

**Fakultät für Technik
Bereich Informationstechnik**

Labor Rechnergestützte Mathematik

Versuch 1

MATLAB / OCTAVE Grundlagen

Computerarithmetik

Lineare Algebra

Laboranleitung

Prof. Dr.-Ing. Stefan Hillenbrand
Yvonne Beck, M.Sc.

Version 2020.1

Inhaltsverzeichnis

1	Einleitung	4
2	Wichtige Hinweise zur Versuchsvorbereitung	5
2.1	Zeitmanagement.....	5
2.2	Mathematik 1 und Grundlagen der Softwareentwicklung sind grundlegend	5
2.3	Arbeiten Sie mit der Versuchsanleitung.....	5
2.4	Bearbeiten Sie die Vorbereitungsaufgaben am Rechner	6
2.5	Nutzen Sie Literatur	7
2.6	Fragen	7
2.7	Ihre Vorbereitung wird überprüft	7
3	Versuchsdurchführung.....	8
3.1	VOR dem Labor	8
3.1	WÄHREND des Labors	8
3.2	NACH dem Labor.....	8
4	Erste Schritte in MATLAB/Octave.....	9
4.1	Installation und Start von MATLAB	9
4.1.1	Installation von MATLAB	9
4.1.2	Start von MATLAB	10
4.1.3	MATLAB Toolboxes.....	11
4.2	MATLAB als Taschenrechner.....	12
4.2.1	Der erste Befehl	12
4.2.2	Variablen	13
4.3	Skripte.....	15
4.3.1	Einrichten des Editors.....	15
4.3.2	Beispiel: Umfang und Fläche	16
4.3.3	Strings und formatierte Ausgabe der Ergebnisse	18
4.4	Komplexe Zahlen.....	20
5	Matrizen und Vektoren.....	22
5.1	Matrizen.....	23
5.2	Vektoren	25
5.3	Spezielle Vektoren und Matrizen	26
5.4	Arrays.....	27
5.4.1	Anlegen von Arrays	27
5.4.2	Array mit Zahlenfolge.....	27
5.4.3	Zusammenfassen von Arrays.....	28
5.4.4	Arrays aus Daten	29
5.4.5	Zugriff auf eindimensionale Arrays	31
5.4.6	Zugriff auf zweidimensionale Arrays	32
5.4.7	Elementweise Rechenoperationen	34

6	Grafiken und Funktionen	38
6.1	Darstellen von Daten in Grafiken	38
6.2	Schaubilder von Funktionen	41
6.3	Funktionen	42
7	Versuche zur Computerarithmetik	46
7.1	Addition, Subtraktion und die Reihenfolge	46
7.2	Rundungsfehler – auch ohne große oder kleine Zahlen	47
7.3	IEEE-Fließkommazahlen	49
7.3.1	Fließkommazahl in Hexadezimaldarstellung	49
7.3.2	Hexadezimaldarstellung in Fließkommazahl	50
7.3.3	Bequemes Umwandeln von 32 Bit IEEE-Fließkommazahlen	50
8	Lineare Algebra	51
8.1	Lösung linearer Gleichungssysteme	51
8.1.1	Beispiel aus Mathematik 1	51
8.1.2	Lösung der Matrix-Vektor-Gleichung	52
8.2	Beispiel: Widerstandsnetzwerk	54
8.2.1	Aufgabenstellung	54
8.2.2	Herleitung der Gleichungen	54
8.2.3	Ermittlung des optimalen Widerstandswertes	55
8.2.4	Numerische Optimierung	57
9	Funktionen in der Anwendung	58
	Literatur	62

1 Einleitung

Denken Sie zu Beginn kurz darüber nach, wofür Sie Ihren Laptop am häufigsten verwenden. Welche Apps nutzen Sie auf Ihrem Smartphone am häufigsten? Vermutlich denken Sie dabei an viele unterschiedliche Anwendungen wie zum Beispiel die Suche nach Informationen im Internet, das Einkaufen in Online-Shops oder die Kommunikation mit Freunden mithilfe von Email, WhatsApp oder Facebook. Auch die Entspannung bei guter Musik, einem spannenden Film oder einem Computerspiel darf in der Aufzählung nicht fehlen. Darüber hinaus werden Sie für das Studium – also für die Arbeit – ein Office-Paket verwenden, um zum Beispiel für die Projektarbeiten und für die Thesis Texte zu schreiben und Präsentationen zu erstellen.

Computer wurden ursprünglich erfunden, um mathematische Probleme zu lösen. Dies erkennen Sie auch heute noch daran, dass auf praktisch jedem Rechner – sei es PC oder Smartphone – eine Taschenrechner-App installiert ist und im Office-Paket eine Tabellenkalkulation¹ enthalten ist. Mit Microsoft EXCEL oder LibreOffice Calc ist es möglich, umfangreiche mathematische Berechnungen durchzuführen oder Messdaten zu plotten. Allerdings sind diese Programme mehr für kaufmännische und weniger für technische Anwendungen konzipiert und stoßen daher bei Aufgabenstellungen der Ingenieurmathematik schnell an Grenzen.²

In den Ingenieur- und Naturwissenschaften ist MATLAB das wohl am häufigsten eingesetzte Werkzeug für numerische Berechnungen, das Auswerten von Messungen, die Simulation technischer Systeme und viele weitere mathematische Aufgabenstellungen. Grundkenntnisse im Umgang mit MATLAB werden daher bei der Einstellung von Ingenieuren – sei es für das Praxissemester, die Thesis oder den Berufseinstieg – meist ebenso vorausgesetzt wie die Kenntnis einer Programmiersprache wie C/C++ oder der sichere Umgang mit einem Office-Paket.

Im Labor Rechnergestützte Mathematik werden Sie daher MATLAB³ bzw. die Open-Source-Alternative Octave⁴ kennenlernen. MATLAB wird in diesem Labor für numerische Berechnungen, das Erstellen von Plots, das Auswerten von Messdaten und die Simulation eines einfachen dynamischen Systems eingesetzt. Hierbei werden Sie viele der numerischen Verfahren aus der Vorlesung Rechnergestützte Mathematik selbst programmieren sowie die in MATLAB / Octave enthaltenen professionellen numerischen Algorithmen einsetzen.

¹ Die Tabellenkalkulation VisiCalc war 1983 die „Killerapplikation“ für den Apple II und damit für die Verbreitung von PCs auf den Schreibtischen.

² In Microsoft EXCEL können maximal $1.049 \cdot 10^6$ Zeilen und $1.6 \cdot 10^4$ Spalten angelegt werden – zur Arbeit mit großen Datenmengen stellt dies eine Einschränkung dar. (<https://support.office.com> -> Excel Specifications and limits, abgerufen am 21.01.2019)

³ de.mathworks.com

⁴ www.octave.org

2 Wichtige Hinweise zur Versuchsvorbereitung

2.1 Zeitmanagement

Das Erlernen einer mathematischen Programmiersprache wie MATLAB erfordert Zeit und Übung. Das Labor Rechnergestützte Mathematik ist in Ihrer Prüfungsordnung mit 1 ECTS bewertet, das heißt insgesamt müssen Sie mit einem Arbeitsaufwand von etwa 30 Stunden rechnen. Pro Laborversuch sind dies 10 Stunden, von denen 3 Stunden auf den eigentlichen Versuchstermin fallen. Da Sie keine Nachbereitung oder Dokumentation für die Versuche erstellen müssen, ist die übrige Zeit für die Vorbereitung vorgesehen.

Planen Sie daher für die Vorbereitung ca. 5 – 6 Stunden ein, am besten verteilt auf mehrere kürzere Einheiten! Bei Versuch 1 werden Sie sogar noch mehr Zeit benötigen, die Vorbereitung der Versuche 2 und 3 ist dafür weniger umfangreich. Wenn Einführungsveranstaltungen zum Umgang mit MATLAB bzw. Octave angeboten werden, können diese eine gute Ergänzung zum Selbststudium darstellen.

Tipp: Nutzen sie MATLAB auch für andere Veranstaltungen und Labore oder in ihrer Vorbereitung auf Klausuren.

2.2 Mathematik 1 und Grundlagen der Softwareentwicklung sind grundlegend

Die Laborversuche setzen voraus, dass Sie die mathematischen Grundlagen aus der Vorlesung Mathematik 1 sowie den Stoff aus Grundlagen der Softwareentwicklung beherrschen. Sollte dies nicht (mehr) der Fall sein, müssen Sie den entsprechenden Stoff mithilfe Ihrer Mitschrift oder geeigneter Literatur wiederholen.

Parallel zum Labor hören Sie im zweiten Semester die Vorlesungen Rechnergestützte Mathematik und Analysis 2. Die Laborversuche bauen auf den dort vermittelten Inhalten auf. Es lohnt sich daher umso mehr, „am Ball zu bleiben“ und die Nachbereitung der Vorlesungen nicht nur in die Klausurvorbereitungszeit zu schieben.

Sie können den jeweiligen Versuch nur bestehen, wenn Sie die in der Anleitung geforderten und bis dahin in den Vorlesungen vermittelten Grundlagen beherrschen!

2.3 Arbeiten Sie mit der Versuchsanleitung

Damit wir im ersten Versuch gemeinsam die ersten mathematischen Probleme mit MATLAB lösen können, müssen Sie sich zuvor anhand einfacher Beispiele mit den wesentlichen Grundlagen der MATLAB-Sprache, also den wichtigsten Befehlen, der Syntax und den Datentypen, vertraut machen.

Daher besteht Ihre Vorbereitung für den ersten Versuch im Wesentlichen aus der selbständigen Einarbeitung in die Grundlagen der MATLAB-Sprache. Um Sie dabei zu unterstützen ist

diese Anleitung sehr ausführlich und führt Sie anhand einfacher Beispiele und mithilfe von Übungsaufgaben Schritt für Schritt in die MATLAB-Sprache ein. Hierfür müssen Sie diese Anleitung nicht nur lesen, sondern damit arbeiten. **Drucken Sie hierfür die Anleitung aus⁵** und legen Sie diese bei der Vorbereitung neben den Computer:

- Tragen Sie Ihre Antworten direkt in die freien Stellen der Anleitung ein.
- Heben Sie wichtige Punkte im Text oder den Beispielen hervor.
- Tragen Sie Notizen und Anmerkungen direkt in die Anleitung ein.
- Wenn etwas nicht ganz klar geworden ist, markieren Sie die Stelle und notieren Sie die Frage in der Anleitung.

Zusätzlich zur Versuchsanleitung finden Sie auf der Laborseite die **MATLAB / Octave Kurzreferenz [4]**. Drucken Sie auch dieses kurze Dokument aus. Es ist sehr hilfreich, dieses beim Bearbeiten der Aufgaben neben dem Rechner liegen zu haben. Es kann im Labor als Hilfsmittel dienen.

2.4 Bearbeiten Sie die Vorbereitungsaufgaben am Rechner

Die MATLAB-Sprache lernt man – wie jede Programmiersprache – nicht alleine theoretisch durch Lesen, sondern es sind auch praktische Versuche am Rechner erforderlich. Viele der Vorbereitungsaufgaben müssen Sie daher am Rechner bearbeiten.

In den **PC-Pools der Fakultät für Technik** ist dafür auf allen Rechnern sowohl **MATLAB** als auch die Open-Source-Alternative **Octave** installiert. Gerade zu Beginn des Semesters sind die PC-Pools wenig durch Lehrveranstaltungen belegt, sodass Sie die Einarbeitung in die Grundlagen der MATLAB-Sprache gut dort durchführen können.

Wenn Sie die Vorbereitung lieber mit Ihrem **eigenen Rechner** durchführen möchten, können Sie über die Total Academic Headcount Lizenz, über welche die HSPF verfügt, Matlab nutzen (vgl. 4.1.1). Alternativ können Sie das kostenlose Open Source Programm **Octave** verwenden, das weitgehend zu MATLAB kompatibel ist.⁶

Da die MATLAB-Sprache eine Skriptsprache ist, können Sie die Befehle sehr einfach ausprobieren und direkt das Ergebnis sehen. Wenn etwas bei den Übungsaufgaben nicht auf Anhieb funktioniert, können Sie dies nutzen und solange mit den Befehlen „spielen“, bis das erwünschte Ergebnis berechnet wird.

Vergessen Sie nicht, die von Ihnen erstellten Dateien zu speichern und notieren Sie sich Fragen in der Anleitung.

⁵ Wenn Sie statt mit Papier mit Tablet-Computer schreiben, verwenden Sie eine Software, mit der Sie in der Anleitung markieren und notieren können.

⁶ Informationen zur Installation von Octave finden Sie auf der Kursseite.

2.5 Nutzen Sie Literatur

Diese Laboranleitung ist so ausführlich geschrieben, dass Sie nicht unbedingt weitere Literatur benötigen. Dennoch ist es sehr sinnvoll, zumindest für spezielle Fragen auch in die Literatur zu schauen:

- Die Erklärungen sind ausführlicher und es gibt zusätzliche Beispiele.
- Ein anderer Blickwinkel auf ein Problem kann helfen, dieses besser zu verstehen.
- Wenn Sie für Ihre Projekt- oder Abschlussarbeit oder im Beruf mehr Informationen benötigen, ist es gut zu wissen, wo man diese findet.

