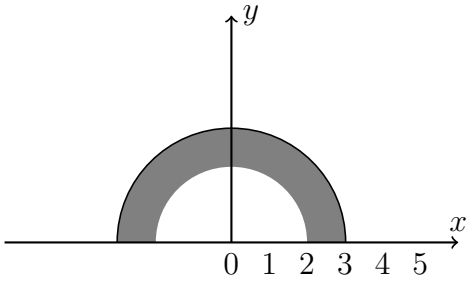


Abbildung D.2.: Halbkreisring

Die relevanten Parameter des Halbkreisrings werden hier in kartesischen Koordinaten angegeben. Trotzdem ist es einfacher, das Beispiel mithilfe von Polarkoordinaten zu lösen. Da in Polarkoordinaten nicht die x- und y-Koordinaten, sondern *r* und φ integriert werden sollen, müssen zunächst Gleichungen für *r* und φ gefunden werden.

Die Gleichung eines Kreises mit Radius r um den Ursprung ist $x^2 + y^2 = r^2$. Daraus folgt für die erste Bedingung: $2 \leq r \leq 3$. $y = 0$ ist für alle Winkel zwischen 0 und 180° bzw. π (Halbkreis!) erfüllt.

Nun muss, wie in Beispiel 1, eine sinnvolle Parametrisierung (Aufteilung auf mehrere Teil-Integrale) erfolgen. Diese ergibt sich hier aus den eben erstellten Bedingungen für *r* und φ. Der Flächeninhalt des Halbkreisringes lässt sich also wie folgt berechnen:

$$\int_R \mathrm{d}(x,y) = \int_{R'} r\mathrm{d}r\mathrm{d}\varphi = \int_{r=2}^{3}\int_{\varphi=0}^{\pi} r\mathrm{d}\varphi\mathrm{d}r = \tfrac{1}{2}\cdot r^2\big|_2^3 \cdot \varphi\big|_0^\pi = \left(\tfrac{9}{2}-\tfrac{4}{2}\right)\cdot(\pi-0) = \tfrac{5}{2}\pi$$

1.3. Kugelkoordinaten

Kugelkoordinaten werden auch als räumliche Polarkoordinaten bezeichnet. Hierbei wird ein Punkt durch seinen Abstand vom Ursprung *r* und zwei Winkel φ und θ angegeben.

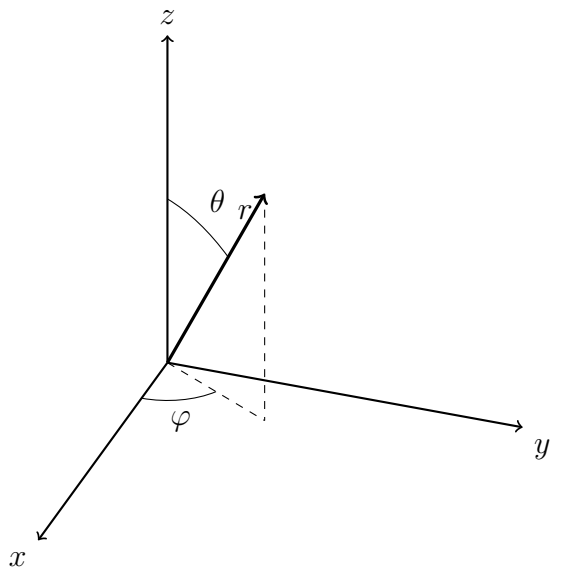


Abbildung D.3.: Kugelkoordinaten

Für den Zusammenhang zwischen kartesischen und Kugelkoordinaten gilt:

$$\begin{aligned} x &= r\cdot sin(\theta)\cdot cos(\varphi) & r &\geq 0 \\ y &= r\cdot sin(\theta)\cdot sin(\varphi) & 0 &\leq \varphi \leq 2\pi \\ z &= r\cdot cos(\theta) & 0 &\leq \theta \leq \pi \end{aligned}$$

Bei der Integration in Kugelkoordinaten wird stets auf das Volumenelement $r^2 sin(\theta)\mathrm{d}r\mathrm{d}\varphi\mathrm{d}\theta$ zurückgegriffen.

1.3.1. Beispiel 4

Zu berechnen ist das Volumen einer Kugel mit $r = 1$.

Die Parametrisierung ergibt sich aus der Aufgabenstellung und den vorgegebenen Wertebereichen der Winkel φ und θ.

$$\int_{r=0}^{1}\int_{\varphi=0}^{2\pi}\int_{\theta=0}^{\pi} r^2 sin(\theta)\mathrm{d}r\mathrm{d}\varphi\mathrm{d}\theta = \int_{r=0}^{1} r^2\mathrm{d}r \cdot \int_{\varphi=0}^{2\pi} 1\mathrm{d}\varphi \int_{\theta=0}^{\pi} sin(\theta)\mathrm{d}\theta$$

$$= \tfrac{1}{3}r^3\big|_0^1 \cdot \varphi\big|_0^{2\pi} \cdot (-cos(\theta))\big|_0^\pi = \tfrac{1}{3}\cdot 2\pi\cdot\big[1-(-1)\big] = \tfrac{1}{3}\cdot 2\pi\cdot 2 = \tfrac{4}{3}\pi$$

2. Anwendung: Raumwinkel

Der Raumwinkel Ω ist derjenige Winkel, der die Größe eines Raumes, welcher von einem Kegelmantel aufgespannt wird, beschreibt. Der Raumwinkel hat die Einheit Steradiant [sr]. Ein Raumwinkel von 1 sr beschreibt auf einer Kugel mit dem Radius $r = 1m$ eine Fläche von $A = 1m^2$.
Die ganze Kugel hat die Oberfläche $O = 4\pi r^2$, weshalb der zugehörige Raumwinkel $\Omega = 4\pi \approx 12,57sr$ ist.

Besondere Bedeutung hat die Rechnung mit Raumwinkeln beispielsweise in der Lichttechnik. Hier ist es unter Umständen sinnvoll, bereits an der verwendeten Einheit erkennen zu können, um welche lichttechnische Größe es sich handelt. Zum Beispiel wird der Lichtstrom inlm $=$ [cd · sr] und die Lichtstärke in [cd] angegeben. Es ist anhand der Einheiten ersichtlich, dass der Lichtstrom im Gegensatz zur Lichtstärke vom Raumwinkel abhängt.

Der Raumwinkel kann sehr einfach in einem Kugelkoordinatensystem definiert werden. Dies hat den Grund, dass bei der Verwendung von Kugelkoordinaten der Abstand vom Ursprung in diesem Fall konstant ist. Deshalb spricht man hier auch von *sphärischen Koordinaten*, bei denen lediglich zwei Winkel benötigt werden, um einen Punkt eindeutig anzugeben, da sich dieser, wie der Name schon sagt, auf einer definierten Sphäre, also Kugeloberfläche befindet.

Der Raumwinkel lässt sich über die Betrachtung eines Flächenelements auf der Kugeloberfläche in allgemeiner Form berechnen (polare Darstellung):

Zunächst werden die Winkel δ und γ definiert.
δ ist der Meridianwinkel und hat damit einen Wertebereich von $0° \leq \delta \leq 360°$ bzw. $0 \leq \delta \leq 2\pi$.
γ ist der Breitenwinkel mit einem Wertebereich von $0° \leq \gamma \leq 180°$ bzw. $0 \leq \gamma \leq \pi$.

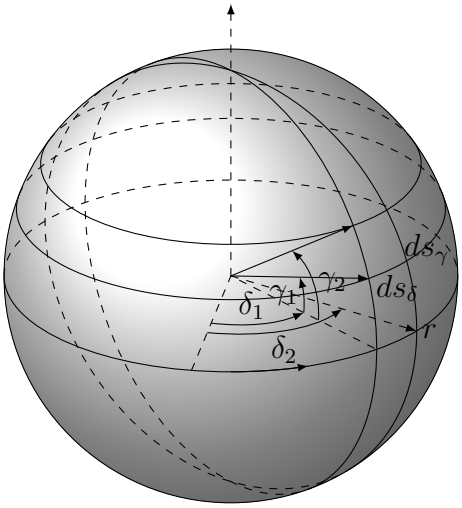


Abbildung D.4.: Raumwinkelberechnung

Für die Änderung des Raumwinkels Ω in Abhängigkeit von den Winkeln δ und γ gilt daher: $\mathrm{d}\Omega = \frac{\mathrm{d}A_K}{r^2}$.
Weiterhin gilt:
$\mathrm{d}\gamma = \frac{\mathrm{d}s_\gamma}{r} \Rightarrow \mathrm{d}s_\gamma = \mathrm{d}\gamma \cdot r$
$\mathrm{d}\delta = \frac{\mathrm{d}s_\delta}{r'} \Rightarrow \mathrm{d}s_\delta = \mathrm{d}\delta \cdot r'$

$sin(\gamma) = \frac{r'}{r} \Rightarrow r' = r \cdot sin(\gamma)$
$\mathrm{d}A_K = \mathrm{d}s_\gamma \cdot \mathrm{d}s_\delta$
$\mathrm{d}\Omega = \frac{\mathrm{d}A_K}{r^2} = \frac{\mathrm{d}s_\gamma \cdot \mathrm{d}s_\delta}{r^2} = \frac{\mathrm{d}\gamma \cdot r \cdot \mathrm{d}\delta \cdot r'}{r^2} = \frac{\mathrm{d}\gamma \cdot r \cdot \mathrm{d}\delta \cdot r \cdot sin(\gamma)}{r} = \mathrm{d}\gamma \cdot d\delta \cdot sin(\gamma)$
Daraus folgt: $\Omega = \int_{\delta_1}^{\delta_2} \int_{\gamma_1}^{\gamma_2} sin(\gamma)\mathrm{d}\gamma\mathrm{d}\delta$

2.0.2. Beispiel 5

Zu berechnen ist der Raumwinkel des Drittels einer Kugel.

$\Omega = \int_{\delta=0}^{2\pi} \int_{\gamma=0}^{\frac{\pi}{3}} sin(\gamma)\mathrm{d}\gamma\mathrm{d}\delta = \int_{\delta=0}^{2\pi} \left(-cos(\gamma)\big|_0^{\frac{\pi}{3}}\right)\mathrm{d}\delta = \int_{\delta=0}^{2\pi} -cos(\frac{\pi}{3}) + 1\mathrm{d}\delta = \int_{\delta=0}^{2\pi} (-\frac{1}{2}) + 1\mathrm{d}\delta =$
$\delta\big|_0^{2\pi} \cdot (\frac{1}{2}) = 2\pi \cdot \frac{1}{2} = \pi$

3. Integralrechnung mit MATLAB

Auch Integrale können mithilfe von MATLAB berechnet werden. Für die oben berechneten Beispiele werden hier exemplarisch die Musterlösungen bezüglich der Eingaben in das Command Window dargestellt. Hier sind Integrale mit angegebenen Grenzen zu berechnen. Diese können in MATLAB direkt innerhalb des Befehls zur Integration, getrennt von der Integrationsvariablen durch Komma, angegeben werden.

Beispiel 1

```
>> syms x y
>> f=x^2*cos(y);
>> int(int(f,y,0,pi/2),x,-1,1)
```

Beispiel 3

```
>> syms p r
>> f = r;
>> int(int(f,p,0,pi),r,2,3)
```

Beispiel 4

```
>> syms r p t
>> f=r^2*sin(t);
>> int(int(int(f,t,0,pi),p,0,2*pi),r,0,1)
```

Beispiel 5

```
>> syms g d
>> f=sin(g);
>> int(int(f,g,0,pi/3),d,0,2*pi)
```

3.1. Aufgaben

Aufgabe 1

Berechnen Sie den Raumwinkel einer Vollkugel.

Aufgabe 2

Berechnen Sie den Raumwinkel einer Halbkugel.

Aufgabe 3

Berechnen Sie den Raumwinkel eines Kegels mit dem Öffnungswinkel α.

3.2. Lösungen

Lösung 1

$$\Omega = \int_{\delta=0}^{2\pi} \int_{\gamma=0}^{\pi} sin(\gamma) \mathrm{d}\gamma \mathrm{d}\delta = \int_{\delta=0}^{2\pi} \Big(-cos(\gamma)\big|_0^{\pi} \Big) \mathrm{d}\delta = \int_{\delta=0}^{2\pi} 2 \mathrm{d}\delta = \delta\big|_0^{2\pi} \cdot 2 = 2\pi \cdot 2 = 4\pi$$

Lösung mit MATLAB:

```
>> syms g d
>> f=sin(g);
>> int(int(f,g,0,pi),d,0,2*pi)
```

Lösung 2

$$\Omega = \int_{\delta=0}^{2\pi} \int_{\gamma=0}^{\frac{\pi}{2}} sin(\gamma)\mathrm{d}\gamma\mathrm{d}\delta = \int_{\delta=0}^{2\pi} \left(-cos(\gamma)\Big|_0^{\frac{\pi}{2}}\right)\mathrm{d}\delta = \int_{\delta=0}^{2\pi} 1\mathrm{d}\delta = \delta\Big|_0^{2\pi} \cdot 1 = 2\pi \cdot 1 = 2\pi$$

Lösung mit MATLAB:

```
>> syms g d
>> f=sin(g);
>> int(int(f,g,0,pi/2),d,0,2*pi)
```

Lösung 3

$$\Omega = \int_{\delta=0}^{2\pi} \int_{\gamma=0}^{\frac{\alpha}{2}} sin(\gamma)\mathrm{d}\gamma\mathrm{d}\delta = \int_{\delta=0}^{2\pi} \left(-cos(\gamma)\Big|_0^{\frac{\alpha}{2}}\right)\mathrm{d}\delta = \int_{\delta=0}^{2\pi} (-cos(\tfrac{\alpha}{2}) + 1)\mathrm{d}\delta$$

$$= (-cos(\tfrac{\alpha}{2}) + 1)\delta\Big|_0^{2\pi} = 2\pi \cdot (-cos(\tfrac{\alpha}{2}) + 1) = 2\pi \cdot (1 - cos(\tfrac{\alpha}{2}))$$

Lösung mit MATLAB:

```
>> syms g d a
>> f=sin(g);
>> int(int(f,g,0,a/2),d,0,2*pi)
```

E. Skript zum Praktikum MATLAB - Teil 1: Direkte Eingabe

Das Skript, welches den ersten Teil des Praktikums MATLAB im Rahmen der Lehrveranstaltung Mathematik begleiten soll, befindet sich, inklusive der von den Studierenden im Praktikum zu bearbeitenden Aufgaben, auf den folgenden Seiten.

MATLAB
Teil 1

1. Allgemeines

MATLAB ist ein nummerisches Berechnungs- und Simulationswerkzeug. Das bedeutet, dass es sich hierbei nicht um ein Computeralgebra-Programm handelt, mit dem symbolische Operationen durchführbar sind, es also nicht möglich ist, mathematische Formeln so berechnen zu können, wie ein Mensch dies normalerweise tut, sondern alle Berechnungen prinzipiell rein nummerisch erfolgen. Auf die Computeralgebra-Funktionalität kann allerdings innerhalb der MATLAB-Umgebung über die „Symbolics"-Toolbox zugegriffen werden. Die grundsätzliche Struktur, auf der alle MATLAB-Operationen beruhen, ist jedoch das numerische Feld, also eine Matrix. Daher auch der Name MATLAB, welcher eine Abkürzung für MATrix LABoratory ist.

Typische Anwendungen von MATLAB sind:

- Mathematische Berechnungen
- Entwicklung von Algorithmen
- Datenerfassung und -bearbeitung
- Datenanalyse, -auswertung und -visualisierung
- Wissenschaftliche und technische grafische Darstellungen
- Entwicklung von Anwendungen, inklusive der Gestaltung von grafischen Benutzeroberflächen

Besonderen Wert wird während des Praktikums auf mathematische Berechnungen und grafische Darstellungen gelegt.

2. Grundlagen der Arbeit mit MATLAB

2.1. Der MATLAB-Desktop

Je nachdem, welches Fenster gerade aktiv ist, ändert sich die Taskleiste über den Fenstern, wobei die Kategorie *Desktop* immer erhalten bleibt (siehe Abb. E.1). Hier kann unter *Desktop Layout* mit Anwählen von *Default* jederzeit der Ausgangsbildschirm mit den vier Fenstern in der nachfolgend beschriebenen Anordnung wieder hergestellt werden.

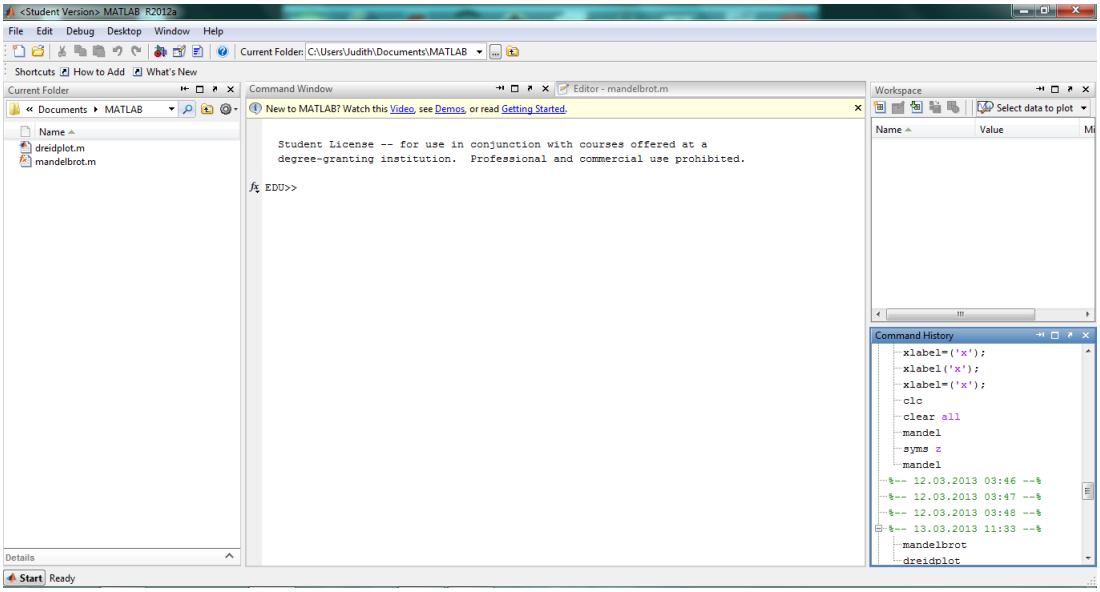


Abbildung E.1.: MATLAB-Desktop nach dem Start (Studenten-Version R 2012a)

2.2. Die verschiedenen Fenster

Die Fenster, die nach dem Start von MATLAB zu sehen sind, werden im Folgenden näher erläutert.

2.2.1. Command Window

Das Command Window ist das Fenster, in dem hauptsächlich gearbeitet wird (siehe Abb. E.2). Hier werden Befehle eingegeben, Funktionen gestartet, Ergebnisse der Berechnungen wiedergegeben und Fehlermeldungen angezeigt.

Eingaben sind immer hinter dem Zeichen >> zu machen. Fehlt dieses Zeichen, so ist entweder

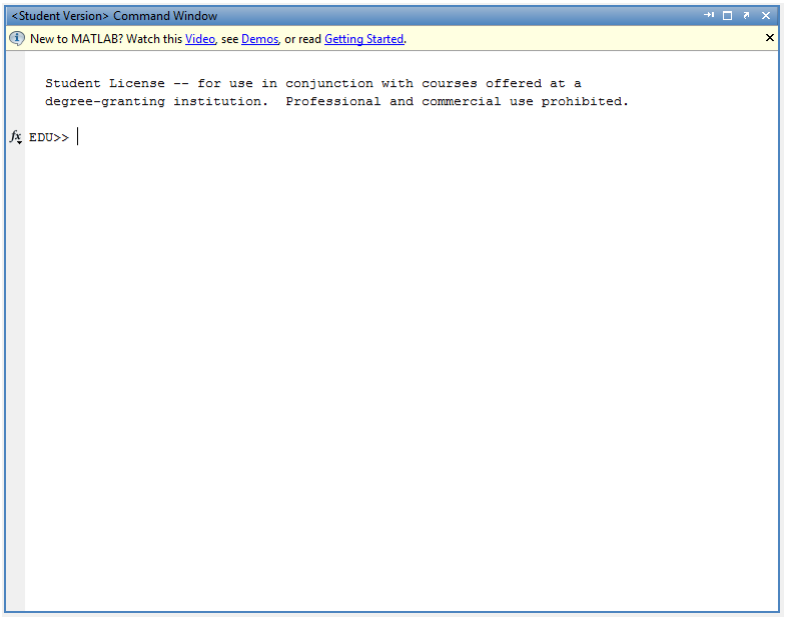


Abbildung E.2.: MATLAB-Command Window nach dem Start (Studenten-Version R 2012a)

eine zuvor begonnene Eingabe nicht fertiggestellt oder MATLAB befindet sich noch im Prozess einer Berechnung. Soll hinter diesem Zeichen keine Eingabe erfolgen, sondern lediglich ein Kommentar eingefügt werden, so ist dieser durch das Prozentzeichen % einzuleiten und durch die Eingabetaste abzuschließen. Kommentare werden von MATLAB in grüner Schrift hervorgehoben.

Es besteht die Möglichkeit, mehrere MATLAB-Befehle zusammenzufassen. Dazu werden *Shortcuts* erstellt, die es ermöglichen, die zusammengefassten Befehle auf einmal auszuführen. Zur Definition eines Shortcuts wird der Shortcut-Editor geöffnet. Dieser ist über den Start-Button (ganz unten links) zu erreichen. Hier wird im Menü `Shortcuts - New Shortcut` aufgerufen. Die erstellten Shortcuts können wahlweise im Start-Button-Menü verankert oder oder als Icon in die Shortcut-Leiste (unterhalb der Symbolleiste) integriert werden.

2.2.2. Current Folder bzw. Current Directory

Links befindet sich eine Spalte des aktuellen Verzeichnisses, unter Windows im Normalfall das Verzeichnis `MATLAB` unter `Eigene Dateien` (siehe Abb. E.3). Hier ist zu erkennen, ob bereits MATLAB-Dateien erzeugt wurden (Endungen `.m` oder `.mat`). Im Fenster darunter können bei Bedarf Details angezeigt werden.

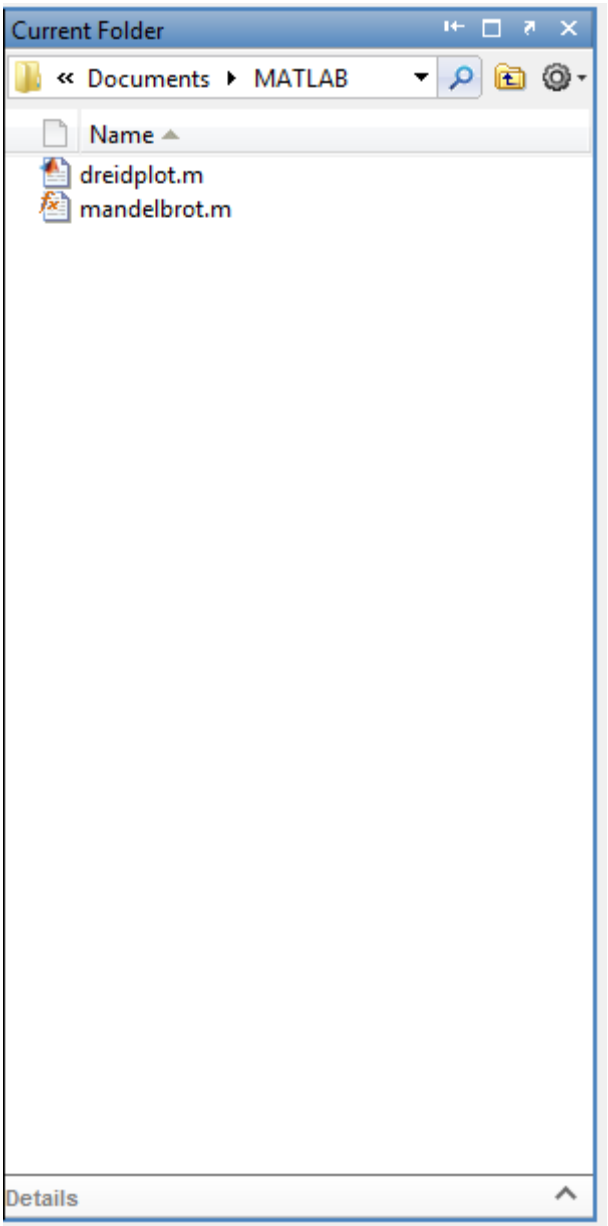


Abbildung E.3.: Current Folder bzw. Current Directory (Studenten-Version R 2012a)

2.2.3. Workspace

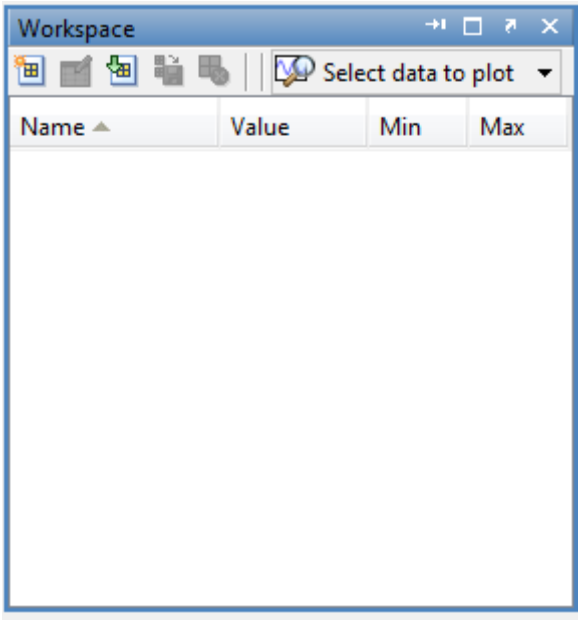


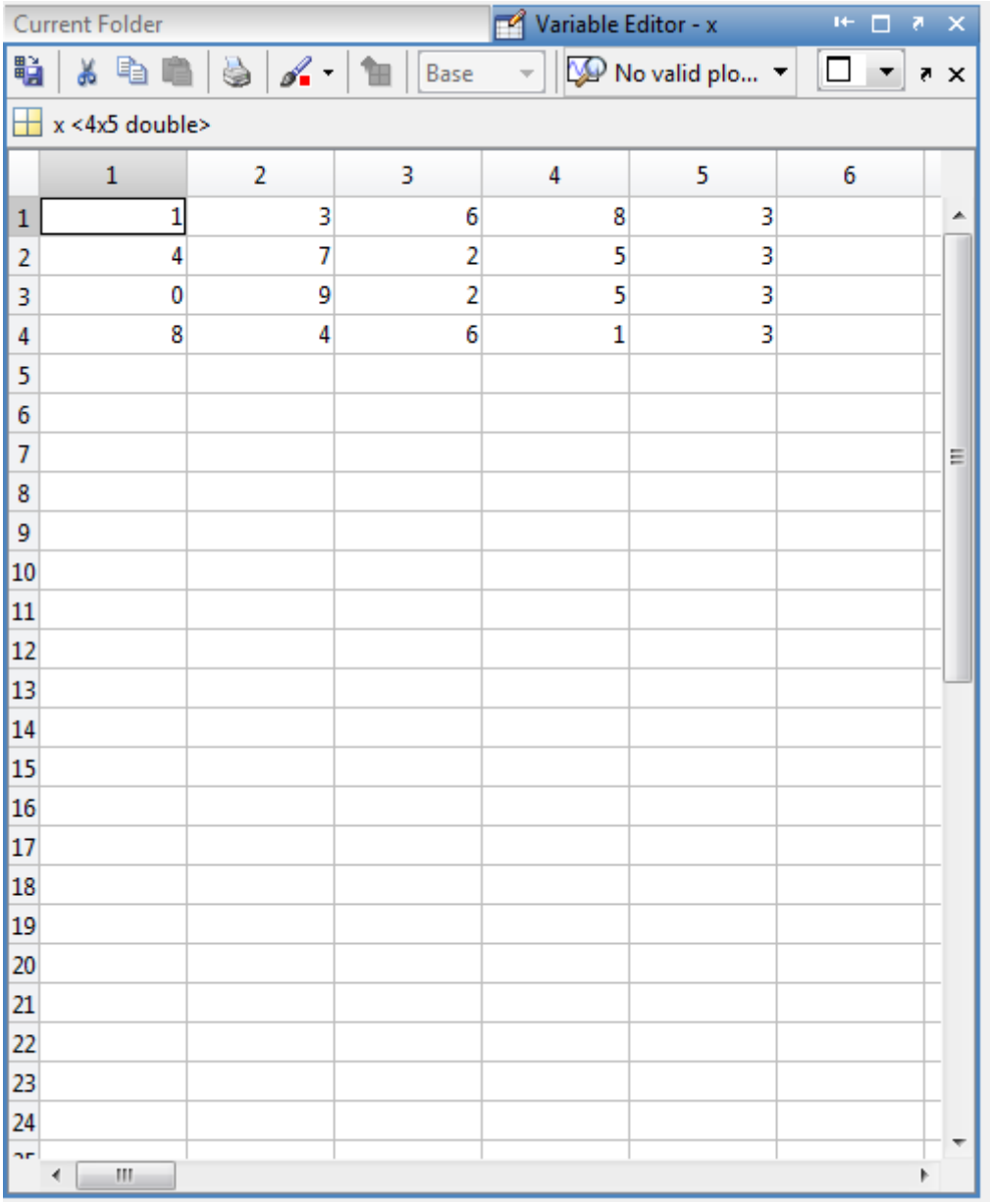
Abbildung E.4.: MATLAB-Workspace mit den momentan verfügbaren Variablen (Studenten-Version R 2012a)

Rechts oben befindet sich der Workspace, also der aktuelle Arbeitsbereich von MATLAB (siehe Abb. E.4). Hier sind die aktuell im Arbeitsspeicher abgelegten Variablen und Parameter, nach großen und kleinen Buchstaben sowie alphabetisch, aufgelistet. Durch Drag & Drop können Variablen vom Workspace in das Command Window gezogen und dort in Befehle eingebunden werden.

Durch Doppelklicken mit der linken Maustaste auf eine der Variablen wird diese im Variable Editor (siehe Abb. E.5) geöffnet, was besonders bei der Eingabe von Matrizen hilfreich ist. Der Variable Editor ähnelt einem Excel-Sheet, in dem die Variablen auch bearbeitet werden können.

2.2.4. Command History

Im Fenster rechts unten wird die Command History, also die Chronik der eingegebenen Befehle mit entsprechendem Eingabedatum, angezeigt (siehe Abb. E.6). Durch Doppelklick mit der linken Maustaste wird der gewählte Befehl im Command Window erneut ausgeführt. Soll die Command History gelöscht werden, so geschieht dies am einfachsten durch einen rechten Mausklick in das Fenster Command History. Hier öffnet sich nun ein Auswahlmenü, in dem



	1	2	3	4	5	6
1	1	3	6	8	3	
2	4	7	2	5	3	
3	0	9	2	5	3	
4	8	4	6	1	3	
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						

Abbildung E.5.: MATLAB-Variable Editor mit dem Inhalt der Matrix x, bestehend aus 5 Spalten und 4 Zeilen (Studenten-Version R 2012a)

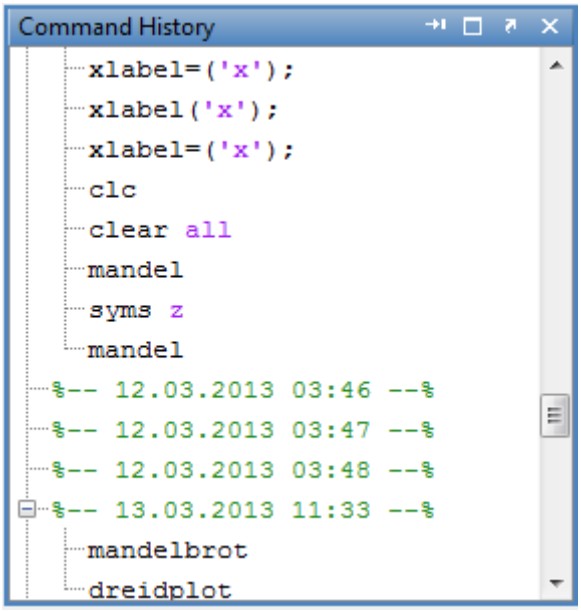


Abbildung E.6.: Command History mit Wiedergabe der zuletzt eingegebenen MATLAB-Befehle (Studenten-Version R 2012a)

mit „Clear Command History“ alle Befehle gelöscht werden.

2.3. Die MATLAB-Hilfe

Die MATLAB-Hilfe lässt sich auf verschiedene Arten aufrufen und nutzen. Zum Einen besteht die Möglichkeit, die „Product Help“ durch Mausklick auf *Help* auf der Taskleiste und anschließendem Klick auf *Product Help* oder durch Klicken auf das *?*-Symbol neben Current Folder zu öffnen. Dies ist die ausführlichste Hilfe zu MATLAB, die ein Inhaltsverzeichnis besitzt (*Contents*), in dem die einzelnen Hilfethemen nach Kategorien und Toolboxen sortiert sind. *Search Results* ruft alle Suchergebnisse ab, die im oberen Suchfeld eingegeben werden. Der *Index* listet alle MATLAB-Befehle und andere wichtige Begriffe, zu denen es Befehle und Funktionen gibt, alphabetisch auf. Schließlich werden unter *Demos* einige Demo-Videos zur Verfügung gestellt, welche jedoch zum Teil nur über das Internet abrufbar sind.