Alle in dieser Anleitung genannten Bücher sind online verfügbar! Hinweise zum Download finden Sie in der Literaturübersicht. Auch mit der MATLAB-Hilfe sollte man sich vertraut machen.

2.6 Fragen

Wenn Sie beim Durcharbeiten der Anleitung oder beim Lösen der Aufgaben Fragen haben, sollen diese natürlich beantwortet werden.

Wenn Sie die **Antwort auf eine Frage vor dem Labor** benötigen, zum Beispiel weil Sie Schwierigkeiten mit den Vorbereitungsaufgaben haben, haben Sie mehrere Möglichkeiten:

- Diskutieren Sie die Frage mit Ihren Kommilitonen und helfen Sie sich gegenseitig. Oft reicht ein kleiner Hinweis oder eine einfache Idee, um weiterzukommen. Am Ende dieses Dokuments finden Sie eine Übersicht über Literatur zum Thema.
- Nutzen Sie die Zeit vor bzw. nach der Vorlesung Rechnergestützte Mathematik oder die Sprechstunde, um direkt Ihre/n Dozent/In zu fragen.

Auch wenn Sie die Vorbereitung komplett verstanden haben, bleiben vielleicht Fragen offen. Stellen Sie diese Fragen im Labor, damit wir diese **gemeinsam diskutieren** können.

2.7 Ihre Vorbereitung wird überprüft

Der Laborversuch kann nur dann sinnvoll durchgeführt werden, wenn sich alle Teilnehmer zuvor mit den wesentlichen Grundlagen der MATLAB-Sprache vertraut gemacht haben.

Sie können daher den Laborversuch nur bestehen, wenn Sie die Vorbereitungsaufgaben bearbeitet haben. Details zur Überprüfung der Vorbereitung, z.B. in Form von Testaten, finden Sie im Moodle-Kurs zum Labor Ihres Studiengangs.

3 Versuchsdurchführung

3.1 VOR dem Labor

- **Arbeiten Sie diese Laboranleitung komplett gründlich durch.**
- **Mithilfe der Kapitel 4 – 6 arbeiten Sie sich selbständig in die Grundlagen der MATLAB-Sprache ein. Bearbeiten Sie dazu die zahlreichen Beispiele und Aufgaben. Diese Grundlagen sind Voraussetzung für das Labor.**
- Wiederholen Sie bei Bedarf die mathematischen Grundlagen.
- Notieren Sie alle Fragen, die während der Vorbereitung oder bei der Bearbeitung der Aufgaben aufgetaucht sind und klären Sie diese im Labor.

Die Vorbereitungsaufgaben sind im Skript gelb markiert. Ihre Bearbeitung ist Grundlage zum Bestehen des jeweiligen Versuchs. Bringen Sie die Vorbereitung (Matlab-Dateien und handschriftliche Rechnungen) ins Labor mit und informieren Sie sich im Moodle Kurs, ob Vorbereitungsaufgaben vorab einzureichen sind.

3.1 WÄHREND des Labors

- Starten Sie den Rechner, erstellen Sie einen Arbeitsordner auf der Festplatte (Verzeichnis USERDATA). Laden Sie die ZIP-Datei zum Labor von der Kursseite herunter und entpacken Sie darin enthaltenen Daten in den Arbeitsordner.
- Kopieren Sie die *.m-Files aus der Vorbereitung in Ihren Arbeitsordner.
- Beteiligen Sie sich an der Diskussion. Es gibt fast immer mehrere richtige Lösungen!
- Speichern Sie regelmäßig Ihre Ergebnisse. Verwenden Sie für verschiedene Aufgaben oder Lösungswege eigene Dateinamen.
- Machen Sie sich Notizen!

3.2 NACH dem Labor

- Sichern Sie Ihre Arbeitsergebnisse!
 - Die Festplatte des Laborrechners ist dafür nicht geeignet.
 - Sichern Sie die Daten auf Ihrem persönlichen Bereich des Netzlaufwerks oder auf einem USB-Stick.
- Stellen Sie sicher, dass Sie den in im Labor gelernten Stoff beherrschen. Hierfür kann es notwendig sein, für die Vorbereitung des nächsten Labors einen Teil der Versuche nochmals nachzuvollziehen.

Die Versuche bauen aufeinander auf, Sie müssen daher die in diesem Versuch erlernten Grundlagen in den nächsten Versuchen beherrschen!

4 Erste Schritte in MATLAB/Octave

MATLAB („MATrix LABoraty“) wurde ursprünglich als Werkzeug für die Lehre im Bereich der numerischen linearen Algebra (daher der Name) entwickelt, hat sich aber inzwischen zu einem Quasistandard für numerische Mathematik in Industrie und Forschung entwickelt. Besonders in der Großindustrie (z. B. Fahrzeugbau), die sich die hohen Lizenzkosten (ca. 2000€ pro MATLAB-Lizenz plus ca. 1000€ für jede Toolbox) leisten kann, ist MATLAB sehr weit verbreitet.

Mit Octave wurde eine auch für kommerzielle Anwendungen kostenlose Open Source Alternative entwickelt, die zu fast 100% kompatibel mit MATLAB ist und die Funktionalität zahlreicher MATLAB-Toolboxen in Form sogenannter Packages verfügbar macht. Alle im Labor Rechnergestützte Mathematik verwendeten Befehle, Skripte und Funktionen können sowohl mit MATLAB als auch mit Octave ausgeführt werden.

Im Labor wird mit MATLAB gearbeitet. Sie können alternativ zuhause auch Octave nutzen. Über eine Total Academic Headcount Lizenz steht Ihnen jedoch auch für das Studium eine eigene MATLAB-Lizenz zur Verfügung:

4.1 Installation und Start von MATLAB

4.1.1 Installation von MATLAB

Für das Labor Rechnergestützte Mathematik wird ein aktuelles Release von MATLAB verwendet, das auf den PC-Pools installiert ist. Für die Installation auf Ihrem eigenen Rechner informieren Sie sich auf

https://engineeringpf.hs-pforzheim.de/studium/im_studium/matlab/

bzw. direkt über

<https://de.mathworks.com/academia/tah-portal/hochschule-pforzheim-40677885.html>

über die Installation von MATLAB im Rahmen der Total Academic Headcount Lizenz der HSPF. Um die Installation durchzuführen, müssen Sie sich mit Ihrer Hochschul-E-Mailadresse einen MathWorks Account anlegen.⁷ Bei der Installation können Sie auswählen, welches Release installiert werden soll und welche Toolboxen Sie mit installieren möchten (vgl.

4.1.3).⁸ Die Installation müssen Sie mit Admin-Rechten ausführen.

⁷ Aktuell ist auf den Pools der HS Pforzheim das Release MATLAB R2018a installiert.

Für die Installation werden 2.6 GB HDD Speicher benötigt. Die aktuellen Matlab Releases (ab R2018b) unterstützen Windows 8.1. nicht. Mehr Informationen, auch zu anderen Betriebssystemen als Windows, finden sich unter <https://de.mathworks.com/support/sysreq.html>

⁸ Zum Lösen symbolischer mathematischer Gleichungen lohnt sich die Installation der Symbolic Math Toolbox, die ein Computeralgebrasystem bietet. SIMULINK wird in manchen Studiengängen in späteren Lehrveranstaltungen eingesetzt.

4.1.2 Start von MATLAB



Starten Sie MATLAB.

Es öffnet sich die in Abbildung 1 dargestellte Benutzeroberfläche (Graphical User Interface – GUI).

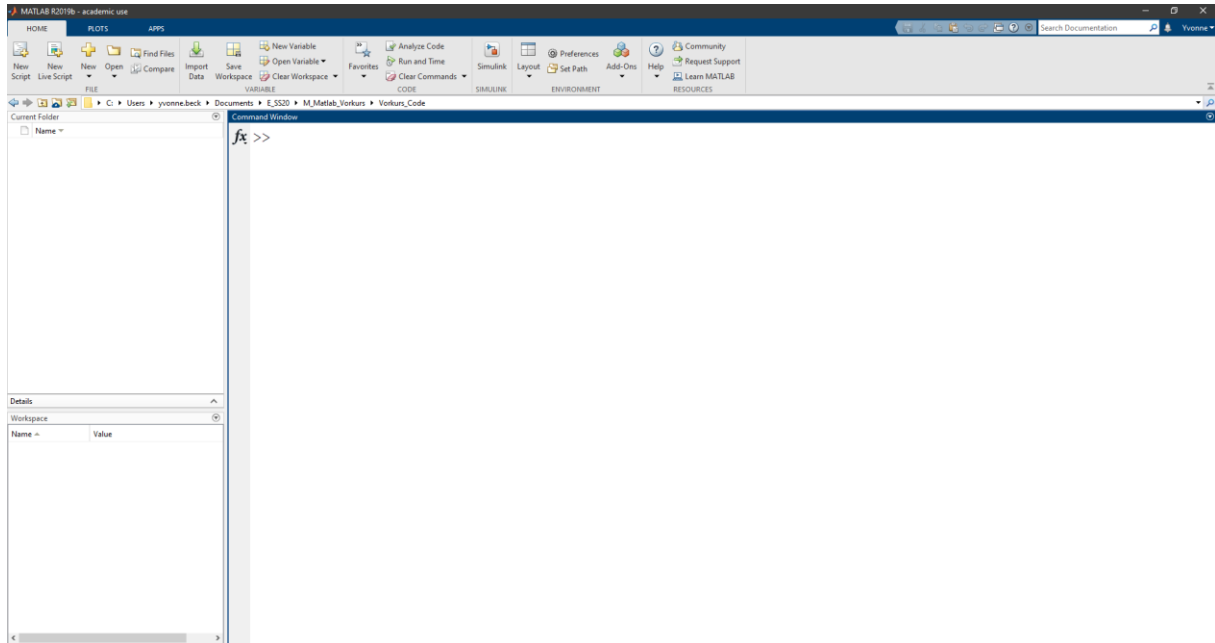





Abbildung 1: MATLAB GUI in der Standardansicht (MATLAB R2019b)

Die Benutzeroberfläche kann individuell angepasst werden. Änderungen im Layout können Sie über **HOME**-> **Environment** -> **Layout** vornehmen. Durch Auswahl von **Default** stellen Sie die Standardansicht aus Abbildung 1 her.⁹

Die Benutzeroberfläche setzt sich aus mehreren Fenstern zusammen, die Sie beliebig anordnen können. Die Standardfenster werden in Tabelle 1 kurz erläutert.

⁹ Wenn man mit mehreren Bildschirmen arbeitet, kann es praktisch sein, einzelne Fenster mithilfe der Schaltfläche in der oberen rechten Ecke „abzudocken“ und über Windows+ Pfeiltasten auf den Bildschirmen anzuordnen.

Tabelle 1: Fenster der MATLAB GUI [ergänzt durch Befehle für das command window]

Current Folder	<ul style="list-style-type: none"> In der Kopfzeile wird das aktuelle Verzeichnis angezeigt. Darunter werden wie im Explorer die Dateien des aktuellen Verzeichnisses aufgelistet.  ermöglicht den Wechsel in ein übergeordnetes Verzeichnis [<code>cd . .</code>]  ermöglicht die Auswahl eines neuen Ordners [<code>cd('IhrPfad')</code>]
Workspace	Anzeige der aktuell definierten Variablen
Command History	<p>Liste der letzten eingegebenen Befehle.</p> <p>Wenn Sie Home -> Layout -> Command History -> docked auswählen, erscheint eine Anzeige Befehlsverlauf am rechten Fensterrand. Sie kann durch Anklicken aufgeklappt werden.</p>
Command Window	Befehlsfenster ¹⁰ . Hier geben Sie die MATLAB-Befehle ein und erhalten die Ergebnisse zurück (wenn Sie die Ausgabe nicht durch ein Semikolon am Zeilenende unterdrücken).
Editor	<p>Editor zum Erstellen von Skripten und Funktionen. Der Editor wird durch Erstellen eines neuen *.m-Files Home-> New -> Script sichtbar. Es erscheint sich zudem ein Reiter EDITOR</p> 
Documentation	Mit diesem Feld am oberen rechten Rand des Fensters kann die Dokumentation durchsucht werden. Die Hilfe ist auch über die Befehle doc , help oder die F1 -Taste aufrufbar.

4.1.3 MATLAB Toolboxes

Die MATLAB-Sprache stellt eine große Zahl an Funktionen für die numerische Ingenieurmathematik zur Verfügung. Einige davon werden Sie im Laufe des Labors Rechnergestützte Mathematik kennenlernen.