Weiterhin besteht die Möglichkeit der Hilfe über den „Function Browser“, welcher sich durch Klicken auf das *fx*-Icon links neben dem >>-Eingabezeichen oder durch die Tastenkombination <Umschalt/Shift> + <F1> öffnen lässt. Hier werden als Kategorien MATLAB und die verschiedenen installierten Toolboxen vorgegeben, durch Auswahl der jeweiligen Kategorie und der entsprechenden Unterkategorien kann man eine gesuchte Funktion finden, die dann durch Doppelklick oder Drag & Drop in das Command Window übernommen werden kann.

Schließlich kann die MATLAB-Hilfe mit dem Befehl `help` im Command Window aufgerufen werden. Die Eingabe dieses Befehls bewirkt einen Aufruf aller Haupt- und Unterkategorien, zu denen es MATLAB-Befehle gibt. Die Ergebnisliste wird direkt im Command Window angezeigt. Nun kann man sich die MATLAB-Hilfe zu einem gesuchten Befehl durch den Aufruf von `help <Befehl>` ausgeben lassen.

3. Berechnungen mit MATLAB

Wie bereits erwähnt, ist die grundlegende Datenstruktur von MATLAB die Matrix. Alle anderen Datenstrukturen wie Vektoren oder Skalare sind Spezialfälle von Matrizen. Daraus ergeben sich einige Besonderheiten bei Berechnungen mit Variablen, auf die im Folgenden ebenso eingegangen wird, wie auf Besonderheiten mathematischer Funktionen und das Erstellen von Grafiken. Auf die einzelnen spezifischen Befehle wird anschließend im Kapitel 4 Übersicht wichtiger Befehle detailliert eingegangen.

3.1. Variablen

Eine MATLAB-Variable ist ein Objekt eines bestimmten Datentyps. Wie bereits erwähnt, ist der grundlegende Datentyp hier die Matrix. Das bedeutet, dass jede MATLAB-Variable eine Matrix ist, die aus reellen oder komplexen Zahlen bestehen kann.

Die Matrix wird im Command Window definiert und mit einem frei wählbaren Variablennamen versehen. Dabei ist die folgende Syntax anzuwenden:

```
>> x = 2
```

Dadurch wird die Zahl 2 (eine einfache Zahl ist in MATLAB eine 1×1-Matrix!) der Variablen `x` zugewiesen und kann im Folgenden unter diesem Variablennamen angesprochen werden. MATLAB bestätigt diese Eingabe durch eine entsprechende Ausgabe. Bei syntaktischen Fehlern erfolgt eine Fehlermeldung. Erscheint das von MATLAB verwendete Zahlenformat ungeeignet, so kann dieses über den Menübefehl *File – Preferences...* in der Karteikarte *Command-Window – Numeric Format* geändert werden.

Komplexe Zahlen können mithilfe der Symbole `i` und `j` in der oben angegebenen Form definiert werden. Da `i` und `j` dafür vorab reserviert sind, sollten diese Buchstaben nicht für andere Variablen verwendet werden.

Die Eingabe von Matrizen geschieht wie folgt: Die Einträge in den Zeilen der Matrix sind jeweils durch ein Leerzeichen zu trennen, die verschiedenen Spalten sind durch Semikola zu

trennen. Der Matrix wird hier der Name `Matrix` zugewiesen.

```
>> Matrix = [a b c; d e f]
```

Die Ausgabe im Command Window zur Bestätigung der Eingabe gibt die Matrix in korrekter Anordnung zurück:

```
Matrix =
      a b c
      d e f
```

Sämtliche definierten Variablen werden im Workspace gespeichert und können über die Kommandos `who` (für die Namen der Variablen) und `whos` (für Namen und Zusatzinformationen) im Command Window zur Information abgerufen werden. Durch Doppelklick auf eine Variable im Workspace öffnet sich der Variable Editor, hier können die definierten Matrizen weiter bearbeitet werden. Benötigt man eine Variable nicht mehr, so kann man die im Command Window mit dem Befehl `clear [Variablenname]` löschen.

3.2. Arithmetische Operationen

Bei den arithmetischen Operationen ($+$, - etc.) ist zu beachten, dass die grundlegende Struktur von MATLAB die Matrix ist, weshalb diese Operationen zunächst als Matrixoperationen zu verstehen sind. Deshalb werden die Rechenregeln der Matrizenalgebra mit allen sich ergebenden Konsequenzen unterstellt. Beispielsweise ist deshalb das Produkt zweier Variablen A und B nicht definiert, falls die zugrunde liegende Matrizenmultiplikation nicht definiert ist, d.h. falls Spaltenzahl von A und Zeilenzahl von B nicht gleich sind. Eine Ausnahme von der Regel besteht dann, wenn mindestens eine der Variablen ein Skalar, also eine 1×1-Matrix ist.

Falls keine der definierten Variablen ein Skalar ist und die vorgesehene arithmetische Operation trotzdem komponentenweise ausgeführt werden soll, so ist der entsprechenden Operation in der MATLAB-Syntax ein Punkt (.) voranzustellen. Ein * alleine löst demnach eine Matrizenmultiplikation aus, ein .* ist als komponentenweise Multiplikation zu verstehen.

3.3. Mathematische Funktionen

MATLAB verfügt, inklusive der verschiedenen Toolboxes, über eine Vielzahl verschiedener vorgefertigter mathematischer Funktionen wie beispielsweise Sinus, cosinus, die Exponentialfunktion, der Logarithmus etc. Das scheinbar auftretende Problem, dass die Datenstruktur von MATLAB auf Matrizen beruht und diese Funktionen für Matrizen nicht definiert sind,

wird hier dadurch gelöst, dass die jeweiligen Funktionen komponentenweise auf die eingegebenen Matrizen wirken. Beispielsweise ist der Cosinus eines angegebenen Vektors wiederum ein Vektor:
$\vec{x} = (cos(0), cos(1), cos(2), \ldots, cos(n))$
Dies ist sinnvoll, da ein solches Vorgehen dazu führt, dass zur Darstellung mathematischer Funktionen keine Programmierschleifen notwendig werden.

3.4. Grafikfunktionen

MATLAB kann jegliche Berechnungsergebnisse grafisch darstellen. Dabei sind sowohl zweials auch dreidimensionale Grafiken möglich. Dieses Skript beschränkt sich dabei auf zweidimensionale Grafiken (xy-Plots). Zu einem vollständigen Überblick über alle Grafikfunktionen führen, für Interessierte, die Eingabe von `help graph2d`, `help graph3d` und `help graphics` im Command Window oder die entsprechenden Einträge in der MATLAB Product Help. Prinzipiell werden zwei Vektoren benötigt, um einen Graph zu plotten. Dabei stehen der erste Vektor für die x-Werte des Graphen und der zweite Vektor für die zugehörigen y-Werte. Die Vektoren müssen dafür die gleiche Länge haben. Es können auch mehrere Funktionen gleichzeitig geplottet werden, entweder indem man die x,y-Paare nacheinander in die Parameterliste schreibt oder indem man die y-Vektoren zu einer entsprechenden Matrix zusammenfasst. Letzeres funktioniert nur dann, wenn für alle Plots der gleiche x-Vektor verwendet werden soll.

Soll beispielsweise die Sinus-Funktion geplottet werden, so ist über den x-Vektor der Definitionsbereich $(1 \leq x \leq 10)$ inklusive Schrittweite (1) anzugeben, Der y-Vektor ergibt sich aus der Funktion. Die einzelnen Befehle sind dabei durch Semikola zu trennen. Die Eingabe in das Command Window ist dann:

```
>> x = (0:1:10);
>> y = sin(x);
>> plot(x,y)
```

Zudem können zusätzliche Parameter die Linienart und Farbe des Graphen verändern.

3.5. Symbolische Rechnungen

Obwohl MATLAB prinzipiell ein numerisches Werkzeug ist, sind mit der integrierten Symbolic Math Toolbox auch symbolische Rechnungen möglich. Einen kompletten Überblick über die Möglichkeiten, die mit dieser Toolbox zur Verfügung stehen, bekommt man durch die Einga-

be von `help symbolic` in das Command Window oder durch Aufsuchen der entsprechenden Einträge in der MATLAB Product Help. Unter anderen ist es mit dieser Toolbox möglich, Differenziale, Integrale, Grenzwerte, Taylor-Reihen und vieles mehr zu berechnen.

Um eine symbolische Berechnung durchführen zu können, muss man MATLAB mitteilen, dass es sich bei den für diese Berechnungen definierten Variablen um Symbole und nicht, wie sonst, um numerische Variablen handelt. Dies geschieht durch die Anweisung `syms`, also werden durch

```
>> syms x y z
```

Die Symbole `x`, `y` und `z` angelegt. Diese sind nun auch im Workspace zu sehen. Dazu kann eine entsprechende Funktion angelegt werden:

```
>> f = x + y + z
```

Sollen die Ergebnisse nach Anwendung eines bestimmten Befehls übersichtlicher dargestellt werden, so ist das durch den zusätzlichen Befehl `pretty(Name der Funktion/des Befehls)`, von der Eingabe der Funktion getrennt durch ein Semikolon, zu bewerkstelligen (im vorliegenden Beispiel macht das selbstverständlich wenig Sinn, bei komplexeren Funktionen kann dieser Befehl hingegen von Nutzen sein):

```
>> f = x + y + z
>> pretty(f)
```

Auch können mittels der Symbolic Math Toolbox zwischenzeitliche Berechnungen symbolischer Ausdrücke in prinzipiell numerischen Rechnungen durchgeführt werden.

4. Übersicht wichtiger Befehle

Die folgenden Tabellen geben eine Übersicht über Befehle, die zur Lösung relevanter Aufgaben bezügliche der Lehrveranstaltung Mathematik notwendig sein können. Diese Übersicht stellt keinen Anspruch auf Vollständigkeit.

4.1. Wichtige Kommandos

Tabelle E.1.: Übersicht über nützliche Kommandos

Eingabe in das Command Window	**Verarbeitung**
`>> cd`	gibt das aktuelle MATLAB-Verzeichnis an

`>> clc`	das Command Window wird gelöscht, alle vorherigen Ein- und Ausgaben sind nicht mehr sichtbar
`>> clear x`	löscht die Variable x aus dem Arbeitsspeicher
`>> clear all`	löscht alle Variablen aus dem Arbeitsspeicher
`>> delete datei`	löscht die Datei „datei“ aus dem aktuellen Verzeichnis
`>> diary datei`	Alle folgenden Ein- und Ausgaben im Command Window werden in die Datei „datei“ geschrieben. Der Dateiname kann, muss aber nicht, mit einer Endung wie .txt, .doc etc. versehen werden.
`>> diary off`	Beendet das Schreiben und schließt die aktuelle Datei. Durch Entfernen der Ergebnisse und Anfügen bzw. Austausch der Extension .txt bzw. .doc durch .m entsteht ein lauffähiges M-file.
`>> diary on`	Öffnet die zuletzt geschlossene Datei wieder und schreibt weiter in sie hinein.
`>> dir/ls`	Listet den Inhalt des aktuellen Verzeichnisses auf.
`>> disp('Text')`	Gibt den 'Text' aus.
`>> echo on` bzw. `off`	Gibt die Ausgabe im Command Window frei bzw. verhindert sie.
`>> format...`	Legt das ausgegebene Zahlenformat fest.
`>> format compact`	Unterdrückt die Leerzeile bei der Ausgabe.
`>> fprintf('Text%f\n', Wert)`	Gibt 'Text' und einen oder mehrere numerische Werte aus.
`>> help „function\`	Gibt den zu der „function“ gehörenden Text aus, z.B. `help format`.
`>> load Daten2_1.mat`	Lädt die in Daten2_1.mat gespeicherten Variablen in den Arbeitsspeicher.
`>> save Daten2_1.mat`	Speichert alle im Arbeitsspeicher hinterlegten Variablen in den Daten2_1.mat.
`>> save Daten2_1.mat a b`	Speichert die Variablen `a` und `b` des Arbeitsspeichers in Daten2_1.mat.
`>> type name`	Zeigt den Inhalt der Datei name.m an.
`>> what`	Gibt die M-, MAT- und MEX-Files des Verzeichnisses an.
`>> which name`	Zeigt das Verzeichnis an, in dem sich die Datei name.m befindet.
`>> who`	Gibt eine Liste mit den aktuellen Variablen im Arbeitsspeicher aus.
`>> whos`	Gibt eine erweiterte Variablenliste aus.
`>> name;`	Das Semikolon verhindert die Ausgabe des Inhalts von name.
`>> %`	Leitet einen Kommentar ein.
`>> ...`	Am Ende einer Befehlszeile, bedeutet Fortsetzung des Befehls in der nächsten Zeile.

4.2. Arithmetische Operationen

Tabelle E.2.: Übersicht über arithmetische Operationen

Operation	**Command Window**	
	Eingabe	**Ausgabe**
Zuweisung	`>> a = 4; b = 5`	
Addition	`>> s = a + b`	`s =` `9`
Subtraktion	`>> d = a - b`	`d =` `-1`
Multiplikation	`>> m = a * b`	`m =` `20`
Division von rechts	`>> r = a / b`	`r =` `0.8000`
Division von links	`>> r = a \ b`	`l =` `1.2500`
Potenz	`>> p = a^b`	`p =` `1024`
Quadratwurzel	`>> w2 = sqrt(p)`	`w2 =` `32`
n-te Wurzel	`>> wn = nthroot(s,a)`	`wn =` `1.7321`

4.3. Spezielle Werte

Tabelle E.3.: Übersicht über spezielle Werte

Beschreibung	**Command Window**	
	Eingabe	**Ausgabe**
ans: Fehlt die Ergebnisvariable, so wird das Resultat „ans“ zugewiesen	`>> a + b`	`ans =` `9`
i,j $\hat{=}$ imaginäre Einheit	`>> i`	`ans =` `0 + 1.0000i`

Inf $\hat{=} \infty$	`>> d0 = a/0`	`Warning: Divide by zero.`
		`d0 =`
		`Inf`

NaN: Not-a-Number	`>> nn = Inf/Inf`	`nn =` `NaN`
pi $\hat{=} \pi$	`>> format long, fl = pi`	`fl =` `3.14159265358979`

4.4. Häufig benötigte Funktionen

Tabelle E.4.: Übersicht über häufig benötigte Funktionen

Funktion	**Command Window**	
	Eingabe	**Ausgabe**
exp(x) $\hat{=}$ e^x: Exponentialfunktion von x	`>> e1 = exp(1)`	`e1 =` `2.7183`
log(x) $\hat{=}$ ln(x): natürlicher Logarithmus von x	`>> lo = log(e1)`	`lo =` `1`
log10(x) $\hat{=}$ lg(x): dekadischer Logarithmus von x	`>> lo1 = log10(e1)`	`lo1 =` `0.4343`
round(x): rundet zur nächsten ganzen Zahl	`>> r = round(e1)`	`r =` `3`
factorial(n) $\hat{=}$ n!: Fakultät	`>> factorial(4)`	`ans =` `24`

4.5. Komplexe Zahlen

Tabelle E.5.: Übersicht über Funktionen der komplexen Zahlen

Funktion	**Command Window**	
	Eingabe	**Ausgabe**
z: komplexe Zahl	`>> Z = a + b*i`	`z =` `4.0000 + 5.0000i`
abs(z) $\hat{=}$ |z|: absoluter Wert von z	`>> Z = abs(z)`	`Z =` `6.4031`
angle(z) $\hat{=}$ arg(z): Winkel im Bogenmaß	`>> phi = angle(z)`	`phi =` `0.8961`
Winkel in Grad	`>> phi_g =` `angle(z)*180/pi`	`phi_g =` `51.3402`
conj(z) $\hat{=}$ $\bar{z}$: konjugiert komplexe Zahl zu z	`>> zc = conj(z)`	`zc =` `4.0000 - 5.0000i`
imag(z) $\hat{=}$ Im(z): imaginärer Teil der komplexen Zahl z	`>> zi = imag(z)`	`zi =` `5`
real(z) $\hat{=}$ Re(z): realer Teil der komplexen Zahl z	`>> zr = real(z)`	`zr =` `4`

4.6. Matrizen

Hier bezeichnet m jeweils die Anzahl der Zeilen, n bezeichnet die Anzahl der Spalten.

Tabelle E.6.: Übersicht zur Matrizenrechnung

Funktion	**Command Window**	
	Eingabe	**Ausgabe**

$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 0 & 11 & 12 \end{pmatrix}$	`>> A = [1 2 3;4 5`
	`6;7 8 9;0 11 12]`
		`A =`
		`1 2 3`
		`4 5 6`
		`7 8 9`
		`0 11 12`

Addition
„+“
Bedingung:
mA = mB, nA = nB
	`>> B = [1 -1 1;2 5 -1;4 0 2;-1 2 1]`	`ans =`
		`2 1 4`
		`6 10 5`
		`11 8 11`
		`-1 13 13`

Subtraktion
„-“
Bedingung:
mA = mB, nA = nB
	`>> A -B`	`ans =`
		`0 3 2`
		`2 0 7`
		`3 8 7`
		`1 9 11`

Multiplikation
„*“
Bedingung: nA = mB
→ Typ(mA,nB)
	`>> C = [1 3 0;2 0 1;2 1 1];`
	`>> A * C`
		`ans =`
		`11 6 5`
		`26 18 11`
		`41 30 17`
		`46 12 23`

Division
„/“
Bedingung:
mA = mB, nA = nB
	`>> A / B`	`ans =`
		`0 -0.1739 0.6957 1.4348`
		`0 -0.0435 1.6739 2.6087`
		`0 0.0870 2.6522 3.7826`
		`0 -0.6522 2.1087 7.1304`

Multiplikation mit einem Skalar. Jedes Element der Matrix wird mit einem Skalar multipliziert.	`>> 3 * A`	`ans =`
		`3 6 9`
		`12 15 18`
		`21 24 27`
		`0 33 36`

elementeweise Multiplikation	`>> A.*B`	`ans =`
		`1 -2 3`
		`8 25 -6`
		`28 0 18`
		`0 22 12`

elementeweise Division	`>> A./B`	`Warning: Divide by zero.`
		`ans =`
		`1.0000 -2.0000 3.0000`
		`2.0000 1.0000 -6.0000`
		`1.7500 Inf 4.5000`
		`0 5.5000 12.0000`

elementeweises Potenzieren	`>> A.^2`	`ans =`
		`1 4 9`
		`16 25 36`
		`49 64 81`
		`0 121 144`

Transponierte einer Matrix	`>> At = A'`	`At =`
		`1 4 7 0`
		`2 5 8 11`
		`3 6 9 12`

Erstellen einer Nullmatrix	`>> N = zeros(1,3)`	`N =` `0 0 0`

4.7. Grafische Darstellungen

Tabelle E.7.: Übersicht über Befehle für Grafiken

Funktion	**Beschreibung**
`>> annotation('Typ','Position')`	Fügt anmerkungen wie Pfeile 'arrow', Text-Pfeile 'textarrow' usw. in eine Figur ein. 'Position' entspricht den normalisierten Koordinaten, um Positionen innerhalb der Abbildung zu spezifizieren. In den normalisierten Koordinaten sind immer der Punkt (0,0) die linke untere und der Punkt (1,1) die rechte obere Ecke des Abbildungsfensters, unabhängig von der Größe der Abbildung.
`>> axis([`$x_{min}x_{max}y_{min}y_{max}$`])`	Legt die Bereiche der zwei Achsen bei einer 2-D Grafik fest.

`>> axis equal`
`>> axis square`
`>> axis tight`
⋮
	Erzeugt gleich lange Achseinheiten.
	Stellt die Achsenboxflächen quadratisch dar.
	Die Achsen werden durch Daten der Grafik begrenzt.
	Weitere Eigenschaften, siehe `help axis`

`>> ezplot('f(x)',[`$x_{min}x_{max}$`])` `>> ezplot('f(x)',[`$x_{min}x_{max}y_{min}y_{max}$`])`	Stellt explizite Funktionen als 2-D Grafiken im vorgegebenen Bereich dar. Ohne Angabe der Grenzen wird im Bereich [-2π bis +2π] dargestellt. Die Funktion erscheint im 'titel'.
`>> figure(i)`	Erzeugt die i-te Figur.
`>> linspace(`x_a`,`x_e`,n)`	Erzeugt einen Vektor mit n Elementen zwischen x_a = Anfangs- und x_e = Endpunkt. Wird n nicht angegeben, werden 100 Punkte berechnet.

`>> logspace(a,e,n)`	Erzeugt einen Vektor mit logarithmischen Elementen auf der Basis 10. Der Vektor beginnt mit 10^a und endet bei 10^e. Die Größe n legt die Anzahl der Elemente fest. Bei fehlendem n werden 50 Werte berechnet.
`>> plot(x,y,S)`	Erzeugt eine 2-D Grafik, wobei x und y Vektoren vom gleichen Typ sein müssen. Mit S werden die Art und die Farbe der Linie angegeben. Entfällt S, dann wird der Graph als Volllinie farbig dargestellt. Die Funktion ermöglicht es, gleichzeitig mehrere Kurven darzustellen und diese unterschiedlich farbig sowie grafisch zu kennzeichnen. Die einzelnen Befehle sind der MATLAB-Hilfe zu entnehmen.
`>> plot(x,y,z,S)`	Erzeugt eine 3-D Grafik, wobei x, y und z Vektoren vom gleichen Typ sein müssen. Für S gilt das unter `plot` gesagte.
`>> plotyy(`x_1`,`y_1`,`x_2`,`y_2`)`	Stellt zwei Grafiken mit unterschiedlicher Einteilung der Ordinaten in einer Figur dar. Die WErte von y_1 werden links und die von y_2 rechts aufgetragen.
`>> semilogx(x,y)`	Erzeugt eine halblogarithmische Darstellung der x-Achse auf der Basis 10 einer Funktion.
`>> semilogy(x,y)`	Erzeugt die halblogarithmische Darstellung der y-Achse auf der Basis 10 einer Funktion.
`>> set(a,'Eigenschaft',{Wert}` `>> set(f,'Eigenschaft',{Wert}`	Legt für den aktuellen Subplot oder die aktuelle Figur den Wert oder die Werte der benannten Eigenschaft fest.
`>> subplot(m,n,i)`	Stellt den i-ten Subplot bzw. die i-te Grafik von (m $\times$ n)-Grafiken in einer Figur dar. Die Figur hat m Zeilen und n Grafiken pro Zeile. Bei bereits vorhandener Figur weist sie auf den i-ten Subplot.
`>> text(x,y,'Text')`	Fügt den 'Text' in die Grafik an den entsprechenden Koordinaten x und y bei 2-D Grafiken ein.
`>> title('Text')`	Bezeichnung des Graphen oberhalb der Figur.
`>> xlabel('Text')` `>> ylabel('Text')`	Beschriftet die entsprechenden Achsen mit 'Text'.

4.8. Symbolische Rechnungen mit der Symbolic Math Toolbox

Tabelle E.8.: Übersicht: Symbolische Rechnungen

Funktion	**Command Window**	
	Eingabe	**Ausgabe**
Differentiation	`>> f = x^2 + 3; >> diff(f,x)`	`ans = 2*x`
Integration	`>> f = x^2 + 3; >> int(f,x)`	`ans = (x*(x^2 + 9))/3`
Grenzwerte hier: x $\rightarrow$ 0	`>> f = sin(x) / x; >> limit(f,x,0)`	`ans = 1`
Taylor-Reihen	`>> f = sin(x); >> taylor(f,n)`	`ans = x^5/120 - x^3/6 + x`
Fourier-Transformation	`>> f = exp(-x ^2); >> fourier(f,x,y)`	`ans = pi^(1/2)*exp(-y^2/4)`
Terme vereinfachen	`>> f = x^2 + 5 - 2; >> simplify(f)`	`ans = x^2 + 3`
Zahlen in Funktionsgleichungen einsetzen	`>> f = x^2 + 3; >> subs(f,n)`	`ans = n^2 + 3`
Gleichungen lösen	`>> S = solve(x^2 + 3 == 4); >> S=[S.x]`	`S = [1]`

Ist bei den oben aufgeführten Berechnungen jeweils nur eine Variable definiert, so muss diese nicht explizit angegeben werden. Die Angabe kann dann entfallen (Bsp.: `diff(f)` statt `diff(f,x)`).
Trigonometrische Funktionen können wir gewohnt als `sin`, `cos` und `tan` eingegeben werden, die zugehörigen Arcus-Funktionen sind mit `asin`, `acos` und `atan` anzugeben.

5. Aufgaben

In diesem Kapitel werden einige Aufgaben aus verschiedenen Tehmenkomplexen der Sammlung an Übungsaufgaben (moodle) genannt. Diese sind hier mit Matlab zu lösen. Die korrekte Eingabe in das Command Window ist für die jeweilige Aufgabe unter Kapitel 6 zu finden.

Aufgabe 1
Berechnen Sie den Real- und Imaginärteil von $\underline{z}$: $\underline{z} = (2+3i) \cdot (-3+2i)$

Aufgabe 2
Bestimmen Sie den Real- und Imaginärteil von $\underline{z}^2$ und $\underline{z}^3$ für: $\underline{z} = \sqrt{2} + \sqrt{2} \cdot i$

Aufgabe 3

Berechnen Sie für die folgende Funktion den Grenzwert für $x \to \infty$: $f(x) = \frac{4-3x}{x+1}$

Aufgabe 4

Berechnen Sie für die folgende Funktion den Grenzwert für $x \to \infty$: $f(x) = \frac{\sqrt{x-1}}{\sqrt{x+1}}$

Aufgabe 5

Berechnen Sie für die folgende Funktion den Grenzwert für $x \to 0$: $f(x) = \frac{tan(2x)}{5x}$

Aufgabe 6

Leiten Sie die folgende Funktion nach x ab: $f(x) = \frac{ln(x)+1}{x}$

Aufgabe 7

Leiten Sie die folgende Funktion nach t ab: $f(t) = sin(cos(t))$

Aufgabe 8

Leiten Sie die folgende Funktion nach x ab: $f(x) = \frac{x^2}{1-x^2}$

Aufgabe 9

Leiten Sie die folgende Funktion nach x ab: $y(x) = \frac{b^x}{ln(x)} - \frac{ln(b)}{b}$

Aufgabe 10

Berechnen Sie das totale Differential: $h(a, b) = 3a^2 \cdot sin(b)$

Aufgabe 11

Berechnen Sie das folgende Integral: $I = \int \frac{(ln(x))^3}{x}\, dx$

Aufgabe 12

Berechnen Sie das folgende Integral: $I = \int \frac{x}{(cos(x))^2}\, dx$

Aufgabe 13

Entwickeln Sie folgende Funktion in einer Taylor-Reihe bis zur 3. Ordnung ($n = 3$) für $x_0 = 0$:

$f(x) = 4^x$

Aufgabe 14

Entwickeln Sie folgende Funktion in einer Taylor-Reihe bis zur 3. Ordnung ($n = 3$) für $x_0 = 0$ und berechnen Sie den relativen Fehler in % für $x = 0,2$

$f(x) = e^{-x} \cdot cos(x)$

Aufgabe 15

Stellen Sie die Funktion $f(x) = (cos(x))^2$ grafisch dar.

Aufgabe 16

Führen Sie für die Funktion $f(x) = e^{-x^2}$ eine Fourier-Transformation durch.

6. Lösungen

Lösung 1

```
>> z=(2+3*i)*((-3)+2*i);
>> imag(z)
```

```
>> z=(2+3*i)*((-3)+2*i);
>> real(z)
```

Lösung 2

```
>> z=sqrt(2)+sqrt(2)*i;
>> imag(z^2)
```

```
>> z=sqrt(2)+sqrt(2)*i;
>> real(z^2)
```

```
>> z=sqrt(2)+sqrt(2)*i;
>> imag(z^3)
```

```
>> z=sqrt(2)+sqrt(2)*i;
>> real(z^3)
```

Lösung 3

```
>> syms x
>> f=(4-3*x)/(x+1);
>> limit(f,x,Inf)
```

Lösung 4

```
>> syms x
>> f=(sqrt(x-1))/(sqrt(x+1));
>> limit(f,x,Inf)
```

Lösung 5

```
>> syms x
>> f=(tan(2*x))/(5*x);
>> limit(f,x,0)
```

Lösung 6

```
>> syms x
>> f=(log(x)+1)/x;
>> simplify(diff(f))
```

Lösung 7

```
>> syms t
>> f=sin(cos(t));
>> diff(f)
```

Lösung 8

```
>> syms x
>> f=(x^2)/(1-x^2);
>> simplify(diff(f))
```

Lösung 9

```
>> syms x b
>> y=((b^x)/log(x))-(log(b)/b);
>> diff(y,x)
```

Lösung 10

Hier werden lediglich die partiellen Ableitungen berechnet! Die Gesamtlösung ist nach der bekannten Formel zusätzlich selbst zu erstellen.

```
>> syms h a b
>> h=3*a^2*sin(b);
>> diff(h,a)

>> diff(h,b)
```

Lösung 11

Hier wird das Ergebnis von MATLAB ohne die zugehörige Konstante C dargestellt.

```
>> syms x
>> f=((log(x))^3)/x;
>> int(f)
```

Lösung 12

```
>> syms x
>> f=x/((cos(x))^2);
>> int(f)
```

Lösung 13

Zu beachten ist hier, dass „bis zur 3.Ordnung“ bedeutet, dass die ersten vier Glieder der Reihe angezeigt werden sollen.

```
>> syms x
>> f=4^x;
>> taylor(f,4)
```

Lösung 14

Zu beachten ist hier, dass „bis zur 3.Ordnung“ bedeutet, dass die ersten vier Glieder der Reihe angezeigt werden sollen.

```
>> syms x
>> f=exp(-x)*cos(x);
>> t=taylor(f,4)

>> ((subs(t,0.2)-subs(f,0.2))/subs(f,0.2))*100
```

Lösung 15

```
>> syms x
>> x = 0:0.1:10;
>> plot(x,cos(x))
```

Lösung 16

```
>> syms x y
>> f = exp(-x^2);
>> fourier(f, x, y)
```

F. Skript zum Praktikum MATLAB - Teil 2: Indirekte Eingabe

Das Skript, welches den zweiten Teil des Praktikums MATLAB im Rahmen der Lehrveranstaltung Mathematik begleiten soll, befindet sich, inklusive der von den Studierenden im Praktikum zu bearbeitenden Aufgaben, auf den folgenden Seiten.

MATLAB
Teil 2

1. Allgemeines

Neben der in Teil 1 ausgeführten direkten Eingabe in MATLAB besteht noch die Möglichkeit der indirekten Eingabe über den MATLAB-Editor. Dieser ist dann sinnvoll, mehrere Berechnungen zur Lösung einer Problemstellung notwendig sind. Auf die verschiedenen Möglichkeiten der Nutzung des Editors wird nachfolgend eingegangen. Auch werden hier Befehle vorgestellt, die erst mit der Nutzung des MATLAB-Editors übersichtlich durchzuführen sind und deshalb erst an dieser Stelle Sinn ergeben.

2. Der MATLAB-Editor

Wird mit den Menü-Befehlen `File - Open ...` oder `File - New ...` ein M-File geöffnet oder neu angelegt, so öffnet sich der MATLAB-Editor. Auch öffnet er sich automatisch, wenn man im Current Folder-Fenster einen Doppelklick auf ein M-File ausführt oder im Kontext-Menü der rechten Maustaste `Open` anwählt. Zusätzlich besteht die Möglichkeit, in der Command History eine Gruppe von Befehlen anwählt und dann im Kontext-Menü der rechten Maustaste `Create M-File` auswählt. Das sich öffnende Editierfenster dient der Aufnahme des Programmtextes.

Werden Befehle in den MATLAB-Editor eingegeben, so werden diese nicht sofort nach der Eingabe ausgeführt, sondern erst dann, wenn das Programm ausgeführt wird. Dies ist durch Klicken auf das entsprechende Icon (grüner Pfeil in der Symbolleiste) oder durch Drücken von F5 zu bewerkstelligen.

3. Der MATLAB-Debugger

Bei der Entwicklung von Programmen können prinzipiell zwei Arten von Fehlern auftauchen: *Syntaxfehler* und *Laufzeitfehler*. Syntaxfehler sind meist relativ einfach zu beheben, da entsprechende Fehlermeldungen darauf hinweisen. Laufzeitfehler hingegen treten bei syntaktisch korrekten Programmen während der Ausführung auf und zeigen sich entweder in einem unerwarteten Programmabbruch (Absturz) oder offensichtlich falschen Ergebnissen. In solchen Fällen ist es notwendig, sich den Ablauf des Programms während der Ausführung genauer anzusehen. Hierfür steht der Debugger als Programm, mit dem andere Programme während der Ausführung an bestimmten Stellen (*Breakpoints*) unterbrochen und geprüft werden können, zur Verfügung. Mithilfe der Menüeinträge unter `Debug` oder durch Klick auf das Icon mit dem roten Punkt können die erwähnten Breakpoints gesetzt oder gelöscht werden. Dabei wird der Breakpoint immer in derjenigen Zeile gesetzt oder gelöscht, in der der Cursor gerade steht. Während der Ausführung eines Programms erfolgt dann ein automatischer Wechsel in den Editor, wenn das Programm an der Stelle des Breakpoints angehalten wird. Die bis dahin belegten Variablenwerte können nun angeschaut werden, indem man mit dem Mauszeiger (ohne Klick) auf die jeweilige Variable zeigt. Die bis dahin errechneten Werte werden dann in einem Fenster angezeigt. Die Variablenwerte können allerdings nicht nur angezeigt, sondern auch verändert werden. Wird das Programm anschließend im Debug-Modus fortgesetzt, wird mit diesen Werten weitergerechnet.