Der Funktionsumfang kann mit Toolboxes (MATLAB) bzw. Paketen (Octave) erweitert werden, die umfangreiche Funktionen für verschiedene Anwendungsgebiete wie z. B. Signalverarbeitung oder Regelungstechnik bereitstellen. Im Labor Rechnergestützte Mathematik wird weitgehend ohne Funktionen aus MATLAB-Toolboxes gearbeitet. Wenn Sie dennoch solche Funktionen nutzen möchten, finden Sie dafür notwendigen Informationen in der MATLAB Dokumentation.¹¹

¹⁰ Auch Eingabezeile, engl. Prompt genannt

¹¹ <https://de.mathworks.com/help/matlab/>, bzw. über die Befehle **doc** oder **help**

4.2 MATLAB als Taschenrechner

4.2.1 Der erste Befehl

Die MATLAB-Sprache ist eine Skriptsprache. Das heißt, Sie können Befehle direkt ins Befehlsfenster (Command Window) eintippen und erhalten sofort das Ergebnis zurückgeliefert (Interpreter). Im Unterschied zur C-Programmierung mit Compiler können Sie damit alle Funktionen und Befehle direkt interaktiv ausprobieren, ohne dafür erst ein Programm schreiben und dieses vor dem Ausführen vom Compiler übersetzen lassen zu müssen. Nutzen Sie diese einfache interaktive Möglichkeit beim Kennenlernen der MATLAB-Sprache.

Als erstes Beispiel soll die folgende Rechnung durchgeführt werden:

$$3 \cdot \frac{23 + 14,7 - \frac{4}{6}}{3,5} + \sin\left(\frac{\pi}{3}\right)$$

Hierzu geben Sie den mathematischen Ausdruck hinter den beiden Pfeilen (**>>**) im Command Window¹² ein und bestätigen mit der Enter-Taste:

```
>> 3*(23 + 14.7 - 4/6)/3.5+sin(pi/3)
ans = 32.6089
```

Dieses einfache Beispiel hätten Sie auch problemlos mit Ihrem Taschenrechner berechnen können. Sie können daran jedoch einige wichtige Grundlagen der MATLAB-Sprache lernen:

- Wie in anderen Programmiersprachen auch wird als Dezimaltrennzeichen der Punkt und nicht das Komma verwendet.
- Die Kreiszahl π ist als Variable **pi** vordefiniert.
- Die Sinusfunktion **sin** erwartet den Winkel im Bogenmaß.
- Das Ergebnis der Rechnung wird der Standardvariablen **ans** (englisch für answer) zugewiesen. Die Variable wird im Befehlsfenster ausgegeben und im Fenster Workspace angezeigt:



Workspace	
Name ▲	Value
ans	32.6089

- Der Befehl wird in der Command History aufgelistet und kann von dort aus per drag-and drop in den Editor oder das Command Window gezogen werden.¹³

¹² Befehlsfenster in Octave. Im Folgenden wird jeweils der englische Ausdruck verwendet.

¹³ Die Anzeige der Command History ist in Tabelle 1 beschrieben.

Das Ergebnis der Rechnung wird mit 4 Nachkommastellen angezeigt. Es wäre jedoch falsch, daraus auf die Rechengenauigkeit von MATLAB / Octave zu schließen. Mit den folgenden Befehlen können Sie das Ergebnis mit deutlich mehr Nachkommastellen anzeigen lassen:

```
>> format long
>> ans
ans = 32.6088825466416
```

Es wurde keine neue Rechnung ausgeführt, sondern lediglich die Variable **ans** erneut ausgegeben. Der Befehl **format** beeinflusst daher nicht die Rechengenauigkeit, sondern nur – wie der Name sagt – die Formatierung der Ausgabe. Mit dem Befehl **format short** können Sie wieder auf die übersichtlichere Ausgabe mit weniger Nachkommastellen zurückschalten.

Solange Sie nicht explizit andere Datentypen definieren, erfolgen alle numerischen Berechnungen in MATLAB und Octave mit den aus der Vorlesung bekannten IEEE-Fließkommazahlen mit einer Länge von 64 Bit.

4.2.2 Variablen

Mit den folgenden drei Befehlen wird die Länge c der Hypotenuse eines rechtwinkligen Dreiecks mit den Katheten $a = 2$ und $b = 3$ berechnet¹⁴:

```
>> a = 2
a = 2
>> b = 3;
>> c = sqrt(a^2+b^2)
c = 3.6056
```

Folgendes lässt sich anhand dieser drei Eingaben beobachten:

- Variablen müssen in der MATLAB-Sprache nicht deklariert werden.
- Die Zuweisung eines Ausdrucks zu einer Variablen erfolgt mittels `=`.
- Das Semikolon `;` am Ende einer Zeile verhindert die Ausgabe des Ergebnisses.
- In der MATLAB-Sprache stehen die vom Taschenrechner bekannten mathematischen Funktionen zur Verfügung, wie hier die Wurzel- (**sqrt**) und Potenzfunktion (**^**). Weitere Funktionen finden Sie in der **MATLAB / Octave Kurzreferenz** [4] sowie in der MATLAB / Octave Hilfe. Auch wenn Sie MATLAB nicht auf Ihrem Rechner installiert haben, können Sie die MATLAB-Hilfe online nutzen [5].
- Wenn Sie einen Blick ins Fenster Workspace¹⁵ werfen, stellen Sie fest, dass auch die neu definierten Variablen **a**, **b** und **c** genau wie die Variable **ans** zur Klasse **double** gehören und damit IEEE-Fließkommazahlen mit der Länge 64 Bit sind.

¹⁴ Hinweis: Wir verzichten hier auf die Angabe physikalischer Maßeinheiten, die man in Form von Kommentaren ergänzen könnte – in späteren Laboren müssen Sie hierauf ggf. Rücksicht nehmen.

- Im Beispiel nicht zu sehen aber wichtig: MATLAB / Octave unterscheidet bei Variablennamen Groß- und Kleinschreibung. Valide Variablennamen beginnen mit einem Buchstaben, gefolgt von weiteren Buchstaben, Ziffern oder `_`. Leerzeichen, Minuszeichen, Umlaute, sowie führende Ziffern sind nicht zulässig. Auch bei Dateinamen sollten diese Regeln berücksichtigt werden.

VORSICHT

Achten Sie bei der Definition von Variablen darauf, keine Variablennamen zu verwenden, die bereits genutzt werden. So könnten Sie mit dem Befehl `pi = 4` die vordefinierte Kreiszahl π mit dem Wert 4 überschreiben, wie es in einem Gesetzesentwurf im amerikanischen Bundesstaat Indiana im Jahr 1897 gefordert wurde¹⁶. Alle folgenden Berechnungen mit `pi` liefern dann falsche Ergebnisse! Vermeiden Sie also das Überschreiben von Variablen- und Funktionsnamen, die bereits vordefiniert sind.

Aufgabe 1: Erste Rechenschritte

Löschen Sie mit dem Befehl `clear` die Variablen aus Ihren ersten Versuchen aus dem Workspace. Mit `clc` können Sie die Einträge des Command Windows leeren.

Führen Sie dann die folgenden Berechnungen durch und notieren Sie das Ergebnis. Bei Bedarf können Sie die **MATLAB Kurzreferenz [4]** zur Hilfe nehmen.

1. $\sin(60^\circ)$
2. $\cos\left(\frac{\pi}{3}\right)$
3. Gegeben ist das rechtwinklige Dreieck mit den Katheten $a = 3$ und $b = 4$. Berechnen Sie die folgenden Größen. Eine Skizze kann beim Lösen der Aufgabe helfen.
 - a. Länge der Hypotenuse c
 - b. Winkel zwischen den Katheten und der Hypotenuse
 - c. Flächeninhalt
4. Ermitteln Sie die Lösung der Gleichung $2 \cdot e^{5x} = 7$.
5. Ermitteln Sie die Lösung der Gleichung $2 \cdot 3^{5x} = 7$.

Hinweis zu 4. und 5.: Lösen Sie die Gleichungen zunächst von Hand nach x auf und nutzen Sie dann MATLAB / Octave wie einen Taschenrechner. Überprüfen Sie Ihr Ergebnis, indem Sie die berechnete Lösung in die gegebenen Gleichungen einsetzen.

¹⁵ Arbeitsumgebung in Octave

¹⁶ Mehr zur Indiana Pi Bill in Wikipedia: http://de.wikipedia.org/wiki/Indiana_Pi_Bill

4.3 Skripte

Die Eingabe der MATLAB / Octave-Befehle im Befehlsfenster ist praktisch für kurze Berechnungen. Sobald jedoch mehrere aufeinanderfolgende Befehle benötigt werden, wird diese Vorgehensweise schnell unpraktisch: ändert sich ein Wert in einem früheren Rechenschritt, müssen alle folgenden Befehle erneut eingegeben und ausgeführt werden.¹⁷

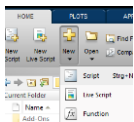
Um die Auswertung von Befehlen zu automatisieren, werden diese in eine Textdatei eingegeben. Die Datei wird mit der **Dateiendung** **.m** gespeichert. Das so erstellte Skript, auch „m-file“ genannt, kann dann in MATLAB und Octave automatisch ausgeführt werden. Skripte sind damit Programme in der MATLAB-Sprache. Im Unterschied zu C-Programmen müssen die Skripte jedoch nicht vor der Ausführung kompiliert werden, sondern MATLAB bzw. Octave arbeitet diese Befehl für Befehl ab.

Zum Erstellen eines Skripts kann im Prinzip jeder beliebige Texteditor verwendet werden. Es empfiehlt sich jedoch, den in der MATLAB- Entwicklungsumgebung enthaltenen Editor zu verwenden, da dieser einige für die Programmentwicklung hilfreiche Funktionen bietet:

- Wie aus anderen Entwicklungsumgebungen (z. B. Visual Studio) bekannt, werden die Skripte automatisch koloriert, um die Lesbarkeit zu verbessern.
- Skripte können direkt aus dem Editor gestartet werden.
- In den Editor ist ein Debugger für die Fehlersuche integriert.
- Mit Kommentaren, die über **% Kommentar** eingegeben werden, können Skripte nutzerfreundlich gestaltet werden. Blöcke, die mit **%% Block** abgetrennt werden, strukturieren die Darstellung und können zur Blockweisen Ausführung eines Skripts genutzt werden.
- Octave bzw. MATLAB-Skripte können mit **Strg+S** gespeichert werden.
- Kommentare, die zu Beginn eines Skripts bzw. einer Funktion stehen werden auch in der Kurzhilfe zur Datei angezeigt. Mehr Informationen entnehmen Sie dem MATLAB-Style Guide.

4.3.1 Einrichten des Editors

Öffnen Sie über **Home -> New Script** oder über die Tastenkombination **Strg+N** ein neues Skript.



Es erscheint ein Editor-Fenster. Sie sehen es nun oberhalb des Command Windows. Über **Home -> Environment -> Layout** können Sie die Ansicht des Fensters anpassen. Damit erhalten Sie die in Abbildung 2 gezeigte Oberfläche.

¹⁷ Ein Aufruf der letzten Befehle aus der **Command History** ist mithilfe der Pfeiltasten (nach oben / nach unten) möglich.

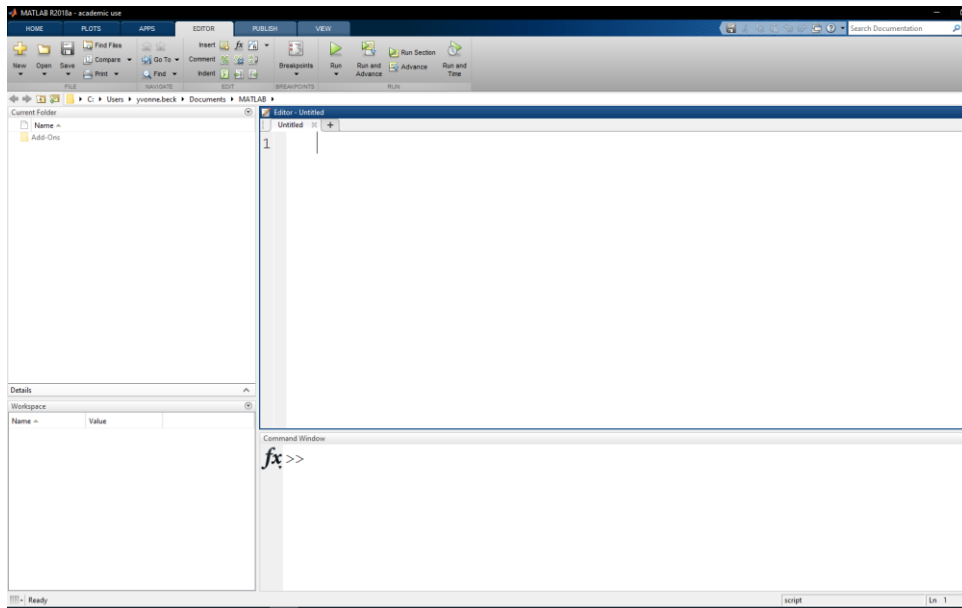


Abbildung 2: MATLAB-GUI mit Editor

Mit dem Editor-Fenster erscheint ein Reiter **Editor**, mit dem Skripte erstellt, editiert, debuggt werden können usw.



4.3.2 Beispiel: Umfang und Fläche

Es soll nun ein Skript erstellt werden, das ausgehend vom Radius r für die drei in Abbildung 3 gezeichneten Figuren jeweils Umfang und Fläche berechnet.

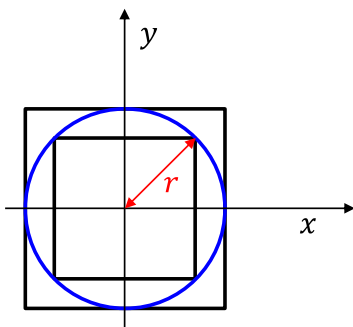



Abbildung 3: Kreis mit innerem und äußerem Quadrat

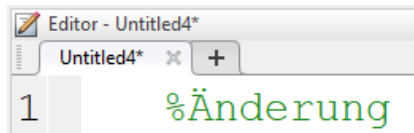
Die Formeln für die Berechnung können leicht hergeleitet werden und sind in **Tabelle 2** zusammengestellt.

Tabelle 2: Berechnung von Umfang und Fläche

	Umfang	Fläche
Kreis	$U_K = 2 \cdot \pi \cdot r$	$A_K = \pi \cdot r^2$
inneres Quadrat	$U_i = 4 \cdot \sqrt{2} \cdot r$	$A_i = 2 \cdot r^2$
äußeres Quadrat	$U_a = 8 \cdot r$	$A_a = 4 \cdot r^2$

Um für die Berechnung ein Skript zu erstellen führen Sie folgende Schritte aus:

- Öffnen Sie mit **Strg+N** oder über **Home -> New Script** ein neues Skript. Dies wird im Editor als Datei mit dem Namen **<Untitled>** angezeigt.
- Klicken Sie im Reiter **Editor** auf das Icon  oder geben Sie **Strg+S** ein, um die **Datei zu speichern**.
- Geben Sie im bekannten Speichern-Dialog den Dateiname **Flaeche** ein. Die Dateiendung **.m** wird automatisch hinzugefügt.
- Die Datei wird im aktuellen Verzeichnis (Anzeige in der Symbolleiste und im Dateibrowser) gespeichert. Achten Sie darauf, dass Sie ein Verzeichnis wählen, in dem Sie volle Schreib- und Leserechte haben.¹⁸
- Nach dem Klick auf **Speichern** wird das noch leere Skript gespeichert.
- Werden Änderungen im Skript vorgenommen, so wird anhand eines ***** hinter dem Dateinamen angezeigt, dass die Änderung noch nicht gespeichert wurde.



Geben Sie nun die ersten Zeilen des Skripts in den Editor ein:

```
% Flaeche.m
%
% Berechnung von Flaeche und Umfang von
% - Kreis
% - innerem Quadrat
% - aeusserem Quadrat
```



```
% Loeschen aller Variablen
clear
```

```
% Definition des Radius
r = 5;
```

```
% Berechnung fuer den Kreis
U_K = 2*pi*r
A_K = pi*r^2
```

¹⁸ Neben der Ansicht im Current folder können Sie den Aktuellen Arbeitspfad über den Befehl **pwd** (path of working directory) herausfinden. Eine Änderung des Arbeitsverzeichnisses ist auch über **cd** (change directory) möglich.

Es gibt mehrere Möglichkeiten dieses Skript zu testen:

- Klicken Sie auf das Icon  oder nutzen Sie die Taste **F5**: die Datei wird gespeichert und das Programm ausgeführt. Die Ergebnisse der Berechnungen werden im Befehlsfenster ausgegeben.
- Speichern Sie die Datei durch Klick auf  (alternativ: **Strg+S**) und geben Sie im **Command Window** (Befehlsfenster) den Skriptnamen als Befehl ein: **Flaeche**. Das Skript wird ausgeführt und das Ergebnis der Berechnungen wird im Befehlsfenster ausgegeben.

Aufgabe 2: Skript vervollständigen

Um Skriptdateien später wiederzufinden, wird im Folgenden eine fortlaufende Bezeichnung eingeführt, die der Nomenklatur

Versuchsnummer_Aufgabennummer_Beschreibung.m folgt.

Speichern Sie so zunächst das Skript **Flaeche.m** unter **V1_A02_Flaeche2.m**.

Erweitern Sie es um die noch fehlenden Berechnungen und testen Sie das Skript, indem Sie Umfänge und Flächen für verschiedene Radien berechnen. Beantworten Sie anschließend folgende Fragen:

- Der Befehl **clear** löscht alle Variablen im Workspace. Wie kann man nur bestimmte Variablen löschen? Schauen Sie hierzu mit **help clear** oder mit **doc clear** in der Dokumentation nach und geben Sie den Befehl zum Löschen der Variablen **r** an.
- Worin unterscheiden sich die Befehle **help** und **doc**? Finden Sie weitere Möglichkeiten, die Hilfe aufzurufen?
- Warum steht hinter **r = 5** ein Semikolon, hinter den anderen Gleichungen nicht?
- Markieren Sie zum Debuggen einen Ausdruck mit dem Cursor und führen Sie ihn mit der Taste **F9** aus. Beobachten Sie, was passiert.
- Setzen Sie Blöcke **%% Blockbezeichnung** ein, um das Skript zu strukturieren und führen Sie die Blöcke abschnittsweise mit der Tastenkombination **Strg+Enter** aus.

4.3.3 Strings und formatierte Ausgabe der Ergebnisse

Wenn Sie Ihr Skript ausführen, erhalten Sie die folgende, nicht sonderlich übersichtliche, Ausgabe im Befehlsfenster:

```
>> Flaeche
U_K = 31.416
A_K = 78.540
U_i = 28.284
A_i = 50
U_a = 40
A_a = 100
```

Schöner wäre eine Ausgabe in der Form

```
Radius r = 5.0
Kreis:           U = 31.42,   A = 78.54
inneres Quadrat: U = 28.28,   A = 50.00
aeusseres Quadrat: U = 40.00, A = 100.00
```

Aus der C-Programmierung ist der Befehl **fprintf** bekannt, der beim Schreiben in eine Datei eine formatierte Ausgabe ermöglicht. Dieser Befehl steht mit fast identischer Syntax auch in der MATLAB-Sprache zur Verfügung.

Darüber hinaus gibt es die Funktion **sprintf**, die das formatierte Ergebnis statt in eine Datei in einen String schreibt, der dann mit dem Befehl **disp** im Befehlsfenster ausgegeben werden kann. Der Befehl für die erste Zeile der Ausgabe lautet damit:

```
disp(sprintf('Radius r = %2.1f', r))
```

Dieser Befehl soll nun genauer analysiert werden:

disp()	Befehl zur Ausgabe eines Strings im Befehlsfenster
sprintf()	Befehl zum Erzeugen eines Strings mit formatierten Variablen
'Radius r = %2.1f'	String mit dem festen Text Radius r = und dem Platzhalter %2.1f für die formatierte Ausgabe des Radius. Strings werden in der MATLAB-Sprache durch <u>einfache</u> Hochkommas begrenzt.
%2.1f	Platzhalter für die formatierte Ausgabe einer Variablen. Die Notation ist wie aus C bekannt, das heißt % leitet den Platzhalter ein, die Variable wird als Fließkommazahl (f) mit 2 Vor- und einer Nachkommastelle ausgegeben.
, r	Nach dem String werden durch Kommas abgetrennt für jeden Platzhalter die Variablen übergeben.

Dieselbe Ausgabe lässt sich auch mit der Funktion **fprintf()** (formatierte Ausgabe in eine Datei / file) erreichen: in der Hilfe **doc fprintf()** ist nachzulesen, dass **fprintf('Radius r = %2.1f\n', r)** das Ergebnis auf dem Bildschirm ausgibt.¹⁹ Anders als beim Einsatz von **sprintf()** muss das newline-Zeichen **\n** explizit angegeben werden. In neueren MATLAB-Releases wird diese Syntax zur formatierten Standardausgabe empfohlen.

¹⁹ Hierbei wird die Syntaxoption **fprintf(formatSpec,A1,...,An)** eingesetzt. Alternativ kann die Syntaxoption **fprintf(fileID,formatSpec,A1,...,An)** gewählt werden, wobei als **fileID** der Wert **1** gewählt wird, was dem Standard output (**Screen**) entspricht).

Aufgabe 3: Formatierte Ausgabe

Speichern Sie das Skript **V1_A02_Flaeche2.m** unter dem neuen Namen **V1_A03_Flaeche_3.m** und modifizieren Sie es so, dass das Ergebnis wie oben dargestellt ausgegeben wird.

- Nutzen Sie hierfür die beschriebenen Befehle **disp** und **sprintf** bzw. **fprintf**
- Unterdrücken Sie die Ausgabe der Ergebnisse bei der Berechnung mit einem Semikolon.
- Schauen Sie bei Bedarf mit dem Befehl **doc** in der Dokumentation nach, um mehr über die Syntax und Möglichkeiten der Befehle zu erfahren.
- Testen Sie Ihr Programm

Im zweiten Schritt sollen Sie mit dem Befehl

```
r = input('Geben Sie den Radius ein: ')
```

den Radius für die Berechnung interaktiv vom Benutzer abfragen.²⁰

4.4 Komplexe Zahlen

Die bisherigen Rechnungen wurden mit 64 Bit IEEE-Fließkommazahlen als Standarddatentyp durchgeführt. MATLAB / Octave kann jedoch deutlich mehr und rechnet mit komplexen Zahlen genauso wie mit reellen. Als Beispiel berechnen Sie im Command Window die Quadratwurzel von -1 :

```
>> sqrt(-1)  
ans = 0 + 1i
```

Sie erhalten als Ergebnis die korrekte Lösung $\sqrt{-1} = 0 + 1 \cdot i = i$. Anhand dieses Ergebnisses lassen sich zwei wichtige Eigenschaften ablesen:

- MATLAB / Octave bezeichnet die imaginäre Einheit mit i und nicht wie in der Elektrotechnik mit j .
- Entsprechend dem Fundamentalsatz der Algebra gibt es eigentlich zwei Lösungen: $\sqrt{-1} = \pm i$. MATLAB / Octave liefert jedoch nur die positive Lösung. Dies gilt – genau wie bei Ihrem Taschenrechner – auch für reelle Zahlen:

```
>> sqrt(4)  
ans = 2
```

²⁰ In welcher Hinsicht unterscheidet sich **input()** von den aus C bekannten Ein- und Ausgabefunktionen?

Umgekehrt können Sie natürlich auch komplexe Zahlen direkt eingeben. Hierfür sind die Variablen **i** und **j** als imaginäre Einheit vordefiniert:

```
>> i^2
ans = -1
>> j^2
ans = -1
>> 1j^2
ans = -1
```

VORSICHT

Wie andere Variablen auch, können Sie **i** und **j** mit anderen Werten überschreiben. Da diese beiden Buchstaben auch gerne als Zählvariablen in Schleifen verwendet werden, kann es so zu schwierig zu findenden Fehlern kommen.

Soll mit **i** bzw. **j** die imaginäre Einheit bezeichnet werden, so kann die Darstellung des Imaginärteils einer komplexen Zahl ohne Multiplikationsoperator ***** erfolgen, z.B. **1i** bzw. **1j**.

Mit diesem Wissen können Sie komplexe Zahlen sowohl in kartesischer als auch in Exponentialform eingeben und damit rechnen:

```
>> z1 = 2-3j
z1 = 2 - 3i
>> z2 = sqrt(2)*exp(1j*pi/4)
z2 = 1.0000 + 1.0000i
>> z1/z2
ans = -0.50000 - 2.50000i
```

Anhand dieses Beispiels können Sie folgendes beobachten:

- Sie können mit komplexen Zahlen genauso rechnen wie mit reellen.
- Bei der Ausgabe verwendet MATLAB / Octave immer die algebraische Form mit *i* als imaginärer Einheit

Aufgabe 4: Rechnen mit komplexen Zahlen

Gegeben sind die komplexen Zahlen $z_1 = 3 + j$ und $z_2 = -1 + 2j$.

a) Berechnen Sie $\frac{z_1}{(z_2)^2}$

b) Lösen Sie $(z_3)^4 = z_1 + z_2$ nach z_3 auf.

Geben Sie das Ergebnis von Teil a) in algebraischer Form und von Teil b) in Exponentialform an (ehemalige Klausuraufgabe)

Erstellen Sie ein Skript **V1_A04_Komplex.m** zur Lösung der Aufgabe. Beachten Sie dabei:

- Befehle für komplexe Zahlen finden Sie in der MATLAB / Octave Kurzreferenz [4] oder der MATLAB / Octave Hilfe [5].
- Geben Sie die Ergebnisse formatiert aus. Für die algebraische Form müssen Sie hierbei Real- und Imaginärteil einzeln übergeben.
- MATLAB / Octave kann nicht direkt in Exponentialform ausgeben. Sie müssen daher aus dem Ergebnis aus Teil b) Betrag und Winkel berechnen.

5 Matrizen und Vektoren

MATLAB steht für „MATrix LABoratory“, wurde also ursprünglich für das bequeme Rechnen mit Matrizen entwickelt. Ohne es zu merken haben Sie in den vorherigen Aufgaben bereits mit Matrizen gerechnet: die reellen und komplexen Zahlen sind für MATLAB nichts weiter als Matrizen mit nur einer Zeile und einer Spalte.

Sie können dies leicht erkennen, wenn Sie sich mit dem Befehl **whos** („who is“) genauere Informationen zu einer Variablen²¹ ausgeben lassen:

```
>> a = 5.1;
>> whos a
Variables in the current scope:
  Attr Name      Size      Bytes  Class
  ==== =====
      a         1x1         8  double
Total is 1 element using 8 bytes
```

Die Variable **a** ist demnach eine Matrix der Größe (**Size**) **1x1**, ist vom Typ (**Class**) **double** und benötigt daher als 64 Bit IEEE-Fließkommazahl 8 Bytes im Speicher.