4. Grafiken in MATLAB

Bereits in Teil 1 sind die grundlegenden Befehle zur Erstellung von Grafiken mittels direkter Eingabe zu finden. Sollen die Grafiken etwas komplexer werden, ist es nützlich, diese mithilfe des Editors zu erstellen. Die dazu notwendige Vorgehensweise gleicht derjenigen zur Erstellung von Grafiken im Command Window, weshalb sie an dieser Stelle nicht noch einmal erläutert wird. Einige wichtige Befehle werden im Folgenden vorgestellt. Neben den nachfolgend vorgestellten Möglichkeiten bestehen noch zahlreiche weitere Optionen, verschiedene Diagramme, auch zur Visualisierung statistischer Auswertungen, darzustellen. Aus Zeitgründen kann nicht auf alle Möglichkeiten explizit eingegangen werden, einen Überblick kann man sich jedoch per MATLAB-Hilfe verschaffen.

4.1. Grafikeigenschaften

Tabelle F.1.: Übersicht über die Eigenschaften einer Grafik

Befehl	**Beschreibung**

`>> grid`
`>> grid on`
`>> grid off`
	Mit `grid` werden Gitternetzlinien angezeigt oder bestehende wieder entfernt. Dabei wird `grid on` für das Anzeigen und `grid off` für das Entfernen von Gitternetzlinien verwendet.

`>> legend('Kurve1', 'Kurve2')`	Einfügen einer Legende, wobei beim Erstellen mehrerer Kurven die jeweilige Bezeichnung in der Reihenfolge erfolgt, in der die Wertepaare im `plot`-Befehl gelistet werden.
`>> gtext('Text')`	Text, der mit der Maus in der Grafik positioniert wird. MATLAB wechselt dazu automatisch ins aktive Grafikfenster (also in das, das zuletzt geöffnet war).
`>> plot(x,y,'y--')`	Optionen, mit denen Farbe und Stil der Linien und der Punkttyp zur Markierung einzelner Werte festgelegt werden kann. Die Farbenwerte sowie Punkt- und Linientypen werden in einer separaten Tabelle nachfolgend aufgelistet.

Tabelle F.2.: Übersicht über Farbenwerte, Punkt- und Linientypen

Farbe	**Punkttyp**	**Linientyp**
`b` blau	`.` Punkt	`-` durchgezogen
`g` grün	`o` Kreis	`:` gepunktet
`r` rot	`x` x-Marker	`-.` Strich-Punkt
`c` cyan	`+` Plus	`--` gestrichelt
`m` magenta	`*` Stern	
`y` gelb	`s` Quadrat	
`k` schwarz	`d` Raute	
	`v` Dreieck (nach unten)	
	`^` Dreieck (nach oben)	
	`<` Dreieck (nach links)	
	`>` Dreieck (nach rechts)	
	`p` Pentagramm	
	`h` Hexagramm	

Zudem stehen mehrere Farbpaletten zur Verfügung. Diese sind mit dem Befehl `>> colormap('Name der Farbpalette')` anwählbar. Eine Übersicht über alle Farbpaletten bietet die MATLAB-Hilfe. Hier seien nur einige wenige exemplarisch genannt.

Tabelle F.3.: Übersicht über die Farbpaletten

Name	**Beschreibung**
hsv	Farbpalette des HSV-Farbraums (hue-saturation-value), vor allem zur Darstellung periodischer Funktionen geeignet.
jet	Ähnlich zu hsv, die Farben gehen hier von blau bis rot.
colorcube	So viele gleichmäßige Abstufungen des RGB-Farbraums, wie möglich.
gray	Grauskala mit linearer Abstufung der Grautöne.
hot	Geht von schwarz über rot und gelb bis weiß.
pink	Geht von schwarz über pinkfarbene Pastelltöne nach weiß, gibt damit die Sepiatönung wieder.

4.2. Mehrere Diagramme in einem Grafikfenster

Grundsätzlich gibt es zwei Möglichkeiten, mehrere Diagramme in einem Grafikfenster darzustellen. Zum Einen kann in einem Grafikfenster ein Diagrammfenster geöffnet sein, in dem verschiedene Kurven dargestellt werden. Zum Anderen ein Grafikfenster mehrere Unterdiagramme, neben- oder übereinander, enthalten.

Um verschiedene Kurven in einem Diagramm darzustellen, muss vehindert werden, dass jeder neue Grafikbefehl die vorherige Kurve überschreibt. Dazu dient der Befehl `hold`. Durch diesen Befehl wird eine bestehende Kurve so lange im Diagramm gehalten, bis durch eine erneute Eingabe von `hold` das Haöten abgeschaltet wird.

Um mehrere Unterdiagramme in ein Diagrammfenster zu integrieren, dient der Befehl `subplot`. Dabei wird bei der Eingabe von `subplot(z,s,n)` ein Dagrammfenster geöffnet, dass Platz für $z \cdot s$ Unterdiagramme bietet. Durch n wird bestimmt, an welcher Stelle das aktuell eigegebene Diagramm steht. Bespielsweise wird durch die Eingabe von `subplot(2,3,5)` ein Grafikfenster erzeugt, das sechs Unterdiagramme aufnimmt, die in zwei Zeilen (z) und drei Spalten (s) angeordnet werden. Das nachfolgend eingegebene Diagramm erscheint an fünfter Stelle, wobei von links nach rechts und von oben nach unten gezählt wird.

4.3. Grafiktypen

Tabelle F.4.: Übersicht über Grafiktypen

Befehl	**Beschreibung**
`>> area(y)`	Der `area`-Befehl entspricht dem Plot-Befehl, nur ist hier die Fläche unter der Kurve farbig ausgefüllt. Ist `y` eine Matrix, so werden die Werte jeder Zeile aufsummiert, wobei die einzelnen Werte jeder Spalte jeweils eine eigene Farbe erhalten.
`>> imagesc(C)`	Stellt die Matrix C als Grafik dar, wobei jedes Element innerhalb von C einem rechteckigen Bereich in der entstehenden Grafik entspricht. Die Werte der einzelnen Elemente von C bestimmen, welche Farbe aus der ausgewählten Farbpalette dem entsprechenden Bereich in der Grafik zugewiesen wird.
`>> plot3(X,Y,Z)`	Der `plot3`-Befehl entspricht dem `plot`-Befehl, nur dass in diesem Fall die Eingabe von drei Vektoren bzw. Matrizen verlangt wird, deren Elemente die Koordinaten für die x-, y- und z-Richtung enthalten.

`>> mesh(Z)`
`>> mesh(X,Y,Z)`
`>> mesh(...,C)`
	Mit `mesh(Z)` werden Gitternetze erzeugt, die geometrische Oberflächen mit Höhen und Tiefen, die in `Z` definiert sind, über die x-y-Ebene darstellt werden können.
	Ist nur `Z` angegeben, bestimmt die Anzahl der Spalten den x-Wertebereich un die Anzahl der Zeilen von `Z` den y-Wertebereich. Ansonsten bestimmen die Koordinaten in `X`, `Y` und `Z` die jeweiligen Kreuzungspunkte der Gitterlinien und die Werte von `X` und `Y` jeweils den x- und y-Wertebereich.
	Der Wert von `Z` bestimmt im Normalfall die Farbe, so dass die Farbe proportional zu den Oberflächenhöhen ist. Es kann jedoch auch ein Parameter `C` angegeben werden, mit dem die Farbverteilung anhand der aktuellen Farbpalette bestimmt wird.

`>> meshgrid`	Zeigt die dem mesh-Plot entsprechend sinnvollen dreidimensionalen Gitternetzlinen an.

`>> surf(Z)`
`>> surf(X,Y,Z)`
`>> surf(...,C)`
	Mit `surf(Z)` können mathematische Funktionen oder andere Werte in `Z` über der x-y-Ebene als Oberfläche (surface) mit Höhen und Tiefen in höhenabhängigen Farbschattierungen darstellt werden.
	Generell ist der Befehl `surf` geanu so handzuhaben, wie der Befehl `mesh`. Der Unterschied zwischen den beiden Befehlen besteht darin, dass die dargestellten Flächen bei `surf` ausgefüllt sind.

5. Programmierung

Wenn nicht die Funktion der numerischen Berechnungen, sondern die Option, MATLAB als Programmiersprache zu nutzen, eingesetzt werden soll, bietet sich der MATLAB-Editor ebenfalls an. Hier stehen dann auch typische Programmiersprachenkonstrukte wie Schleifen zur Verfügung, ebenso können Funktionen und Prozeduren geschrieben werden.

5.1. MATLAB-Prozeduren

Der erste Schritt zur Programmierung ist die Erstellung von *Script-Files*, in denen MATLAB-Befehlssequenzen zu einfachen Prozeduren zusammengefasst werden. Dazu wird ein neues M-File im MATLAB-Editor geöffnet. In diesen wird nun die gewünschte Befehlssequenz geschrieben und unter einem Namen (z. B. **prozed.m**) in einem für MATLAB zugänglichen Pfad abgespeichert. Die Befehlssequenz kann anschließend im Command Window mit dem Befehl `prozed` ausgeführt werden. Nach Möglichkeit sollte hier ein bereits existierender Befehlsname verwendet werden. Ob ein solcher schon vorhanden ist, kann einfach überprüft werden: Zum einen kann `help <name>` in das Command Window eingegeben werden, zum Anderen kann der Aufruf `exist <name>` ausgeführt werden. Gibt dieser den Wert 0 zurück, so exisitiert im Suchpfad noch keine Funktion dieses Namens.

Sollen in eine Prozedur Kommentare eingefügt werden, so sind diese im MATLAB-Editor mit einem vorangestellten % zu kennzeichnen. Wird dann im Command Window der Befehl `help <name>` ausgeführt, so werden die Kommentare der jeweiligen Prozedur, die vor der ersten leeren Zeile oder der ersten MATLAB-Anweisung stehen, ausgegeben.

Die im Script-File definierten Variablen sind nach Ausführung der jeweiligen Prozedur im MATLAB-Workspace bekannt.

5.2. MATLAB-Funktionen

MATLAB-Befehle können nicht nur in Script-Files zusammengefasst, sondern auch als *Funktionen* definiert werden. Dies ist wesentlich flexibler, da diesen Funktionen Parameter übergeben werden können. Das bedeutet, dass hier nicht nur spezifische Ergebnisse berechnet, sondern, wie der Name sagt, tatsächliche Funktionsgleichungen programmiert werden können. Auch hierzu wird ein M-File geöffnet und unter einem Dateinamen, der dem Funktionsnamen entspricht, abgespeichert. Letzteres ist wichtig, weil MATLAB Funktionen immer in einem namensgleichen M-File sucht.

Die übrige Vorgehensweise gleicht derjenigen zur Erstellung einer Prozedur. Der Unterschied zu Script-Files besteht darin, dass innerhalb einer Funktion nicht auf Variablen aus dem Workspace zugegriffen werden kann.

Auf eine im MATLAB-Editor programmierte Funktion kann nachher im Command Window zugegriffen werden, indem die Funktion mit Werten für die jeweils angegebenen Parameter eingegeben wird. Das bedeutet insgesamt, dass die Funktion an sich im Editor programmiert wird und anschließend für Berechnungen im Command Window nutzbar ist.

Sonderfall: Die Funktion `eval`
Mit der Funktion `eval` ist es möglich, Strings als MATLAB-Ausdrücke wie beispielsweise Befehle auszuwerten. Dazu wird (bei Eingabe in den MATLAB-Editor) eine Variable definiert, innerhalb derer dann wiederum eine Folge von MATLAB-Befehlen als String definiert werden kann. Dieser String wird durch den anschließenden `eval`-Befehl als MATLAB-Befehlsfolge interpretiert und ausgeführt.

Zum besseren Verständnis wird nachfolgend ein Beispiel vorgestellt.

```
clear all
neuerBefehl = ['x = 3;', 'y = 4;','z = x + y']
eval(neuer Befehl)
```

Wird diese Funktion abgespeichert und ausgeführt, so kann anschließend der Befehl `z` in das Command Window eingegeben werden, woraufhin MATLAB das offensichtlich korrekte Ergebnis `7` ausgibt.

5.3. MATLAB-Sprachkonstrukte

Die Funktion von MATLAB als Programmiersprache wird besonders in den Sprachkonstrukten sichtbar. Solche Kontrollstrukturen sind bei der Programmierung sinnvoll, da hierdurch Schleifen, Verzweigungen oder Fehlerkontrollen erzeugt werden können.

5.3.1. for-Schleife

Die for-Schleife ist eine Zählschleife, mit der Anweisungen so oft wiederholt werden können, wie es ein vorgegebener Zahlenwert angibt. Auch können die Zahlenwerte in Befehle integriert werden, beispielsweise beim Aufsummieren bestimmter Zahlen, bis ein vorgegebener Endwert erreicht ist.

5.3.2. while-Schleife

Eine MATLAB-Anweisung oder eine Folge solcher Anweisungen wird solange wiederholt, bi seine bestimmte logische Bedingung erreicht wird. Diese Bedingung muss im Programm direkt hinter `while` stehen.

5.3.3. if-else-elseif-Verzweigung

Hierbei wird eine Bedingung erfragt. Ist diese erfüllt, so wird die dazugehörige Anweisung oder die entsprechende Folge von Anweisungen ausgeführt. Die Bedingung ist dabei hinter `if`, die Anweisung ist hinter `else` anzugeben. Wird die Bedingung nicht erfüllt, so können weitere Abfragen vorgegeben werden, die dann eine andere Anweisung oder eine Folge von Anweisungen zur Folge haben. Die weiteren Abfragen sind hinter `elseif` einzugeben. Wird keine der hinter `elseif` angegebenen Abfragen erfüllt, so wird auf die hinter `else` angegebene Abfrage zurückgegriffen.

5.3.4. switch-case-otherwise-Verzweigung

Mithilfe dieser Verzweigung können Status, Wert oder Zustand einer Variablen über verschiedene Alternativen abgefragt werden. Je nach zutreffendem Fall (case) wird der abgefragte Status, Wert oder Zustand in eine Anweisung oder eine Folge von Anweisungen integriert.

5.3.5. try-catch-Fehlerkontrolle

Mit dieser Fehlerkontrolle kann der unbeabsichtigte Programmabbruch (Absturz) eines M-Files verhindert werden. Wird ein Fehler bei einer Anweisung gefunden (try), so werden Alternativanwendungen (catch) ausgeführt, die auf den Fehler reagieren. Beispielsweise kann hier eine Warnmeldung in ein Programm integriert werden.

5.3.6. Übersicht über wichtige Befehle

Die folgende Tabelle gibt zum Einen die korrekte Syntax zur Verwendung der oben genannten Sprachkonstrukte an, zum Anderen werden weitere Befehle vorgestellt, die im Zusammenhang mit der Programmierung unter Anwendung von Sprachkonstrukten hilfreich erscheinen. Selbstverständlich können auch alle in den Übersichten in Teil 1 aufgeführten Befehle in Sprachkonstrukte und Programme integriert werden. Die folgende Tabelle erhebt keinen Anspruch auf Vollständigkeit.

Tabelle F.5.: Übersicht über wichtige Befehle

Befehl oder Befehlsfolge	**Beschreibung**

`>> for i=zahl`
`  Anweisung 1`
`  Anweisung 2`
`  ...`
`  Anweisung n`
`end`
	Eine for-Schleife muss immer mit einem Zahlenwert, dem ein Index `i` zugewiesen wird, beginnen und mit `end` enden. Dazwischen befindet sich der Abschnitt mit der Anweisung oder der Folge von Anweisungen. Hier können nicht nur Befehle stehen, sondern auch weitere Schleifen eingefügt werden, die dann allerdings innerhalb der for-Schleife beendet werden müssen. Auch hier ist, wie in MATLAB generell, der Ausdruck `zahl` eine Matrix.

`>> for j=s:d:n`
`  Anweisungen`
`end`
`>> for i=s:n`
`  Anweisungen`
`end`
	Soll in der for-Schleife keine Matrix, sondern ein mathematischer Ausdruck stehen, so sind der Indexvariablen `i` Zahlen vom Startwert `s` bis zum Endwert `n` zuzuordnen, wobei `d` den Abstand zwischen den Zahlenwerten definiert. Wird für diesen Abstand keine Angabe gemacht, ist der Abstand automatisch 1.

`>> while matrix`
`  Anweisung 1`
`  Anweisung 2`
`  ...`
`  Anweisung n`
`end`
	Das Ergebnis der logischen Bedingung kann eine Matrix sein: Solange alle Elemente der Matrix ungleich Null sind, wird die Schleife durchlaufen. Dies gilt auch für Sonderfälle einer Matrix wie beispielsweise Skalare.
	Am Ende einer while-Schleife muss immer `end` stehen.

`>> while Abfrage`
`  Anweisung 1`
`  Anweisung 2`
`  ...`
`  Anweisung n`
`end`
	Die Bedingung der while-Schleife kann auch eine logische Abfrage mit dem Ergebnis wahr (1) oder falsch (0) sein. Die Schleife wird ausgeführt, wenn die Bedingung wahr ist und endet, wenn die Bedingung falsch ist.
	Auch innerhalb einer while-Schleife können andere Schleifen enthalten sein.

`>> if Ausdruck 1`
`  Anweisungen 1`
`elseif Ausdruck 2`
`  Anweisungen 2`
`elseif Ausdruck 3`
`  Anweisungen 3`
`...`
`else`
`  Anweisungen n`
`end`
	Die if-elseif-else-Verzweigung muss immer mit `if` starten. Eine elseif-Verzweigung ist nicht notwendig, es können allersings bei Bedarf eine oder mehrere solcher Verzweigungen angegeben werden.
	Als Abschluss muss wieder `end` stehen. Hinter `if` und `elseif` stehen logische Bedingungen, z. B. `if i>0`.

`switch variable`
`case Wert 1`
`  Anweisungen 1`
`case {Wert 2,Wert 3}`
`  Anweisungen 2`
`...`
`case Wert n`
`  Anweisungen n`
`otherwise`
`  Anweisungen n+1`
`end`
	Die abzufragende Variable wird hinter `switch` eingefügt. Hinter jedem `case` steht ein möglicher Wert für die Variable. Das kann sowohl eine Zahl als auch ein Wort sein, welches dann allerdings in Hochkommata gesetzt werden muss. Zahlenwerte und Wörter können hierbei auch gemischt werden.
	Mehrere mögliche Alternativwerte, die die gleichen Anweisungen nach sich ziehen, können in geschweiften Klammern, durch Kommata getrennt, aufgelistet werden. Auch diese Verzweigung muss mit `end` abgeschlossen werden.

`try`
`  Anweisungen 1`
`catch`
`  Anweisungen 2`
`end`
	Die Anweisungen hinter `try` können einen Fehler bewirken, beispielsweise wenn versucht wird, eine Variable aufzurufen, die nicht exisitert. Mit den Anweisungen hinter `catch` wird der mögliche Fehler aufgefangen, z. B. durch eine Warnmeldung, die zur Eingabe der nicht existierenden Variablen auffordert.
	Verschachtelte Schleifen oder Verzweigungen innerhalb der try-catch-Anweisung sind nicht möglich.
	Mit dem Befehl `lasterr` kann der Fehler oder Grund, warum in `catch` abgezweigt wurde, ausgegeben werden.

`>> break`	Mit `break` kann eine for- oder while-Schleife verlassen werden. Das Programm fährt dann mit den Anweisungen nach der Schleife fort. Sinnvoll ist es, diesen Befehl an eine if-Verzweigung zu koppeln, beispielsweise indem bei einer while-Schleife eine Zählvariable mitläuft. Nach einer definierten Anzahl von Durchläufen beendet `break` die Schleife und dient somit als Notbremse im Fall einer unendlichen Schleife.
`>> continue`	Mit `continue` wird der Durchgang einer for- oder while-Schleife verlassen, wobei alle nach `continue` folgenden Anweisungen ignoriert werden und in den nächsten Durchgang der Schleife gesprungen wird. Auch dieser Befehl sollte an eine if-Verzweigung geknüpft werden, beispielsweise indem vor einer Wurzel abgefragt wird, ob das Argument unter der Wurzel negativ ist. Dann würde die Schleife für diesen Fall nicht ausgeführt und stattdessen zur nächsten Zahl in der Schleife gesprungen. In verschachtelten Schleifen springt `continue` zum Beginn derjenigen Schleife, die gerade abläuft.

`>> return`	Mit `return` kann der Anlauf eines Programms vorzeitig beendet werden. MATLAB kehrt dann sofort zum aufrunfenden Programm zurück, falls ein Unterprogramm beendet wurde. Wurde ein Hauptprogramm vorzeitig beendet, so kehrt MATLAB zum Command Window zurück. Auch `return` sollte mit einer if-Verzweigung verknüpft werden, beispielsweise wenn ein Unterprogramm zur Berechnung einer Wurzel einer eingegebenen Variablen nur dann ausgeführt werden soll, wenn die Variable positiv ist.
`>> nargin`	Anzahl der Argumente, die in die entsprechende Funktion eingegeben werden
`>> nargout`	Anzahl der Argumente, die von der entsprechenden Funktion ausgegeben werden
`>> varargin`	Eingabe-Variable innerhalb der Definition einer Funktion, die es der Funktion erlaubt, jede mögliche Anzahl an Argumenten anzunehmen.
`>> varargout`	Ausgabe-Variable innerhalb der Definition einer Funktion, die es der Funktion erlaubt, jede mögliche Anzahl an Argumenten auszugeben.

6. Aufgaben

Die folgenden Aufgaben fordern die Programmierung von Prozuduren und/oder Funktionen. Dazu ist auch die Anwendung von Sprachkonstrukten notwendig. Es gibt jeweils zahlreiche Möglichkeiten, die Aufgaben zu lösen. Die unten angegebenen Tipps beziehen sich auf die jeweils angegebene Lösung, welche nur eine von vielen möglichen Lösungen ist.

Aufgabe 1

Schreiben Sie eine Funktion, mit der komplexe Zahlen von der kartesischen in die polare Schreibweise konvertiert werden können. Der Winkel (das Argument) soll dabei sowohl in rad als auch in Grad berechnet und angegeben werden.

Aufgabe 2

In der Anlage zu diesem Skript wird die Mandelbrot-Menge kurz erklärt. Arbeiten Sie diese Erläuterungen durch und schreiben Sie anschließend eine Funktion, die die Mandelbrot-Menge in Abhängigkeit der Anzahl an durchgeführten Iterationen und in unterschiedlicher Auflösung grafisch darstellt.

Aufgabe 3

Berechnen Sie die Fourier-Koeffizienten folgender räumlicher, 2π-periodisch fortgesetzter

Funktion (Rechteckschwingung):

$$f(x) = \begin{cases} 1 & \text{für } 0 \leq x \leq \pi \\ -1 & \text{für } \pi < x < 2\pi \end{cases} \text{ mit } f(x) = \sum_{n=1}^{\infty} b_n sin(nx)$$

Stellen Sie zusätzlich mithilfe der trigonometrischen Polynome grafisch dar, wie sich die angegebene Reihe mit Zunahme der berechneten Glieder immer weiter an die Rechteckschwingung annähert.

7. Tipps zu den Aufgaben

Tipps zu Aufgabe 1

Entscheiden Sie zunächst, von welchen Variablen die Funktion abhängen soll.

Überlegen Sie, welche Parameter eingegeben werden müssen und welche die Funktion ausgeben soll.

Berechnen Sie zuerst den Betrag (=Radius) der komplexen Zahl.

Zur Berechnung des Winkels: Bedenken Sie, dass eine Fallunterscheidung, je nach Quadrant, gemacht werden muss.

Die Fallunterscheidung zur Berechnung des Winkels lässt sich mithilfe von if-Schleifen realisieren.

Tipps zu Aufgabe 2

Eine einstellbare Auflösung lässt sich realisieren, indem die Anzahl der Pixel, in die die Grafik aufgeteilt wird, variabel gehalten wird.

Beachten Sie: Je mehr Iterationen durchgeführt werden, desto länger dauert die Berechnung. Ebenso wirkt sich die Auflösung auf die Rechenzeit aus.

Legen Sie eine Grundeinstellung für die Anzahl der Iterationen und die Auflösung fest. Dazu eignet sich eine switch-case-Struktur.

Berechnen Sie jeden Pixel einzeln, indem sie eine Matrix definieren, die alle Bildpunkte der Grafik enthält. Jeder Bildpunkt benötigt einen Startwert!

Zur Durchführung der Iterationen eignet sich eine for-Schleife.

Nutzen Sie eine colormap, um ihre Grafik zu gestalten.

Tipps zu Aufgabe 3
Schreiben Sie zunächst ein Programm, das nur die vorgegebenen Funktionen definiert. Dazu eignet sich eine switch-case-Struktur.

Speichern Sie Ihre Funktionen ab, so dass sie im Folgenden auf diese zurückgreifen können.

Legen Sie für die Fourier-Transformation eine Prozedur an.

Berechnen Sie zunächst die Fourier-Koeffizienten.

Berechnen Sie anschließend die trigonometrischen Polynome.

Erstellen Sie die Grafik mit der Rechteckschwingung und den trigonometrischen Polynomen.

8. Lösungen

Lösung 1

```
function [Betrag, Winkel, WinkelGrad]= komplex(c)
% Die Funktion wird unter dem Namen komplex angelegt. Die
   einzugebende
% Variable c ist die komplexe Zahl, die konvertiert werden
   soll.
%
% Aufruf: [Betrag, Winkel, WinkelGrad] = komplex(c)
%
% Eingabeparameter:  c            Zahl (reell oder komplex (
   kartesisch!))
% Ausgabeparameter:  Betrag       Betrag von c (= Radius)
%                    Winkel       Winkel von c im Bogenmaß
%                    WinkelGrad   Winkel von c im Grad
%
% Beschreibung: Das Programm rechnet komplexe Zahlen von der
    kartesischen
% in die polare Form um und gibt Betrag (Radius) und Winkel
   in rad und Grad
```

```
% an.

% Betrag berechnen:
Betrag = abs(c);

% Real- und Imaginärteil berechnen:
creal = real(c);
cimag = imag(c);

% Quadrant feststellen:
if creal > 0
    if cimag >= 0                              % 1. Quadrant
        Winkel = atan(cimag/creal);
    else
        Winkel = atan(cimag/creal) + 2*pi;  % 4. Quadrant
    end;
elseif creal < 0                               % 2. und 3.
   Quadrant
    Winkel = atan(cimag/creal) + pi;
else                                           % Sonderfall:
   Realteil = 0
    if cimag >= 0                                  %
       Imaginärteil positiv
        Winkel = pi/2;
    else                                       % Imaginärteil
       negativ
        Winkel = 3*pi/2;
    end;
end;

% Winkel in Grad umrechnen:
WinkelGrad = Winkel*180/pi;
end
```

Lösung 2

```
function mandelbrot(iter,pixel)
```

```
% mandelbrot(iter,pixel) gibt die Mandelbrot-Menge grafisch
   aus.
% iter: Anzahl Iterationen
% pixel: Anzahl Punkte auf x-Achse, für die gerechnet wird

% Grundeinstellung: iter=100, pixel=400 (Rechenzeit!)

% Grundeinstellung:
switch nargin
    case 0
        iter=100;
        pixel=400;
end

% Achsen festlegen, dabei 3/4-Verhältnis für Aufteilung der
   Pixel
% auf die Achsen beachten:
r = 3/4;
x = linspace(-2.5,1.5,pixel);
y = linspace(-1.5,1.5,round(pixel*r));

% Matrix C erzeugen
% C muss alle Pixel enthalten, die berechnet werden sollen
% => Achsen von x, y auf Re, Im umdefinieren
[Re,Im] = meshgrid(x,y);
C = Re + i * Im;

% Matrix B weist jedem Pixel eine Zahl zu, die abh. ist von
   der
% Divergenzgeschwindigkeit.
% B und C haben deshalb dieselbe Dimension.
% Festlegung von Null als Startwert für alle Pixel,
% dargestellt als Matrix!
B = zeros(round(pixel*r),pixel);
```

```
Cn = B;                          % C0 = 0+0i
for l = 1:iter
    Cn = Cn.*Cn + C;             % berechnet die Grafik
    B = B + (abs(Cn)<2);         % wenn |cn| > 2 liegt Divergenz
       vor, da die
                                 % Mandelbrot-Menge die fraktale
                                    Dimension 2
                                 % hat.
end;

% Grafikeinstellungen:
imagesc(B);
colormap(jet);

axis equal
axis off
```

Lösung 3

Funktion programmieren/definieren:

```
function y = fourieraufgabe(x,n)
% f(x) = 1 (0:pi)
% -1 (pi:2pi)
switch n,
   case 'f',
      for k = 1:length(x),
         if x(k) <= pi,
           y(k) = 1;
         else
           y(k) = -1;
         end;
      end;
case 'f1',
   y = 4/pi*1*sin(x);
case 'f2',
   y = 4/pi*(1*sin(x) + 1/3*sin(3*x));
```

```
case 'f3',
   y = 4/pi*(1*sin(x) + 1/3*sin(3*x) + 1/5*sin(5*x));
case 'f4',
   y = 4/pi*(1*sin(x) + 1/3*sin(3*x) + 1/5*sin(5*x) +1/7*sin
      (7*x));
end;
end
```

Fourier-Transformation (als Prozedur):

```
clear;
clc;
% symbolisches Berechnen der Fourier-Koeffizienten
syms x n;
% an
b_n = 1/pi*(int(sin(n*x),x,0,pi)-int(sin(n*x),x,pi,2*pi));
n = 1; b_1 = eval(b_n)
n = 2; b_2 = eval(b_n)
n = 3; b_3 = eval(b_n)
n = 4; b_4 = eval(b_n)
n = 5; b_5 = eval(b_n)
% Berechnen der x- und y-Werte
% Funktion
xf = 0:0.01:2*pi;
yf = fourieraufgabe(xf,'f');
% trigonometrische Polynome
n = 1:5;
x = 0:0.01:4*pi;
yf1 = fourieraufgabe(x,'f1');
yf2 = fourieraufgabe(x,'f2');
yf3 = fourieraufgabe(x,'f3');
yf4 = fourieraufgabe(x,'f4');
% graphische Darstellung
figure;
clf;
% Funktion
hold on;
```

```
grid on;
% Achsen
plot([-1 4*pi+1],[0 0],'-','Color','k','Linewidth',1.5);
plot([0 0],[-1.5 1.5],'-','Color','k','Linewidth',1.5);
% Funktion
plot([xf xf+2*pi],[yf yf],'-','Color','b','Linewidth',3);
% trigonometrische Polynome
plot(x,yf1,'-','Color','m','Linewidth',1.5);
plot(x,yf2,'-','Color','g','Linewidth',1.5);
plot(x,yf3,'-','Color','c','Linewidth',1.5);
plot(x,yf4,'-','Color','y','Linewidth',1.5);
% Beschriftung
xlabel('x','FontSize',12);
ylabel('y','FontSize',12);
title('Aufgabe␣3','FontSize',12,'Fontweight','bold');
legend('x-Achse','y-Achse','f(x)','F_0(x)','F_1(x)','F_2(x)'
   ,'F_3(x)','Location','NorthEast');
% Achsen
axis([-1 4*pi+1 -1.5 1.5]);
```

Anlage: Die Mandelbrot-Menge

Die Mandelbrot-Menge ist definiert als die Menge der komplexen Zahlen c, für die die Folge $z_n+1 = z_{2n}+c$ mit $z_0 = 0$ für $n \to \infty$ beschränkt bleibt. Das bedeutet, dass man jeweils das vorangegangene Ergebnis für z quadrieren und dann c hinzuaddieren muss, um das nächste Glied von z zu erhalten, welches wiederum Ausgangspunkt für den nächsten Rechendurchgang ist. Gehen für den gewählte Wert von c die Werte von z nicht gegen unendlich, so ist c Teil der Mandelbrot-Menge. Um in den Randbereichen realtiv sicher sein zu können, ob eine Zahl c noch zur Mandelbrot-Menge gehört, muss man unter Umständen hunderte oder tausende Folgenglieder berechnen.
Zur grafischen Darstellung färbt man alle c, die zur Mandelbrot-Menge gehören ein. Der Realteil von c wird im Koordinatensystem jeweils nach rechts und der Imaginärteil von c nach oben aufgetragen.

Eine Iteration ist grundsätzlich dann beschränkt, wenn eine beliebig große Schranke s existiert, so dass alle Zahlen z_n kleiner als s sind. Bei dieser speziellen Iteration ist es so, dass sie auf jeden Fall alle Schranken überschreitet, wenn der Betrag eines Elements ≥ 2 ist (die fraktale Dimension der Mandelbrot-Menge ist damit 2).

Um die Mandelbrot-Menge grafisch darstellen zu können, berechnet man für jeden Punkt des Bildes die Folge mit seinen Koordinaten. Dazu legt man eine maximale Anzahl an Iterationen (das heißt Anzahl an Folgengliedern, die berechnet werden) fest und prüft nach jeder Iteration, ob $z \geq 2$ ist. Falls ja, ist der Punkt mit den zugehörigen Koordinaten definitiv nicht Teil der Mandelbrot-Menge. Ist diese Bedingung nach einer bestimmten Anzahl an Iterationen noch nicht erfüllt, so kann man mit hoher Wahrscheinlichkeit davon ausgehen, dass der geprüfte Punkt Teil der Mandelbrot-Menge ist. Je höher hier die Anzahl der Iterationen ist, desto sicherer ist auch das Ergebnis. Die Punkte, die zur Mandelbrot-Menge gehören, werden dann eingefärbt.

Soll die Mandelbrot-Menge nicht in einer, sondern in mehreren Farben eingefärbt werden, so wählt man als Unterscheidungsmerkmal für die Zuteilung der jeweiligen Farbe die Divergenzgeschwindigkeit des jeweiligen Punktes. Als Maß für diese kann die Anzahl an Iterationen, nach denen die Berechnung abgebrochen wurde, dienen.

G. Skript zur MATLAB-Anwendung: FIR-Filter

Das Skript, welches den Studierenden zur Vertiefung der Thematik der FIR-Filter zur Verfügung gestellt werden soll, befindet sich auf den folgenden Seiten.