²¹ Wenn Sie nur **whos** ohne weitere Angaben eingeben, werden alle Variablen aufgelistet.

Die Angabe über die Größe erhalten Sie im Fenster Arbeitsumgebung mit dem Befehl **size**:

```
>> size(a)
ans =
     1     1
```

Die erste Zahl gibt die Anzahl der Zeilen, die zweite Zahl die Anzahl der Spalten an.

MATLAB / Octave rechnet grundsätzlich mit Matrizen, deren Elemente reell (64 Bit IEEE-Fließkommazahl) oder auch komplex (zwei reelle Zahlen als Real- und Imaginärteil) sein können. Zahlen und Vektoren sind nur Sonderfälle:

- Spaltenvektor: Matrix mit nur einer Spalte
- Zeilenvektor: Matrix mit nur einer Zeile
- Skalar / Zahl: Matrix mit einer Zeile und einer Spalte

5.1 Matrizen

Um eine Matrix zu definieren, geben Sie deren Elemente zeilenweise zwischen eckigen Klammern ein, dabei trennen Sie

- die Elemente einer Zeile mit Leerzeichen oder Kommas,
- die Zeilen mit Semikolons.

Geben Sie nun die folgende Matrix ein:

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

```
>> A = [-2 0 0; 1 1 4; -1 0 1]
A =
    -2     0     0
     1     1     4
    -1     0     1
```

Wenn Sie mit Matrizen rechnen, wendet MATLAB / Octave die aus der Vorlesung Mathematik 1 bekannten **Rechenregeln** an:

Matrizenaddition

```
>> A+A
ans =
    -4     0     0
     2     2     8
    -2     0     2
```

Multiplikation Matrix-Skalar

```
>> 2*A
ans =
    -4     0     0
     2     2     8
    -2     0     2
```

Matrizen- Multiplikation

```
>> A*A
ans =
     4     0     0
    -5     1     8
     1     0     1
```

Potenzieren von Matrizen

```
>> A^2
ans =
     4     0     0
    -5     1     8
     1     0     1
```

Zu Berechnung der Transponierten A^T einer Matrix A dient das einfache Hochkomma:

```
>> A'  
ans =  
    -2     1    -1  
     0     1     0  
     0     4     1
```

Wenn die Determinante $\det(A) \neq 0$ ist,

```
>> det(A)  
ans = -2
```

kann die Matrix invertiert werden. Hierzu gibt es die folgenden Möglichkeiten:

<pre>>> inv(A) ans =</pre>	<pre>>> A^(-1) ans =</pre>
-0.50000	-0.50000
-0.00000	-0.00000
-0.00000	-0.00000
2.50000	2.50000
1.00000	1.00000
-4.00000	-4.00000
-0.50000	-0.50000
0.00000	0.00000
1.00000	1.00000

Aufgabe 5: Rechnen mit Matrizen

Gegeben sind die Matrizen

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

Versuchen Sie direkt im Befehlsfenster das Produkt $A \cdot B$ zu berechnen:

- Welche Ausgabe erhalten Sie im Command Window?
- Wie können Sie das Ergebnis erklären?

Schreiben Sie nun ein Skript **v1_A05_Matrizenrechnung.m**, das die folgenden Berechnungen durchführt:

$$\begin{aligned} D &= A^T \cdot B, & E &= B^T \cdot A \\ F &= A \cdot B^T, & G &= B \cdot A^T \\ H &= F \cdot C, & J &= C \cdot F \end{aligned}$$

Welcher Zusammenhang besteht zwischen D und E bzw. zwischen F und G ?

5.2 Vektoren

Vektoren sind Matrizen mit nur einer Zeile bzw. einer Spalte. Daher werden Vektoren genauso definiert wie Matrizen:

- Spaltenvektor $\vec{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ $\mathbf{a} = [1; 2; 3];$
- Zeilenvektor $\vec{b} = (4 \ 5 \ 6)$ $\mathbf{b} = [4 \ 5 \ 6];$

Auch für Vektoren gelten die aus der linearen Algebra bekannten Rechenregeln. Einige Beispiele sind in Tabelle 3 zusammengestellt.

Tabelle 3: Rechenregeln für Vektoren

Operation	Beispiel	MATLAB / Octave	Anmerkung
Addition Subtraktion	$\vec{c} = \vec{a} + \vec{b}^T$	$\mathbf{c} = \mathbf{a} + \mathbf{b}'$	Dimensionen müssen passen
Multiplikation mit Skalar	$\vec{d} = 2 \cdot \vec{a}$	$\mathbf{d} = 2 * \mathbf{a}$	Jedes Element wird mit dem Faktor multipliziert
Skalarprodukt	$e = \vec{a}^T \cdot \vec{c}$	$\mathbf{e} = \mathbf{a}' * \mathbf{c}$	Zeilenvektor mal Spaltenvektor
Vektorprodukt	$\vec{f} = \vec{a} \times \vec{c}$	$\mathbf{f} = \text{cross}(\mathbf{a}, \mathbf{c})$	Nur für Vektoren mit 3 Elementen definiert
Produkt mit Matrix A	$\vec{g} = A \cdot \vec{b}^T$	$\mathbf{g} = \mathbf{A} * \mathbf{b}'$	Dimensionen müssen passen

Aufgabe 6: Rechnen mit Vektoren

Gegeben sind die Matrizen

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

sowie die Vektoren \vec{a} und \vec{b} aus Abschnitt 5.2.

Berechnen Sie zunächst schriftlich von Hand

- die Beispiele zur Vektorrechnung aus Tabelle 3.
- und die Produkte $\vec{h} = B \cdot \vec{a}$ sowie $C = \vec{a} \cdot \vec{c}^T$

Schreiben Sie dann ein Skript **V1_A06_Vektorrechnung.m**, das dieselben Berechnungen durchführt.

5.3 Spezielle Vektoren und Matrizen

In der Vorlesung Mathematik 1 haben Sie einige spezielle Matrizen und Vektoren kennengelernt. Diese – und weitere – müssen Sie auch in der MATLAB-Sprache nicht elementweise eintippen sondern können die in Tabelle 4 zusammengestellten Funktionen nutzen.²²

Tabelle 4: Spezielle Vektoren und Matrizen

Operation	Beispiel	MATLAB / Octave	Anmerkung
Nullmatrix	$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	<code>A = zeros(3,3)</code>	Als Argument wird die Zahl der Zeilen und Spalten übergeben
Nullvektor	$\vec{a} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$	<code>a = zeros(2,1)</code>	
Matrix mit 1-Elementen	$B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	<code>B = ones(2,3)</code>	Als Argument wird die Zahl der Zeilen und Spalten übergeben
Vektor mit 1-Elementen	$\vec{b} = (1 \quad 1)$	<code>b = ones(1,2)</code>	
Diagonalmatrix	$F = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 4 \end{pmatrix}$	<code>F = diag([2 -1 4])</code>	Argument ist Vektor mit den Diagonalelementen
Einheitsmatrix	$E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	<code>E = eye(3)</code> <code>E = diag(ones(3,1))</code>	Quadratische Matrix

²² Als Funktionsargumente werden bei den meisten elementaren Matrizen die Zahl der Zeilen und Zahl der Spalten eingegeben. Wird nur ein Argument übergeben, so führt dies zu einer quadratischen Matrix. Eine Übersicht über weitere elementaren Matrizen finden Sie in der Hilfe mittels `doc elmat`.

5.4 Arrays

Matrizen und Vektoren sind in der MATLAB-Sprache nicht nur die in den Abschnitten 5.1 und 5.2 diskutierten mathematischen Objekte sondern können auch als Arrays (Felder) eingesetzt werden.

Um mit Arrays zu arbeiten, verfügt MATLAB neben den aus den Abschnitten 5.1 und 5.2 bekannten über weitere Operatoren, die in den nächsten Abschnitten vorgestellt werden.

5.4.1 Anlegen von Arrays

Die einfachste Methode zum Anlegen von Arrays kennen Sie bereits aus den Abschnitten 5.1 und 5.2: Tippen Sie die Elemente der Arrays zeilenweise ein und trennen Sie dabei die Elemente einer Zeile durch Leerzeichen oder Kommas und die Zeilen durch Semikolons.

In vielen Fällen benötigen Sie zunächst – als Platzhalter – ein Array aus Nullen oder Einsen, dessen Elemente Sie dann durch andere Werte überschreiben können. Auch hierfür haben Sie bereits in Abschnitt 0 zwei wichtige Funktionen kennengelernt:

- **zeros** (*n*, *m*) erzeugt ein Array mit *n* Zeilen und *m* Spalten, gefüllt mit Nullen.
- **ones** (*n*, *m*) erzeugt ein Array mit *n* Zeilen und *m* Spalten, gefüllt mit Einsen.²³
- Ein leeres Array **A** kann über **A = []** erstellt werden.²⁴

5.4.2 Array mit Zahlenfolge

In zahlreichen Anwendungen benötigen Sie eindimensionale Arrays (Vektoren) mit aufeinanderfolgenden Zahlenwerten. Diese können Sie auf einfache Weise mit dem **:** Operator erstellen. So können Sie zum Beispiel einen Zeilenvektor, also ein eindimensionales Array, mit den Werten 2, 3, ..., 10 mit einer Schrittweite von 1 erzeugen:

```
>> a = 2:10
a =
     2     3     4     5     6     7     8     9    10
```

In vielen technischen Anwendungen benötigen Sie ein Array mit den zu Ihren Messdaten gehörenden Abtastzeitpunkten. Haben Sie zum Beispiel 1 Sekunde lang mit der Abtastzeit $T_A = 0,2$ s gemessen, können Sie den Zeitvektor wie folgt erstellen:

```
>> t = 0:0.2:1
t =
    0.00000    0.20000    0.40000    0.60000    0.80000    1.00000
```

²³ Anders als in C ist dieses Allokieren keine Voraussetzung zum Einsatz von Arrays, es verbessert aber, gerade beim Einsatz von Schleifen die Performanz.

²⁴ Diese Option sollte nicht eingesetzt werden, falls ein Array dadurch innerhalb einer Schleife seine Größe ändert.

Werfen Sie nun anhand der beiden Beispiele einen genaueren Blick auf die Syntax:

`x = a:s:b`

Hierbei sind

- **`x`** der erstellte Zeilenvektor (Array)
- **`a`** der erste Wert
- **`s`** die Schrittweite (optional, Standard ist +1)
- **`b`** der letzte Wert (sofern dieser mit der Schrittweite erreicht werden kann)

Die Schrittweite darf auch negativ sein:

```
>> b = 5:-1:-2
```

```
b =
```

```
5    4    3    2    1    0   -1   -2
```

```
>> c = 1:-0.25:-0.4
```

```
c =
```

```
1.00000    0.75000    0.50000    0.25000    0.00000   -0.25000
```

Im letzten Beispiel kann der letzte Wert -0,4 mit der gegebenen Schrittweite nicht erreicht werden. Die Folge bricht daher beim letztmöglichen Wert -0,25 ab.²⁵

5.4.3 Zusammenfassen von Arrays

Sie können mehrere Arrays (Vektoren und/oder Matrizen) zu einem neuen Array zusammenfassen, indem Sie eine neue Matrix wie gewohnt definieren, jedoch als Elemente nicht nur Skalare sondern auch andere Vektoren und Matrizen verwenden. Dies funktioniert natürlich nur, wenn die Dimensionen „passen“ und insgesamt ein rechteckiges Array entsteht.

Legen Sie als Beispiel 3 Arrays an

```
>> x = 2:7
```

```
x =
```

```
2    3    4    5    6    7
```

```
>> A = 2*ones(2,4)
```

```
A =
```

```
2    2    2    2
2    2    2    2
```

```
>> B = [1 2; 3 4]
```

²⁵ Eine weitere Option zur Erstellen von Arrays bietet die Funktion `linspace(x1, x2, n)`. Mit ihr wird ein Array mit `n` Einträgen erzeugt, die gleichmäßig zwischen `x1` und `x2` verteilt sind, wobei `x1` und `x2` die Randwerte bilden. Der Vektor `0.00000 0.20000 0.40000` mit äquidistant verteilten Einträgen kann somit entweder durch `0:0.2:0.4` oder durch `linspace(0,0.4,3)` erzeugt werden.