Finite-Impulse-Response-Filter (FIR)

1. Filter

In der digitalen Signalverarbeitung (DSV) werden Filter meist benutzt, um aus einem Signal verschiedene Frequenzbereiche auszusondern. Elektronische Filterschaltungen sind Baugruppen oder Programme, die demnach das Frequenzspektrum eines Signals verändern.

In der Hörakustik werden Filter benötigt, um die Verstärkung verschiedener Frequenzbereiche unterschiedlich zu gestalten. Dies kommt bei der Einstellung von Hörgeräten zum Einsatz, da die Hörverluste, auf die die Hörgeräte eingestellt werden, selten pantonal verlaufen und die Einstellung der Hörgeräte deshalb frequenzabhängig erfolgt. Dazu muss das eingehende Signal in einzelne Frequenzbereiche aufgeteilt werden. Die Aufteilung erfolgt mithilfe von Filtern, die in Filterbänken angeordnet sind. Die entstehenden Frequenzbereiche werden dann als Bänder oder Kanäle bezeichnet. [22]

Filter lassen sich anhand verschiedener Unterscheidungsmerkmale in Gruppen aufgliedern. So kann beispielsweise nach der Übertragungsfunktion (Hochpass-, Tiefpass-, Bandpassfilter etc.), nach der Abhängigkeit vom Signalpegel (lineare und nichtlineare Filter) oder nach der Filtertechnologie unterschieden werden. Unterscheidet man Filter nach der eingesetzten Technologie, so kann eine Gliederung in analoge und digitale Filter vorgenommen werden. Im Folgenden wird näher auf die digitalen Filter eingegangen, da deren Einsatz weitaus verbreiteter ist als derjenige der analogen Filter. [23]

2. Digitale Filter

Digitale Filter können in zwei Klassen unterschieden werden: Nichtrekursive Filter (Finite-Impulse-Response-Filter, FIR) und rekursive Filter (Infinite-Impulse-Response-Filter, IIR). Nichtrekursive Filter (FIR) haben keinerlei Rückkopplungen, ihr Ausgang ist lediglich vom Eingangssignal und endlich vielen früheren Eingangswerten $x(n)$ abhängig. Diese werden in den Zustandsspeichern abgelegt. Die Antwort auf einen Eingangsimpuls ist immer von end-

licher Zeitdauer.
Bei den rekursiven Filtern (IIR) ist eine Rückkopplung vorhanden, der Augang hängt also zusätzlich von vergangenen Ausgangswerten $y(n)$ ab.

Mathematisch können digitale Filter durch Pole und Nullstellen in der komplexen Ebene beschrieben werden. Durch Verändern der Pole und Nullstellen, die immer einzeln auf der x-Achse oder als Paar symmetrisch zu dieser vorkommen, lassen sich die Filtereigenschaften verändern. [24]

Die Funktionsweise von digitalen Filtern lässt sich anhand eines gleitenden Mittelwertbildners erklären. Dieser multipliziert eine bestimmte Anzahl der in den Zustandsspeichern abgelegten Abtastwerte jeweils mit einem konstanten Faktor, dem sogenannten Filterkoeffizienten und addiert diese Werte anschließend. Die Charakteristik eines Filters ändert sich nicht, wenn alle Werte mit einem gleichen Faktor multipliziert werden. Die Mittelwertbildung erfolgt, iwe erwähnt, gleitend. Das bedeutet, dass der Mittelwert jeweils nach einer bestimmten Anzahl an Abtastpunkten, nämlich der Anzahl der eingesetzten Filterkoeffizienten, gebildet wird. Trägt man diese Mittelwerte in ein Diagramm ein, so verläuft die entstehende Kurve deutlich „glatter“ als eine Kurve, die aus den Einzelwerten der Messung gebilten werden könnte. Eine Mittelwertbildung bewirkt also die Glättung eines Kurvenverlaufs, außerdem besitzt sie eine Tiefpassfunktion. Werden die Filterkoeffizienten eines digitalen Filters variiert, so kann die Filterwirkung nahezu beliebig gestaltet werden. Es ist möglich, die benötigten Filterkoeffizienten für die jeweils gewünschte Filterfunktion von geeigneten Programmen berechnen zu lassen. [25]

3. Anwendungsbeispiel: FIR-Filter

Als Anwendungsbeispiel wird hier ein MATLAB-Programm zur Thematik der FIR-Filter, hier exemplarisch an einem Bandpassfilter dargestellt, gezeigt (siehe Abb. G.1). Ein FIR-Filter hat in der komplexen Zahlenebene immer n Nullstellen und einen n-fachen Pol in Ursprung der Ebene. Sein Verhalten wird daher mathematisch durch die Lage der Nullstellen beschrieben. [26]
Diese sind im Programm ebenso sichtbar wie die einzelnen Filterkoeffizienten, welche von MATLAB selbst berechnet werden und der Amplitudengang des jeweiligen Bandpassfilters in linearer und logarithmischer Darstellung. Das Programm bietet die Möglichkeit, die untere und obere Grenzfrequenz des Bandpasses mithilfe der Schieberegler zu verändern. Dabei sind für die untere Grenzfrequenz Werte zwischen 0 und 5000 Hz und für die obere Grenzfre-

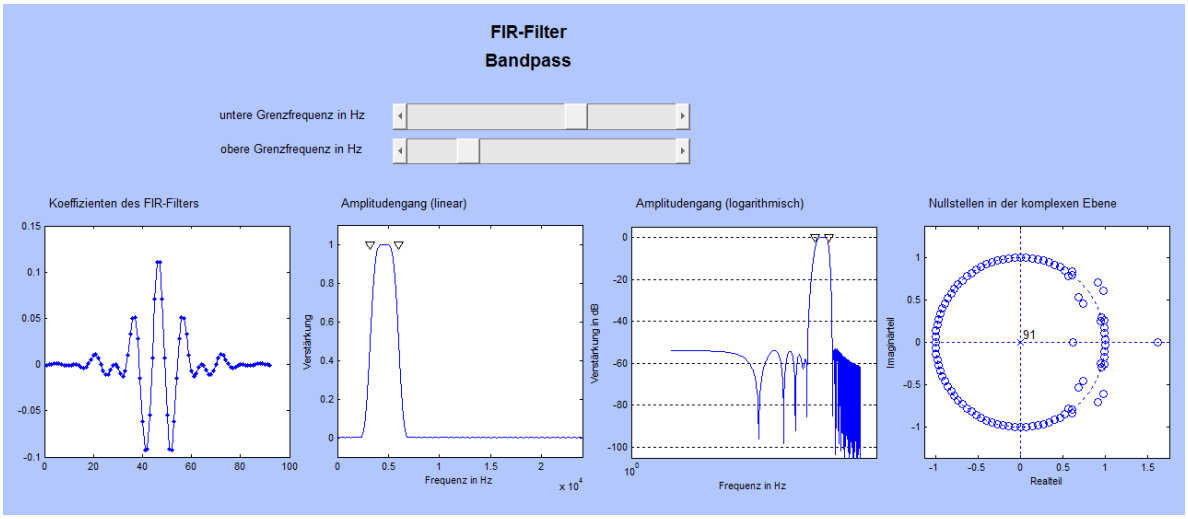


Abbildung G.1.: MATLAB-Programm zum Thema FIR-Filter (Bandpass)

quenz Werte zwischen 5001 und 10000 Hz verfügbar. Beim Start des Programms ist die untere Grenzfrequenz auf 4000 Hz und die obere Grenzfrequenz auf 6000 Hz eingestellt.

Um dieses Programm ausführen zu können, müssen die Dateien `fir_bandpass.m` und `fir_bandpass.fig` in ein Verzeichnis gespeichert werden, welches dann in MATLAB als Current Folder bzw. Current Directory ausgewählt wird. Das Programm lässt sich dann durch die Eingabe von >> `fir_bandpass` in das Command Window öffnen.

Stellen Sie mithilfe der Schieberegler verschiedene Grenzfrequenzen ein und beobachten Sie dabei den Amplitudengang und die Lage der Nullstellen der von Ihnen erzeugten Filter. Beantworten Sie folgende Fragen:

Frage 1

Wie können Sie die eingestellten Grenzfrequenzen aus den dargestellten Grafiken ablesen?

Frage 2

Welche Einstellung müssen Sie mittels der Schieberegler vornehmen, um keinen Bandpass, sondern einen reinen Tiefpass zu erzeugen?

H. Programmcode zur MATLAB-Anwendung: FIR-Filter

Der MATLAB-Programmcode des erstellten Programms zur Thematik der FIR-Filter befindet sich auf den folgenden Seiten.

```
function varargout = fir_bandpass(varargin)
% FIR_BANDPASS MATLAB code for fir_bandpass.fig
%      FIR_BANDPASS, by itself, creates a new FIR_BANDPASS
   or raises the existing
%      singleton*.
%
%      H = FIR_BANDPASS returns the handle to a new
   FIR_BANDPASS or the handle to
%      the existing singleton*.
%
%      FIR_BANDPASS('CALLBACK',hObject,eventData,handles
   ,...) calls the local
%      function named CALLBACK in FIR_BANDPASS.M with the
   given input arguments.
%
%      FIR_BANDPASS('Property','Value',...) creates a new
   FIR_BANDPASS or raises the
%      existing singleton*.  Starting from the left,
   property value pairs are
%      applied to the GUI before fir_bandpass_OpeningFcn
   gets called.  An
%      unrecognized property name or invalid value makes
   property application
```

```
%      stop.  All inputs are passed to
   fir_bandpass_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
   allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
   fir_bandpass

% Last Modified by GUIDE v2.5 19-Sep-2013 05:42:00

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                      @fir_bandpass_OpeningFcn, ...
                   'gui_OutputFcn',  @fir_bandpass_OutputFcn
                      , ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
       {:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before fir_bandpass is made visible.
function fir_bandpass_OpeningFcn(hObject, ~, handles,
   varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see
   GUIDATA)
% varargin   command line arguments to fir_bandpass (see
   VARARGIN)

set(handles.slider1,'Value', 4000);
set(handles.slider3,'Value', 6000);

% Ordnung n festlegen:
n = 91; % Anzahl der Nullstellen in der komplexen Ebene
fa = 48000; % Abtastfrequenz in Hz muss passend zur
   Signalverarbeitungs-Hardware gewählt werden
fn = fa/2; % Nyquistfrequenz

% Bandpassentwurf:
bp1 = 4000; % -6 dB { Grenzfrequenz in Hz
bp2 = 6000; % -6 dB { Grenzfrequenz in Hz
Wn = [bp1/fn bp2/fn];
FIRkoeff = fir1(n, Wn, 'bandpass'); % Bandpass-Filter

% Darstellung der Filterkoeffizienten:
axes(handles.axes1);
plot(FIRkoeff, '-')
hold on;
plot(FIRkoeff, '.')
hold off
```

```
% Zuerst einen Vektor für die Koeffizienten des Nenners
   erstellen für Rechnung
% mit freqz, grpdelay, zplane:
a = zeros(size(FIRkoeff));
a(1) = 1;
% freqz über Aufruf mit Koeffizientenvektoren
cntW = 4096;
% Ohne zusätzliche Parameterangabe (cntW) würden nur 512
   Frequenzpunkte
% berechnet.
[Hfir, Wfir] = freqz(FIRkoeff, a, cntW); % mit 0 <= Wfir <=
   pi
% Falls Auflösung angezeigt werden sollte:
% df = Wfir(2)/pi*fn; % oder: df = fn/length(Wfir);
% df (delta_f) ist der Abstand zwischen den Frequenzpunkten
   in Hz
% d.h. df entspricht der Frequenzauflösung in Hz
amplFIR = abs(Hfir);

% Darstellung des Amplitudengangs (linear):
axes(handles.axes2);
plot(Wfir/pi*fn, amplFIR, 'b');
axis([0 fn -0.1 1.1]);
ylabel('Verstärkung')
xlabel('Frequenz␣in␣Hz')
hold on;
plot(Wn*fn,1,'kv')
hold off;

% Darstellung des Amplitudengangs (logarithmisch):
axes(handles.axes3);
semilogx(Wfir/pi*fn, 20*log10(amplFIR), 'b');
axis([1 fa -105 5]); % 1 Hz bis fa, -105 dB bis +5 dB
ylabel('Verstärkung␣in␣dB')
```

```
xlabel('Frequenz in Hz')
grid
hold on;
plot(Wn*fn,0,'kv')
hold off;

% Darstellung der Nullstellen in der z-Ebene:
axes(handles.axes4);
zplane(FIRkoeff, a)
ylabel('Imaginärteil')
xlabel('Realteil')

% Erzeugen der Koeffizientendatei:
fid = fopen('FIRkoeff.h', 'w'); % File-ID
fprintf(fid,'const float FIRkoeff[] = {\n');
fprintf(fid,'%1.7e,\n',FIRkoeff(1:end-1));
fprintf(fid,'%1.7e};\n',FIRkoeff(end));
fclose(fid);

% Choose default command line output for fir_bandpass
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes fir_bandpass wait for user response (see
   UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command
    line.
function varargout = fir_bandpass_OutputFcn(~, ~, handles)
% varargout  cell array for returning output args (see
   VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see
   GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on slider movement.
function slider1_Callback(~, ~, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see
   GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to
   determine range of slider

% Ordnung n festlegen:
n = 91; % Anzahl der Nullstellen in der kompl. Ebene
fa = 48000; % Abtastfrequenz in Hz passend zur
   Signalverarbeitungs-Hardware wählen
fn = fa/2; % Nyquistfrequenz

% Bandpassentwurf:
bp1 = get(handles.slider1,'Value'); % -6 dB { Grenzfrequenz
   in Hz
bp2 = get(handles.slider3,'Value'); % -6 dB { Grenzfrequenz
   in Hz
Wn = [bp1/fn bp2/fn];
FIRkoeff = fir1(n, Wn, 'bandpass'); % Bandpass-Filter
```

```
% Darstellung der Filterkoeffizienten:
axes(handles.axes1);
plot(FIRkoeff, '-')
hold on;
plot(FIRkoeff, '.')
hold off

% Zuerst einen Vektor für die Koeffizienten des Nenners
   erstellen für Rechnung
% mit freqz, grpdelay, zplane:
a = zeros(size(FIRkoeff));
a(1) = 1;
% freqz über Aufruf mit Koeffizientenvektoren
cntW = 4096;
% Ohne zusätzliche Parameterangabe (cntW) würden nur 512
   Frequenzpunkte
% berechnet.
[Hfir, Wfir] = freqz(FIRkoeff, a, cntW); % mit 0 <= Wfir <=
   pi
% Falls Auflösung angezeigt werden sollte:
% df = Wfir(2)/pi*fn; % oder: df = fn/length(Wfir);
% df (delta_f) ist der Abstand zwischen den Frequenzpunkten
   in Hz
% d.h. df entspricht der Frequenzauflösung in Hz
amplFIR = abs(Hfir);

% Darstellung des Amplitudengangs (linear):
axes(handles.axes2);
plot(Wfir/pi*fn, amplFIR, 'b');
axis([0 fn -0.1 1.1]);
ylabel('Verstärkung')
xlabel('Frequenz␣in␣Hz')
hold on;
plot(Wn*fn,1,'kv')
hold off;
```

```
% Darstellung des Amplitudengangs (logarithmisch):
axes(handles.axes3);
semilogx(Wfir/pi*fn, 20*log10(amplFIR), 'b');
axis([1 fa -105 5]); % 1 Hz bis fa, -105 dB bis +5 dB
ylabel('Verstärkung in dB')
xlabel('Frequenz in Hz')
grid
hold on;
plot(Wn*fn,0,'kv')
hold off;

% Darstellung der Nullstellen in der z-Ebene:
axes(handles.axes4);
zplane(FIRkoeff, a)
ylabel('Imaginärteil')
xlabel('Realteil')

% Erzeugen der Koeffizientendatei:
fid = fopen('FIRkoeff.h', 'w'); % File-ID
fprintf(fid,'const float FIRkoeff[] ={\n');
fprintf(fid,'%1.7e,\n',FIRkoeff(1:end-1));
fprintf(fid,'%1.7e};\n',FIRkoeff(end));
fclose(fid);

% --- Executes during object creation, after setting all
   properties.
function slider1_CreateFcn(hObject, ~, ~)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
   CreateFcns called
```

```
% Hint: slider controls usually have a light gray background
    .
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider3_Callback(~, ~, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
     MATLAB
% handles    structure with handles and user data (see
    GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to
    determine range of slider

% Ordnung n festlegen:
n = 91; % Anzahl der Nullstellen in der kompl. Ebene
fa = 48000; % Abtastfrequenz in Hz passend zur
    Signalverarbeitungs-Hardware wählen
fn = fa/2; % Nyquistfrequenz

% Bandpassentwurf:
bp1 = get(handles.slider1,'Value'); % -6 dB { Grenzfrequenz
    in Hz
bp2 = get(handles.slider3,'Value'); % -6 dB { Grenzfrequenz
    in Hz
Wn = [bp1/fn bp2/fn];
FIRkoeff = fir1(n, Wn, 'bandpass'); % Bandpass-Filter

% Darstellung der Filterkoeffizienten:
```

```
axes(handles.axes1);
plot(FIRkoeff, '-')
hold on;
plot(FIRkoeff, '.')
hold off

% Zuerst einen Vektor für die Koeffizienten des Nenners
   erstellen für Rechnung
% mit freqz, grpdelay, zplane:
a = zeros(size(FIRkoeff));
a(1) = 1;
% freqz über Aufruf mit Koeffizientenvektoren
cntW = 4096;
% Ohne zusätzliche Parameterangabe (cntW) würden nur 512
   Frequenzpunkte
% berechnet.
[Hfir, Wfir] = freqz(FIRkoeff, a, cntW); % mit 0 <= Wfir <=
   pi
% Falls Auflösung angezeigt werden sollte:
% df = Wfir(2)/pi*fn; % oder: df = fn/length(Wfir);
% df (delta_f) ist der Abstand zwischen den Frequenzpunkten
   in Hz
% d.h. df entspricht der Frequenzauflösung in Hz
amplFIR = abs(Hfir);

% Darstellung des Amplitudengangs (linear):
axes(handles.axes2);
plot(Wfir/pi*fn, amplFIR, 'b');
axis([0 fn -0.1 1.1]);
ylabel('Verstärkung')
xlabel('Frequenz␣in␣Hz')
hold on;
plot(Wn*fn,1,'kv')
hold off;
```

```
% Darstellung des Amplitudengangs (logarithmisch):
axes(handles.axes3);
semilogx(Wfir/pi*fn, 20*log10(amplFIR), 'b');
axis([1 fa -105 5]); % 1 Hz bis fa, -105 dB bis +5 dB
ylabel('Verstärkung in dB')
xlabel('Frequenz in Hz')
grid
hold on;
plot(Wn*fn,0,'kv')
hold off;

% Darstellung der Nullstellen in der z-Ebene:
axes(handles.axes4);
zplane(FIRkoeff, a)
ylabel('Imaginärteil')
xlabel('Realteil')

% Erzeugen der Koeffizientendatei:
fid = fopen('FIRkoeff.h', 'w'); % File-ID
fprintf(fid,'const float FIRkoeff[] = {\n');
fprintf(fid,'%1.7e,\n',FIRkoeff(1:end-1));
fprintf(fid,'%1.7e};\n',FIRkoeff(end));
fclose(fid);

% --- Executes during object creation, after setting all
   properties.
function slider3_CreateFcn(hObject, ~, ~)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
   CreateFcns called

% Hint: slider controls usually have a light gray background
   .
```

```
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

I. Skript zur MATLAB-Anwendung: Beugung an der Kreisblende

Das Skript, welches den Studierenden zur Vertiefung der Thematik der Beugung an der Kreisblende zur Verfügung gestellt werden soll, befindet sich auf den folgenden Seiten.
Da dieses Skript lediglich dem Verständnis der Studierenden dienen soll und eine durchgehende Nummerierung der Gleichungen für ein solches Verständnis, wie auch für die übersichtliche Darstellung der Inhalte, nicht notwendig ist, wird darauf verzichtet.

Beugung an der Kreisblende

1. Beugung

Wird eine Öffnung innerhalb eines ansonsten undurchsichtigen Mediums mit kohärentem Licht bestrahlt, so ergibt sich in einem bestimmten Abstand hinter der Öffnung eine Beugungsstruktur, die vom Radius bzw. der Breite der Öffnung und vom Abstand zwischen Beobachtungsebene und Blende abhängt. In der Ebene des Beugungsmusters kann dann die Lichtintensitätsverteilung des Beugungsmusters beobachtet werden.

2. Zusammenhang zwischen Beugung und Fourier-Transformation

Mithilfe der Fourier-Transformation kann die Beugung an beliebig geformten Öffnungen allgemein beschrieben werden. Dies wird nachfolgend dargestellt.

2.1. Allgemeiner Fall

Sie $f(x)$ eine beliebige quadratintegrable Funktion, so wird die Fouriertransformierte zu f(x) folgendermaßen beschrieben:
$F(u) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) \cdot e^{iux} dx.$

Um f(x) aus F(u) zu bestimmen, wird die obige Gleichung mit $e^{i2\pi ux'}$ multipliziert. Anschließend werden beide Seiten der Gleichung über die Variable u integriert. Wird anschließend x' in x umbenannt, so ergibt sich:
$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} F(u) \cdot e^{-iux} du.$

Die Funktionen $f(x)$ und $F(u)$ werden dann als Fourierpaar, die Variablen x und u als fourierkonjugierte Variablen bezeichnet.

2.2. Anwendung auf die Beugung

Nachfolgend wird der allgemeine Fall behandelt, dass auf eine Fläche σ in der Ebene $z = 0$ mit der Transmission $\tau(x, y)$ eine Lichtwelle mit der Feldstärkeverteilung $E_e(x, y)$ fällt. Für eine Blende gilt dabei innerhalb der Blendenöffnung $\tau(x, y) = 1$ und außerhalb der Blendenöffnung $\tau(x, y) = 0$. Direkt hinter der Fläche gilt:
$E(x, y) = \tau(x, y) \cdot E_e(x, y)$

Die Amplitudenverteilung $E(x', y', z_0)$ in der Ebene $z = z_0$ kann aus dem folgenden Beugungsintegral berechnet werden, wobei A die Amplitude angibt:
$E(x', y', z_0) = A(x', y', z_0) \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} E_S(x, y) \cdot exp\left[\frac{+ik}{z_0}(x'x + y'y)\right] dxdy$

Nun setzt man die obige Gleichung für $E(x, y)$ in das Beugungsintegral ein und vergleicht das Ergebnis mit der folgenden zweidimensionalen Fourier-Transformation:
$F(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) \cdot e^{-i2\pi(ux+vy)} dxdy$
$f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F(u, v) \cdot e^{i2\pi(ux+vy)} dudv$

Setzt man hierbei $u = \frac{x'}{\lambda z_0}$ und $v = \frac{y'}{\lambda z_0}$, so ergibt sich:
$f(x, y) = E(x, y) = \tau(x, y) \cdot E_e(x, y)$

Diese Formel stellt die Amplitudenverteilung direkt nach der Beugungsebene $z = 0$ dar. Die Feldstärkeverteilung $E(x', y')$ der Beugungsstruktur in der Beobachtungsebene $z = z_0$ ist nach dem Beugungsintegral:
$E(x', y') = A(x', y', z_0) \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} E_e(x, y) \cdot \tau(x, y) \cdot e^{\frac{-i2\pi(x'x+y'y)}{\lambda z_0}} dxdy.$

Der Vergleich mit der Formel für die zweidimensionale Fourier-Transformation liefert dann:
$E(x', y', z_0) = F(u, v) \cdot A(x', y', z_0).$

Die Amplitudenverteilung des Beugungsbildes in der Ebene $z = z_0$ ist also proportional zur Fouriertransformierten $F(x', y')$ der Funktion $f(x, y) = \tau(x, y) \cdot E_e(x, y)$, wobei $\tau(x, y)$ die Transmissionsfunktion ist. Die Intensitätsverteilung $I(x', y')$ in der Beobachterebene ist dann:
$I(x', y') \propto |E(x', y')|^2 = |F(x', y')|^2,$

da $|A(x', y')|^2 = 1$ ist. [27]

3. Umsetzung durch Fast Fourier Transformation (FFT)

Soll ein Computerprogramm geschrieben werden, für welches die Durchführung einer Fourier-Transformation notwendig ist, so bietet es sich an, die „herkömmliche“ diskrete Fouriertransformation durch eine schnellere Variante, die sogenannte Fast Fourier Transformation (FFT) zu ersetzen. Diese hat gegenüber der DFT den Vorteil, dass damit Rechenzeit gespart werden kann und das Programm somit schneller läuft.

In numerischen Anwendungen werden die Integrale über Funktionen, wie sie bei der Fourier-Transformation notwendig sind, durch geeignet gewichtete Summen über die Funktionswerte an sogenannten Stützstellen genähert. Damit kann man die Fourier-Koeffizientenaus entsprechenden Funktionswerten an bestimmten Punkten bestimmen. Dabei ermöglicht die Auswahl von n äquidistanten Stützstellen die Berechnung von n Fourier-Koeffizienten.
Bei Anwendung der DFT müsste man nun für jeden der n Koeffizienten jeweils n-mal den Cosinus und den Sinus berechnen, was einen erheblichen Rechenaufwand bedeutet. Bei der FFT hingegen ist es möglich, diese Rechnung auf Polynome zurückzuführen. Wird dabei als Wert von n eine Potenz von 2 gewählt, so kann die entsprechende Rechnung auf zwei Teilpolynome zurückgeführt werden, die jeweils aus $\frac{n}{2}$ Termen bestehen. Weiterhin nutzt die FFT zuvor berechnete Zwischenergebnisse und spart somit arithmetische Rechenoperationen ein. [28]

Die FFT ist beispielsweise in MATLAB sehr einfach durchzuführen, da sie als vordefinierte Funktion vorhanden ist. So gibt die Eingabe `Y = fft(X)` die Fouriertransformierte von X an, wobei X eindimensional ist. Soll eine zweidimensionale Fourier-Transformation durchgeführt werden, so ist `U = fft2(V)` einzugeben, wobei U und V zweidimensional sind. Diese Funktion wird beispielsweise im untenstehenden Anwendungsbeispiel zur Kreisblende genutzt.

4. Anwendungsbeispiel: Kreisblende

Als Anwendungsbeispiel wird hier ein MATLAB-Programm zur Beugung an der Kreisblende vorgestellt (siehe Abb. I.1). Dieses verdeutlicht, wie sich das Beugungsmuster ganz allgemein in Abhängigkeit vom Radius der Kreisblende ändert. Dazu wird zunächst die Kreisblende an sich dargestellt. Anschließend wird das Beugungsmuster gezeigt. Hierbei kann neben der Beugungsstruktur an sich auch das Größenverhältnis der Struktur, beispielsweise der Maxima, die im Beugungsbild ersichtlich sind, mit der Größe der Blende verglichen werden. Schließlich wird das Beugungsmuster in logarithmierter Form dargestellt. Dies bietet den Vorteil, dass

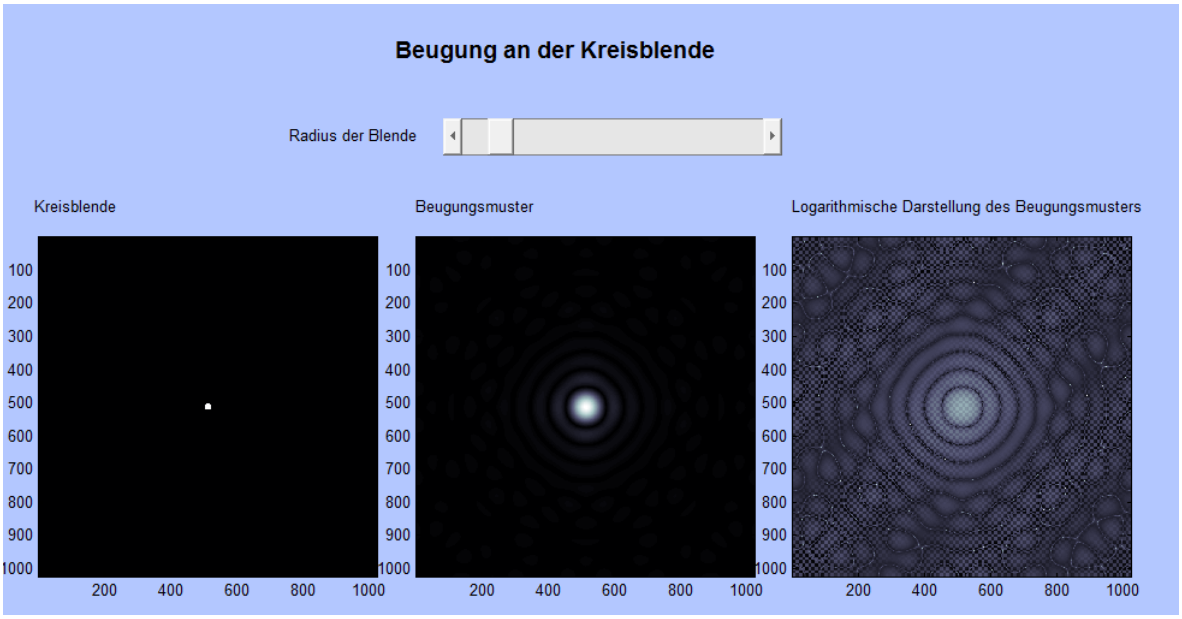


Abbildung I.1.: MATLAB-Programm zum Thema Beugung an der Kreisblende

nun in den Regionen, in denen die Amplituden klein sind, Details sichtbar werden.

Um dieses Programm ausführen zu können, müssen die Dateien `beugung_kreisblende.m` und `beugung_kreisblende.fig` in ein Verzeichnis gespeichert werden, welches dann in MATLAB als Current Folder bzw. Current Directory ausgewählt wird. Das Programm lässt sich dann durch die Eingabe von `>> beugung_kreisblende` in das Command Window öffnen.

Beantworten Sie zum gegebenen Sachverhalt folgende Fragen:

Frage 1
Wie verändert sich die Größe des Airy-Scheibchens im Verhältnis zur Größe der Blendenöffnung?

Frage 2
Wie verändert sich die Anzahl der sichtbaren konzentrischen Intensitätsmaxima im Verhältnis zur Größe der Blendenöffnung?

J. Programmcode zur MATLAB-Anwendung: Beugung an der Kreisblende

Der MATLAB-Programmcode des erstellten Programms zur Thematik der Beugung an der Kreisblende befindet sich auf den folgenden Seiten.

```
function varargout = beugung_kreisblende(varargin)
% BEUGUNG_KREISBLENDE MATLAB code for beugung_kreisblende.
   fig
%      BEUGUNG_KREISBLENDE, by itself, creates a new
   BEUGUNG_KREISBLENDE or raises the existing
%      singleton*.
%
%      H = BEUGUNG_KREISBLENDE returns the handle to a new
   BEUGUNG_KREISBLENDE or the handle to
%      the existing singleton*.
%
%      BEUGUNG_KREISBLENDE('CALLBACK',hObject,eventData,
   handles,...) calls the local
%      function named CALLBACK in BEUGUNG_KREISBLENDE.M with
    the given input arguments.
%
%      BEUGUNG_KREISBLENDE('Property','Value',...) creates a
    new BEUGUNG_KREISBLENDE or raises the
%      existing singleton*.  Starting from the left,
   property value pairs are
%      applied to the GUI before
   beugung_kreisblende_OpeningFcn gets called.  An
```

```
%      unrecognized property name or invalid value makes
   property application
%      stop.  All inputs are passed to
   beugung_kreisblende_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
   allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
   beugung_kreisblende

% Last Modified by GUIDE v2.5 17-Sep-2013 22:52:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                      @beugung_kreisblende_OpeningFcn, ...
                   'gui_OutputFcn',
                      @beugung_kreisblende_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
       {:});
else
    gui_mainfcn(gui_State, varargin{:});
```

```
end
% End initialization code - DO NOT EDIT


% --- Executes just before beugung_kreisblende is made
   visible.
function beugung_kreisblende_OpeningFcn(hObject, ~, handles,
    varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see
   GUIDATA)
% varargin   command line arguments to beugung_kreisblende (
   see VARARGIN)

set(handles.slider1,'Value', 10);

n = 2^10;
M = zeros(n);

I = 1:n;
x = I-n/2;
y = n/2-I;
[X,Y] = meshgrid(x,y);
R = 10;
A = (X.^2 + Y.^2 <= R^2);
M(A) = 1;

axes(handles.axes1)
imagesc(M)
colormap(bone);
axis image
```

```
D1 = fft2(M);
D2 = fftshift(D1);

axes(handles.axes2)
imagesc(abs(D2))
axis image
colormap(bone)

D3 = log2(D2);

axes(handles.axes3)
imagesc(abs(D3))
axis image
colormap(bone)

% Choose default command line output for beugung_kreisblende
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes beugung_kreisblende wait for user response (
   see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command
    line.
function varargout = beugung_kreisblende_OutputFcn(~, ~,
   handles)
% varargout  cell array for returning output args (see
   VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
    MATLAB
```

```
% handles    structure with handles and user data (see
   GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on slider movement.
function slider1_Callback(~, ~, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see
   GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to
   determine range of slider

n = 2^10;
M = zeros(n);

I = 1:n;
x = I-n/2;
y = n/2-I;
[X,Y] = meshgrid(x,y);
R = get(handles.slider1,'Value');
A = (X.^2 + Y.^2 <= R^2);
M(A) = 1;

axes(handles.axes1)
imagesc(M)
colormap(bone)
axis image
```

```
D1 = fft2(M);
D2 = fftshift(D1);

axes(handles.axes2)
imagesc(abs(D2))
axis image
colormap(bone)

D3 = log2(D2);

axes(handles.axes3)
imagesc(abs(D3))
axis image
colormap(bone)

% --- Executes during object creation, after setting all
   properties.
function slider1_CreateFcn(hObject, ~, ~)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
   CreateFcns called

% Hint: slider controls usually have a light gray background
   .
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```