B =

1 2
3 4

Diese drei Arrays sollen nun in ein neues Array **M** der Form

2	3	4	5	6	7
2	2	2	2	1	2
2	2	2	2	3	4

zusammengefasst werden. Hierzu bauen Sie das neue Array **M** wie gewohnt auf, verwenden jedoch **x**, **A** und **B** als „Elemente“:

x					
A				B	

- erste Zeile: Array **x**, Abschluss der Zeile mit Semikolon,
- zweite Zeile: Arrays **A** und **B** getrennt durch Leerzeichen (oder Komma).

```
>> M = [x; A B]
```

M =

2 3 4 5 6 7
2 2 2 2 1 2
2 2 2 2 3 4

5.4.4 Arrays aus Daten

Wenn Sie MATLAB / Octave einsetzen, um Messwerte auszuwerten und zu analysieren, möchten Sie diese oft umfangreichen Daten sicherlich nicht von Hand eintippen. Viele Messsysteme (z. B. digitale Speicheroszilloskope) können die Messwerte in sogenannten CSV-Dateien (comma sparated values) speichern. Diese Dateien können Sie sehr einfach in MATLAB bzw. Octave importieren²⁶ und dann auswerten.²⁷

Als Beispiel nehmen wir an, dass Sie mit einem GPS-Messsystem Weg und Geschwindigkeit eines Fahrzeuges gemessen haben. Das System hat die Messwerte als CSV-Datei auf eine Speicherkarte geschrieben, die Sie dann auf Ihre Festplatte kopiert haben. Für das Labor finden Sie die Messdatei **Messung.csv** in der ZIP-Datei mit den Ressourcen zum Laborversuch 1.

Mit einem Texteditor können Sie einen ersten Blick auf die Messdatei werfen: Abbildung 4 zeigt die ersten Zeilen der Messung im Windows-Editor.

²⁶ MATLAB / Octave kann auch viele andere Datenformate importieren. Hinweise und Beispiele finden Sie in der Hilfe-Funktion – insbesondere in der ausführlicheren Hilfe von MATLAB.

²⁷ Das MATLAB-interne Format zum Speichern von Daten besitzt die Endung *.mat.

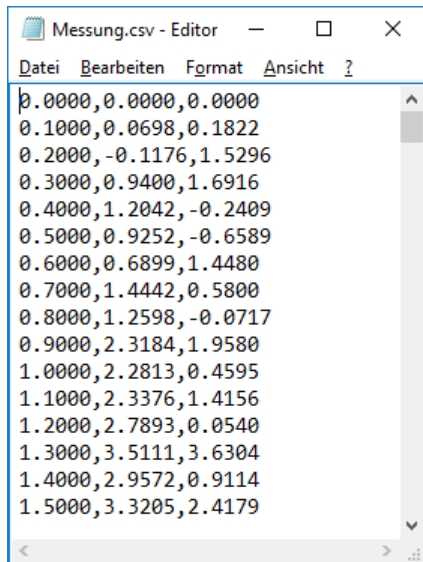


Abbildung 4: Datei `Messung.csv` im Windows-Editor

Es ist nun einfach zu erkennen, dass zu jedem Messzeitpunkt drei durch Kommas getrennte Werte in die Datei geschrieben wurden. Damit ergeben sich drei Spalten, die die folgenden Daten beinhalten:

- Spalte 1: Zeit in Sekunden
- Spalte 2: Geschwindigkeit in m/s
- Spalte 3: zurückgelegter Weg in m

Um eine solche CSV-Datei in MATLAB / Octave zu importieren, können Sie den Befehl `load` verwenden:²⁸

```
>> M = load('Messung.csv');
```

Hierzu müssen Sie zunächst in das Verzeichnis navigieren, in dem sich die Datei befindet (siehe Tabelle 1).

Da das Array sehr groß ist, wurde die Ausgabe der Variablen `M` im Befehlsfenster durch das Semikolon am Schluss unterbunden. Die Messwerte werden in derselben Anordnung wie in der Datei in das Array bzw. die Matrix `M` geschrieben. Mit

```
>> size(M)
ans =
    201     3
```

kann man erkennen, dass das Array 201 Zeilen mit jeweils 3 Spalten besitzt.

Dies kann man auch überprüfen, indem man sich durch Doppelklick auf die gleichnamige Variable im Workspace deren Werte anzeigen lässt.

²⁸ Neben der „function form“ `load('Messung.csv')` unterstützt MATLAB neuerdings auch die syntaktisch kürzere „command form“ `load Messung.csv`.

Für die weitere Auswertung müssen nun die drei gemessenen Größen Zeit, Geschwindigkeit und Weg daraus extrahiert werden. Dies werden Sie in der nächsten Aufgabe selbst tun – nachdem Sie im nächsten Abschnitt mehr über den Zugriff auf Arrays erfahren haben.

5.4.5 Zugriff auf eindimensionale Arrays

Der Zugriff auf Arrays erfolgt im Prinzip genauso wie in jeder anderen Programmiersprache auch: Sie müssen angeben, auf welches Element Sie zugreifen möchten. Am besten probieren Sie die folgenden Beispiele selbst aus.

Gegeben ist ein eindimensionales Array:

```
>> a = [1 3 7 2 4 8 3 4]
a =
     1     3     7     2     4     8     3     4
```

Um auf ein beliebiges Element zuzugreifen, übergeben Sie dessen Index in runden Klammern:

```
>> a(3)
ans = 7
>> a(4) = -1
a =
     1     3     7    -1     4     8     3     4
```

VORSICHT

Bei MATLAB / Octave beginnt die Nummerierung der Array-Elemente wie in der Mathematik bei Vektoren und Matrizen immer mit 1, nicht mit 0 wie in C.

Das erste Element hat immer den Index 1. Wenn Sie jedoch auf das letzte Element eines Arrays zugreifen möchten, müssen Sie zunächst dessen Größe mit dem Befehl **size** (für zweidimensionale Arrays) bzw. **length** (für eindimensionale Arrays) ermitteln.

```
>> laenge = length(a)
laenge = 8
>> a(laenge)
ans = 4
```

Einfacher und besser lesbar erfolgt der Zugriff jedoch mit dem Schlüsselwort **end**

```
>> a(end)
ans = 4
```

MATLAB / Octave bietet noch weit mächtigere Möglichkeiten auf Arrayelemente zuzugreifen. Hierzu müssen Sie sich wieder bewusst machen, dass im Unterschied zu den meisten anderen Programmiersprachen der Grunddatentyp nicht ein Skalar, sondern eine Matrix ist. Sie können damit die Elemente Nummer 7, 1 und 6 – in genau dieser Reihenfolge – aus dem

Array **a** entnehmen, indem Sie das Array statt mit nur einer Zahl mit dem Vektor **[7 1 6]** indizieren:

```
>> a([7 1 6])
ans =
     3     1     8
```

Zusammen mit dem **:** Operator aus Abschnitt 5.4.1 ergeben sich sehr elegante Möglichkeiten, Daten aus Arrays zu entnehmen, z. B:

- die ersten drei Elemente

```
>> a(1:3)
ans =
     1     3     7
```
- die letzten drei Elemente

```
>> a(end-2:end)
ans =
     8     3     4
```
- Elemente 2 bis 6

```
>> a(2:6)
ans =
     3     7    -1     4     8
```
- jedes zweite Element

```
>> a(1:2:end)
ans =
     1     7     4     3
```
- Umkehren der Reihenfolge

```
>> b = a(end:-1:1)
b =
     4     3     8     4    -1     7     3     1
```

Auch in der zählergesteuerten Schleife wird die Zählung über Arrays durchgeführt, vgl. **doc for** bzw. [1], Kapitel 2.3.4.

5.4.6 Zugriff auf zweidimensionale Arrays

Bei zweidimensionalen Arrays werden für den Zugriff zwei Werte übergeben: die Zeile und die Spalte des Elements, auf das zugegriffen werden soll. Die Nummerierung erfolgt dabei wie bei Matrizen gewohnt:

- Zeilen und Spalten werden beginnend mit 1 nummeriert,
- Zeile 1 ist oben,
- Spalte 1 ist links.

Als Beispiel wird das folgende Array verwendet:

```
>> M = [1 3 2.5 4 7; 1.7 2 8 5 1; 3 9 1 2.8 2]
M =
    1.0000    3.0000    2.5000    4.0000    7.0000
    1.7000    2.0000    8.0000    5.0000    1.0000
    3.0000    9.0000    1.0000    2.8000    2.0000
```

Der Zugriff auf ein Element erfolgt durch Übergabe von Zeile und Spalte, die durch ein Komma getrennt werden:


```
>> M(2,4)
ans = 5
>> M(3,2) = 0
M =
    1.00000    3.00000    2.50000    4.00000    7.00000
    1.70000    2.00000    8.00000    5.00000    1.00000
    3.00000    0.00000    1.00000    2.80000    2.00000
```

Wie bei den eindimensionalen Arrays können statt der Zeilen- bzw. Spaltennummern Vektoren übergeben werden. Im folgenden Beispiel werden damit die letzten drei Elemente der ersten Zeile durch [3.2 5 6] ersetzt:

```
>> M(1,end-2:end) = [3.2 5 6]
M =
    1.00000    3.00000    3.20000    5.00000    6.00000
    1.70000    2.00000    8.00000    5.00000    1.00000
    3.00000    0.00000    1.00000    2.80000    2.00000
```

Das klappt natürlich nur dann, wenn die Dimensionen auf beiden Seiten des Gleichheitszeichens – also der Zuweisung – identisch sind.²⁹

Kommen wir nun wieder zurück zum Beispiel der Geschwindigkeits- und Wegmessung am Auto aus Abschnitt 5.4.4. Die Daten stehen in einer Matrix / einem Array zur Verfügung. Um damit weiterarbeiten zu können, müssen daraus die Zeit (erste Spalte), die Geschwindigkeit (zweite Spalte) und der Weg (dritte Spalte) entnommen werden. Mit den bekannten Möglichkeiten zur Datenentnahme ist es kein Problem, Zeilen oder Spalten aus einer Matrix herauszuziehen:

```
>> zeile1 = M(1,1:end)
zeile1 =
    1.0000    3.0000    3.2000    5.0000    6.0000

>> spalte2 = M(1:end,2)
spalte2 =
    3
    2
    0
```

²⁹ Obgleich es eine Syntaxoption hierfür gibt, sollten zweidimensionale Arrays nicht mit nur einem Index indiziert werden. Dann werden nämlich die Matrix spaltenweise durchlaufen. Für `M = [1 2; 3 4]` gibt die Indizierung `M(2)` den Wert 3 zurück, auf den auch über `M(2,1)` zugegriffen werden könnte und nicht, wie man gemäß der Eingabereihenfolge der Matrix vermuten sollte, den Wert 2 an der Stelle `M(1,2)`.

Da das Herausziehen von Zeilen bzw. Spalten eine sehr häufig benötigte Operation ist, gibt es eine Kurzschreibweise:

Statt `1:end` kann ausschließlich ein Doppelpunkt geschrieben werden:

```
>> zeile2 = M(2,:)
zeile2 =
    1.7000    2.0000    8.0000    5.0000    1.0000

>> spalte4 = M(:,4)
spalte4 =
    5.0000
    5.0000
    2.8000
```

Aufgabe 7: Messdaten einlesen

Erstellen Sie ein Skript `V1_A07_Analysiere_Messung.m` zum Einlesen und Auswerten der Messdaten aus der Datei `Messung.csv` aus Abschnitt 5.4.4.

- Löschen Sie zunächst mit `clear` alle Variablen.
- Lesen Sie die Datei `Messung.csv` in das Array `M` ein.
- Ziehen Sie aus `M` die Arrays (Vektoren) `t` für die Zeit, `v` für die Geschwindigkeit und `s` für den Weg heraus.
- Berechnen Sie die Abtastperiodendauer `TA`, indem Sie zwei aufeinanderfolgende Zeiten subtrahieren.
- Ermitteln Sie mithilfe der Funktion `max` die maximale Geschwindigkeit `vmax` (in m/s) und `vmaxkmh` (in km/h), sowie den während der Messung zurückgelegten Weg `smax`. Informieren Sie sich hierzu auch in der MATLAB-Dokumentation über die Syntax-Optionen der Funktion `max()`.
- Geben Sie die ermittelten Werte formatiert aus. Die Ausgabe soll wie folgt aussehen:

```
>> Analysiere_Messung
Abtastzeit:    0.10 s
max. Geschw.: 40.10 m/s = 144.35 km/h
Weg:          399.94 m
```

5.4.7 Elementweise Rechenoperationen

Beim Rechnen mit Matrizen und Vektoren wendet MATLAB / Octave wie in den Abschnitten 5.1 und 5.2 gezeigt, die aus der Vorlesung Mathematik 1 bekannten Rechenregeln an. Wenn Sie jedoch Vektoren und Matrizen als ein- bzw. zweidimensionale Arrays verwenden, sind diese Rechenregeln nicht immer sinnvoll.

Bei der Analyse der Fahrzeugmessung soll die Luftwiderstandskraft F_L mit der bekannten Gleichung

$$F_L = \frac{1}{2} \cdot \rho \cdot c_w \cdot A \cdot v^2$$

berechnet werden. Die hierbei verwendeten Konstanten sind in Tabelle 5 zusammengestellt.

Tabelle 5: Konstanten für die Berechnung der Luftwiderstandskraft

Konstante	Wert
ρ Luftdichte	$\rho = 1,2 \frac{\text{kg}}{\text{m}^3}$
c_w Luftwiderstandskoeffizient	$c_w = \frac{1}{3}$
A Querschnittsfläche	$A = 2,5 \text{ m}^2$

Setzt man die Konstanten ein, so ergibt sich – ohne Einheiten – die folgende Gleichung zur Berechnung der Luftwiderstandskraft:

$$F_L = \frac{1}{2} \cdot v^2 = \frac{1}{2} \cdot v \cdot v$$

Gegeben ist nun ein Array mit den Geschwindigkeitswerten:

```
>> v = [0 2 4 6 8]
v =
    0     2     4     6     8
```

Zur Berechnung der Luftwiderstandskraft muss nun v^2 bzw. das Produkt $v \cdot v$ für jeden Geschwindigkeitswert – also für jedes Element des Vektors/Arrays \mathbf{v} – berechnet werden. Hierfür sind die Rechenregeln der Vektormultiplikation nicht geeignet, denn damit kann für den gegebenen Vektor \mathbf{v} höchstens das Skalarprodukt

```
>> v*v'
ans = 120
```

berechnet werden, was sicherlich nicht für die Berechnung der Luftwiderstandskraft benötigt wird. Die Eingabe von \mathbf{v}^2 oder $\mathbf{v}*\mathbf{v}$ führt zu Fehlermeldungen, da diese Rechenoperationen für Vektoren mathematisch nicht definiert sind.

Die MATLAB-Sprache bietet mit den sogenannten **Punkt-Operatoren** die Möglichkeit elementweise Rechnungen durchzuführen:

- . * elementweise Multiplikation
- . / elementweise Division
- . ^ elementweise Potenzierung

Mit dem Punkt-Operator werden elementweise Rechenoperationen (Multiplikation, Division, Potenzieren) definiert, die auf jedes Element einer Matrix (bzw. auf die Elemente an derselben Position zweier gleichgroßer Matrizen) angewendet werden. Die Ergebnismatrix besitzt dieselben Dimensionen wie die Operanden.

Durch elementweise Operationen kann das Schreiben von Schleifen zur Durchführung numerischer Berechnungen vermieden werden.

Damit kann die Luftwiderstandskraft auf zwei Arten berechnet werden:

```
>> FL = 1/2*v.^2
FL =
    0     2     8    18    32
```

```
>> FL = 1/2*v.*v
FL =
    0     2     8    18    32
```

Es gibt weitere elementweise Rechenoperationen für Arrays:

- Addition und Subtraktion werden bei Vektoren und Matrizen immer elementweise durchgeführt. Daher gibt es hierfür keine Punkt-Operatoren.
- Alle Funktionen, die für Vektoren bzw. Matrizen nicht definiert sind, wie zum Beispiel die Winkelfunktionen oder die Logarithmusfunktion werden elementweise berechnet, z. B.

```
>> sqrt(v)
ans =
    0.00000    1.41421    2.00000    2.44949    2.82843
```

Der Punkt-Operator wird eingesetzt

- wenn beide Operanden Arrays derselben Größe sind und eine der Rechenoperationen Addition, Multiplikation oder Division „elementweise“ auf alle Arrayelemente angewendet werden soll (Beispiel $v \cdot v$)
- Wenn bei der Potenzrechnung ein Array mit einem Skalar verrechnet werden soll (Beispiel $v \cdot 2$)
- Wenn bei der Division im Nenner ein Array steht.

Es kann auch die Berechnung v^v durchgeführt werden, die für die Fahrzeugmessung jedoch nicht sinnvoll ist.

```
>> v.^v
ans =
         1         4        256       46656      16777216
```

Ebenso ist es möglich, den Kehrwert aller Geschwindigkeitswerte zu berechnen:³⁰

```
>> 1./v
ans =
      Inf    0.50000    0.25000    0.16667    0.12500
```

Neben der elementweisen Berechnung der Kehrwerte liefert diese Berechnung noch ein weiteres interessantes Ergebnis: bei der Division durch 0 gibt MATLAB / Octave keine Fehlermeldung aus, sondern liefert als Ergebnis **Inf** (infinity).

Bei der Division 0/0 wird hingegen **NaN** (not a number) zurückgegeben:³¹

```
>> 1/0
warning: division by zero
ans = Inf
>> 0/0
warning: division by zero
ans = NaN
```

Aufgabe 8: Messdatenanalyse erweitern

Erweitern Sie Ihr Skript **V1_A07_analysiere_Messung.m**, um die Berechnung von Luftwiderstandskraft und Leistung und speichern Sie das Ergebnis unter **V1_A08_analysiere_Messung2.m**

- Berechnen Sie aus der Geschwindigkeit die Luftwiderstandskraft **FL** mit den in Tabelle 5 gegebenen Parametern.
- Berechnen Sie zusätzlich die zur Überwindung der Luftwiderstandskraft notwendige Motorleistung $P_L = F_L \cdot v$ in kW. Speichern Sie diese im Array **PL**.
- Berechnen Sie die maximale Luftwiderstandskraft und die maximale Leistung und geben Sie diese formatiert aus.³²

³⁰ Dies ist auch möglich, indem zunächst ein Array mit Einsen gebildet wird, das dieselbe Größe wie den Nenner aufweist: `ones(size(v))./v`

³¹ Die hier gezeigten Warnings werden nur von Octave, nicht jedoch von MATLAB ausgegeben. Im Gegensatz zu **Inf** bezeichnet **NaN** einen nicht definierten Ausdruck (z.B. ein Grenzwert der Form 0/0 oder ∞/∞)

³² Sie können hierbei bei der Textausgabe die physikalischen Einheiten ergänzen.

6 Grafiken und Funktionen

6.1 Darstellen von Daten in Grafiken

Bei der Auswertung von Messdaten – wie zum Beispiel der bereits mehrfach verwendeten Geschwindigkeits- und Wegmessung – ist die graphische Darstellung der Messverläufe oft die Basis für weitere Untersuchungen und Überlegungen.

Da die Daten der Automessung sehr umfangreich sind, nehmen wir zum Einstieg in die Erstellung von Grafiken aus Daten ein einfacheres Beispiel: Nehmen Sie an, Sie haben bei zwei Fahrten mit dem TGV von Karlsruhe nach Paris zu einigen Zeitpunkten die auf dem Display angezeigte Geschwindigkeit notiert (siehe Tabelle 6 für Fahrt 1 und Tabelle 7 für Fahrt 2) und möchten nun die Geschwindigkeitsverläufe grafisch darstellen.

Tabelle 6: Gemessene Daten bei der ersten Zugfahrt von Karlsruhe nach Paris

t / min	0	15	30	60	80	120	130	150	160	180
v / km/h	0	120	50	270	320	250	300	120	50	0

Tabelle 7: Gemessene Daten bei der zweiten Zugfahrt von Karlsruhe nach Paris

t / min	0	10	25	50	90	130	155	170	180
v / km/h	0	100	60	250	320	280	100	30	0

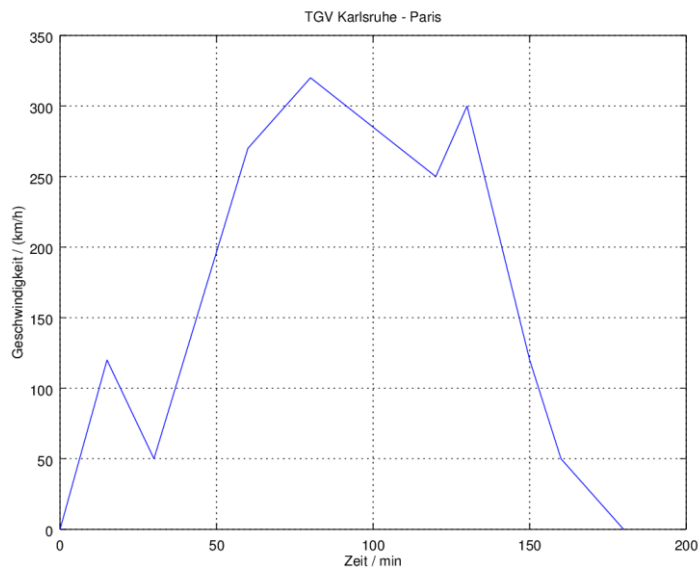
Aufgabe 9: Grafik der ersten Zugfahrt

Erstellen Sie ein Skript **v1_A09_Grafik_Zugfahrt.m** mit den folgenden Befehlen

- Beginnen Sie das Skript mit dem Löschen der Variablen mit **clear**. Mit **clc** können Sie Einträge des Command Windows entfernen.
- Als zweiten Befehl schließen Sie mit **close all** alle Grafikfenster (die es bei der ersten Ausführung natürlich noch nicht gibt)
- Erzeugen Sie dann zwei Zeilenvektoren **t1** und **v1** mit den Daten aus Tabelle 6.
- Geben Sie danach die folgenden Befehle ein:


```
plot(t1, v1)
xlabel('Zeit / min')
ylabel('Geschwindigkeit / km/h')
title('TGV Karlsruhe - Paris')
grid
```
- Führen Sie das Skript aus.

- Sie erhalten die folgende Grafik:



Beantworten Sie nun die folgenden Fragen; schauen Sie dafür bei Bedarf auch in der Dokumentation oder der MATLAB / Octave Kurzreferenz [4] oder in der Hilfe [5] nach:

- In welcher Form werden die Parameter an den Befehl `plot` übergeben?
- Wie werden die Daten dargestellt? Achten Sie insbesondere auf den Verlauf zwischen den Messpunkten.
- Was bewirken die Befehle `xlabel`, `ylabel`, `title` und `grid`? Zur Beantwortung dieser Frage können Sie einen oder mehrere Befehle mit `%` auskommentieren und dann das Ergebnis anschauen.
- Ersetzen Sie nun nacheinander den `plot`-Befehl durch die folgenden Varianten (die anderen kommentieren Sie jeweils aus) und diskutieren Sie das Ergebnis:

a)

```
plot(t1, v1, 'o')
```

b)

```
plot(t1, v1, '-', t1, v1, 'o')
```

c) [Zusatz]

```
plot(t1, v1, 'Color', 'k', 'Marker', 'o', 'LineStyle', 'none')
hold on
plot(t1, v1, 'Color', 'r', 'LineStyle', '--', 'LineWidth', 2)
hold off
```

Hinweis: Die Formatierungen für Linien- und Markerstil, sowie für die Farbe kann auch in Kurzform angegeben werden:

```
plot(t1,v1,'b--o')
```

Aufgabe 10: Grafik beider Zugfahrten

Speichern Sie Ihr Skript **V1_A09_Grafik_Zugfahrt.m** unter dem neuen Namen **V1_A10_Grafik_2_Zugfahrten.m** und nehmen Sie folgende Erweiterungen bzw. Änderungen vor.

- Erzeugen Sie zusätzlich die Vektoren **t2** und **v2** mit den Daten aus Tabelle 7
- Ändern Sie den **plot**-Befehl ab: **plot(t1, v1, t2, v2)**
- Ergänzen Sie nach dem **grid**-Befehl die folgende Zeile:
legend('Fahrt 1', 'Fahrt 2')

Diskutieren Sie die Auswirkungen der vorgenommenen Änderungen, schauen Sie hierfür bei Bedarf auch in der MATLAB / Octave Kurzreferenz [4] und der Hilfefunktion [5] nach.

Aufgabe 11: Grafik der Daten aus der Messdatei

Erweitern Sie Ihr Skript **V1_A08_analysiere_Messung2.m** um die Darstellung der Messverläufe und speichern Sie es unter **V1_A11_analysiere_Messung3.m**

- Hierzu sind mehrere Grafiken erforderlich. Mit dem Befehl **figure(1)** können Sie dazu jeweils ein neues Grafikfenster anlegen.³³
- Schließen Sie zu Beginn des Skripts mit dem Befehl **close all** alle Grafikfenster und leeren Sie das Command Window mit **clc**.
- Erzeugen Sie mit den vorhandenen Daten die folgenden Grafiken mit Achsenbeschriftungen:
 - Geschwindigkeit abhängig von der Zeit
 - Weg abhängig von der Zeit
 - Geschwindigkeit abhängig vom Weg
 - Luftwiderstand abhängig von der Zeit

Nutzen Sie diese Grafiken, um die Messung zu interpretieren:

- Welches Fahrmanöver wurde durchgeführt?
- Bestimmen Sie den wesentlichen Parameter des Fahrmanövers.

³³ Auch die Kurzform **figure** ist möglich. Durch Hinzufügen einer Nummer weisen Sie dem Plot eine eindeutige Nummer zu: **figure(1)**. Ergänzen Sie **clf (clear figure)**, um ein Überschreiben einer anderer **figure** mit derselben Nummer zu vermeiden (die möglicherweise in einem anderen Skript erstellt wurde)

6.2 Schaubilder von Funktionen

Um Schaubilder von Funktionen zu erzeugen, müssen Sie Ihr Wissen zum Erstellen von Grafiken aus Daten (Kapitel 6.1), zum Erzeugen von Arrays (Kapitel 5.4.1) und zu elementweisen Rechenoperationen (Kapitel 5.4.7) kombinieren. Um die mathematische Funktion

$$y(x) = \frac{2 \cdot x^2}{e^x}$$

für $0 \leq x \leq 10$ zu zeichnen, gehen Sie wie folgt vor:

- Erzeugen Sie einen Vektor mit den x -Werten des Schaubilds. Da die Datenpunkte später linear verbunden werden, sollten Sie den Vektor mit einer kleinen Schrittweite erzeugen: **`x = 0:0.01:10;`** damit der Kurvenverlauf möglichst glatt erscheint.
- Berechnen Sie mithilfe elementweiser Rechenoperationen einen Vektor mit den dazugehörigen Funktionswerten: **`y = 2*x.^2./exp(x)`**; Durch die elementweise Rechnung besitzt der Vektor **`y`** dieselben Dimensionen wie der Vektor **`x`**.
- Stellen Sie den Funktionsverlauf mit dem **`plot`**-Befehl dar, z. B. **`plot(x,y)`**
- Ergänzen Sie Beschriftungen und bei Bedarf Gitter und Legende

Für die Beispieleingaben erhalten Sie dann die in Abbildung 5 gezeigte Grafik:

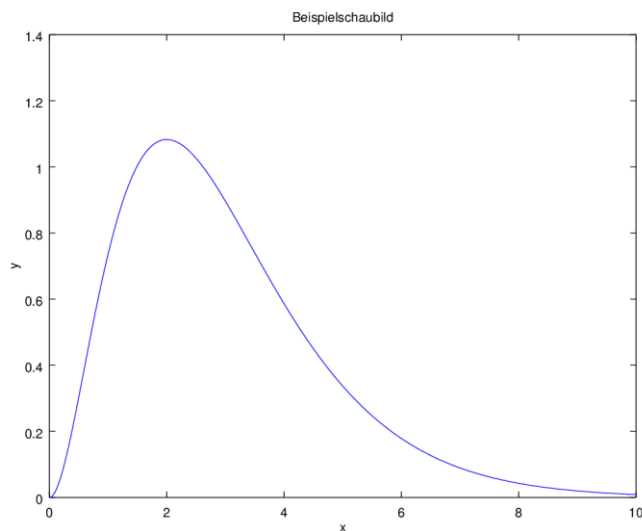


Abbildung 5: Schaubild der Beispielfunktion

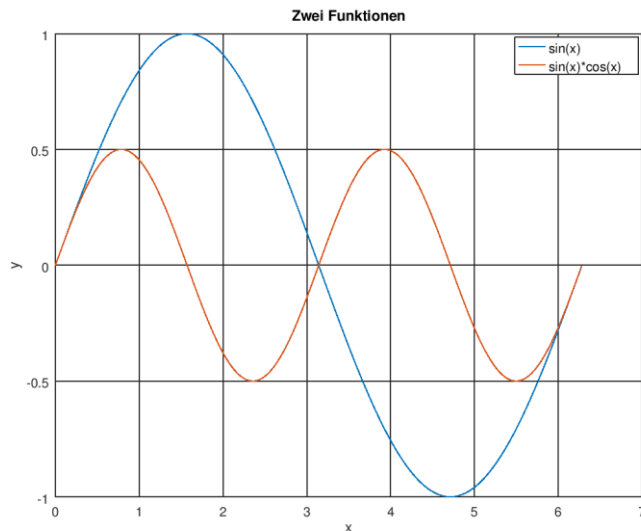
Aufgabe 12: Plotten zweier Funktionen

Erstellen Sie ein vollständig kommentiertes Skript

V1_A12_Zwei_Funktionsgraphen.m, das die beiden Funktionen

$$y_1(x) = \sin(x), \quad y_2(x) = \sin(x) \cdot \cos(x)$$

für $0 \leq x \leq 2 \cdot \pi$ in einem gemeinsamen Schaubild darstellt:



6.3 Funktionen

In Abschnitt 4.3 haben Sie Skripte zur Erstellung von Programmen in der MATLAB-Sprache kennengelernt. Ein Skript ist im Prinzip nichts anderes als eine Abfolge von Befehlen, die Sie mit etwas Mühe auch direkt im Befehlsfenster hätten eingeben können. Daher sind alle in einem Skript verwendeten Variablen global. Daraus ergeben sich folgende wichtige Konsequenzen für die Programmierung von Skripten:

- Nach dem Ablauf des Skripts stehen alle Variablen im Workspace zur Verfügung.
- Wenn vor dem Aufruf eines Skripts bereits Variablen definiert wurden, kann das Skript darauf direkt zugreifen.
- Es gibt keine definierte Schnittstelle zur Übergabe von Parametern an das Skript und zur Rückgabe von Ergebnissen.

Wie in anderen Programmiersprachen (z. B. in C) können Sie in der MATLAB-Sprache auch **Funktionen** erstellen, die sich von den bekannten Skripten in zwei wesentlichen Punkten unterscheiden:

- Die Funktion hat eine definierte Schnittstelle, über die Parameter an die Funktion übergeben und die Ergebnisse der Funktion zurückgegeben werden.
- Die innerhalb einer Funktion verwendeten Variablen sind lokal, also außerhalb der Funktion nicht sichtbar.

Die Funktion wird in ein eigenes m-File geschrieben, das denselben Namen erhält wie die Funktion selber: Als Beispiel für eine Funktion erstellen Sie eine Datei **meinpolynom.m** mit dem folgenden Inhalt:

```
function y = meinpolynom(x)
% Berechnet das Polynom
% y = 3*(x+1)^2*(x-1)
y = 3*(x+1).^2.*(x-1);
end
```

Danach können Sie Ihre erste selbsterstellte Funktion im Command Window genauso verwenden wie die eingebauten Funktionen:

```
>> meinpolynom(0)
ans = -3
>> y1 = meinpolynom(1)
y1 = 0
```

An diesem einfachen Beispiel können Sie sehen, wie Funktionen erstellt werden:

- Jede Funktion benötigt eine eigene Datei mit dem Namen **Funktionsname.m**
- Am Anfang der Datei steht der Funktionskopf
function y = Funktionsname(x, ...)
 - **y** ist die Ausgangsgröße³⁴ der Funktion
 - **x, ...** sind die Eingangsgrößen der Funktion
- Zur Unterscheidung von einem Skript beginnt die erste Zeile mit dem Schlüsselwort **function**
- **Funktionsname** muss – bis auf die Endung **.m** – mit dem Dateinamen der Funktion übereinstimmen.
- Der unmittelbar auf den Funktionskopf folgende Kommentar wird angezeigt, wenn im Befehlsfenster **help Funktionsname** eingegeben wird – probieren Sie es aus.
- Das Ende der Funktion wird mit dem optionalen³⁵ Schlüsselwort **end** markiert.
- Der Funktionsname darf nicht anderweitig als Variablenname eingesetzt werden

Vielleicht ist Ihnen aufgefallen, dass bei der Berechnung des Rückgabewertes die **Punkt-Operatoren** **.******* und **.****^** verwendet werden. Damit funktioniert die Funktion auch dann wie gewünscht, wenn sie mit einem Vektor als Argument aufgerufen wird. Bei der Addition und Subtraktion von Konstanten (z. B. im Term **(x+1)**) müssen keine besonderen Operatoren

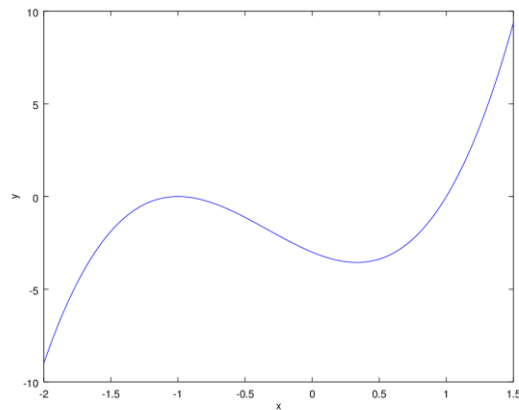
³⁴ Es können auch mehrere Größen zurückgegeben werden: für die Rückgabe der Größen **y** und **z** schreiben Sie **function [y z] = ...** es lohnt sich ein Blick in die Funktions-Vorlage über **Editor -> New -> Function**, um sich mit der Syntax vertraut zu machen.

³⁵ Da das Ende der Funktion auch durch das Ende der Datei **Funktionsname.m** eindeutig definiert ist, kann **end** auch weggelassen werden.

verwendet werden, da MATLAB / Octave automatisch die Konstante zu jedem Element des Vektors addiert bzw. subtrahiert.

Rufen Sie die selbsterstellte Funktion in einem MATLAB-Skript oder über das Command Window auf:

```
>> x = -2:0.01:1.5;  
>> y = meinpolynom(x);  
>> plot(x,y)  
>> xlabel('x')  
>> ylabel('y')
```



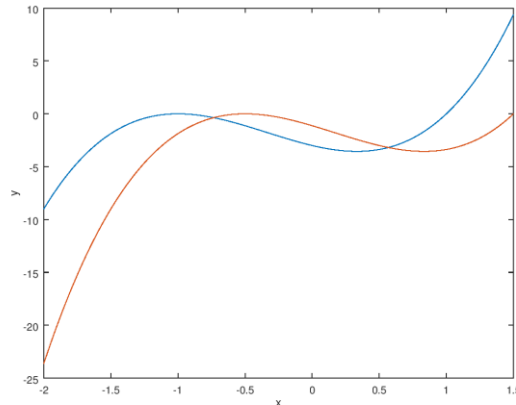
Die Funktion `meinpolynom()` wird nun durch Ergänzung eines weiteren Eingabeparameters erweitert:

In der Schule haben Sie gelernt, dass die mathematische Funktion $f(x - a)$ gegenüber der Funktion $f(x)$ auf der x -Achse um den Wert a nach rechts verschoben ist. Die Funktion `meinpolynom` soll nun so erweitert werden, dass die Verschiebung a zusätzlich übergeben werden kann. Das Ergebnis ist `meinpolynom2`, die Änderungen sind **rot** hervorgehoben:

```
function y = meinpolynom2(x, a)  
% Berechnet das Polynom  
% y = 3*(x+1)^2*(x-1)  
% verschoben um a nach rechts  
x = x-a;  
y = 3*(x+1).^2.*(x-1);  
end
```

Zum Testen zeichnen Sie zwei Verläufe in einem Schaubild:

```
>> x = -2:0.01:1.5;
>> y0 = meinpolynom2(x,0);
>> y1 = meinpolynom2(x,0.5);
>> plot(x,y0,x,y1)
>> xlabel('x')
>> ylabel('y')
```



Innerhalb der Funktion ist nun die Variable **x** lokal, das heißt die Verschiebung um **a** mit dem Befehl **x = x-a**; wirkt sich nur innerhalb der Funktion aus, außerhalb gilt weiterhin die ursprüngliche Variable **x**.³⁶

Aufgabe 13: Funktion programmieren

In dieser Aufgabe werden Sie nochmals Ihr Skript **V1_A11_analysiere_Messung3.m** aus Aufgabe 11 erweitern. Speichern Sie das Skript unter **V1_A13_analysiere_Messung4.m**.

Betrachten Sie die Schaubilder der Geschwindigkeit und des Weges über der Zeit. Es sieht so aus, als ob die Messung bei einer konstanten Beschleunigung a des Fahrzeuges erfolgt ist. In Aufgabe 11 haben Sie bereits versucht, die Beschleunigung zu ermitteln. Bei einer konstanten Beschleunigung a aus dem Stillstand berechnet sich der zurückgelegte Weg s nach folgender Gleichung:

$$s(t) = \frac{1}{2} \cdot a \cdot t^2$$

Ihre Aufgabe ist es jetzt, die Annahme der konstanten Beschleunigung zu überprüfen, in dem Sie den gemessenen und den berechneten Weg in einem gemeinsamen Schaubild darstellen:

- Programmieren Sie die Funktion **s = weg(t, a)**, die für eine gegebene Beschleunigung **a** und einen Zeitvektor **t** den Weg nach der obigen Gleichung berechnet.³⁷
- Erweitern Sie im Skript den Plot des Weges über der Zeit so, dass neben den Messgrößen auch der berechnete Weg eingezeichnet wird.
- Passen gemessener und berechneter Weg zusammen?

³⁶ Wie in C stellt das Gleichheitszeichen den Zuweisungsoperator dar, die Zeile **x = x-a** ist daher möglich.

³⁷ Achten Sie auf eine korrekte Bezeichnung der Funktionsdatei und darauf, dass das Skript und die Funktion im selben Verzeichnis liegen.

7 Versuche zur Computerarithmetik

Auch wenn in der Vorlesung und vielen Büchern darauf hingewiesen wird, dass Computer zwar sehr schnell aber mitunter auch ungenau rechnen, verlassen sich die Anwender von Numerikprogrammen wie MATLAB / Octave gerne „blind“ auf die Ergebnisse. Da die von Taschenrechner oder Computer berechneten Ergebnisse immer korrekt waren, wirkt die Diskussion der Rechengenauigkeit wie ein weltfremdes Thema für Mathe- und Computer-nerds.

Leider ist dem nicht so: wenn Sie in Ihren Programmen – egal, ob Sie diese mit MATLAB oder einer anderen Programmiersprache wie C erstellen – die Beschränkungen der Rechengenauigkeit der Fließkommazahlen nicht berücksichtigen, kann es zu nicht vernachlässigbaren Fehlern kommen. Daher sollen in diesem Abschnitt einige Versuche zur Computerarithmetik durchgeführt werden. Da MATLAB mit 64 Bit IEEE-Fließkommazahlen rechnet, können Sie die dabei gewonnenen Erkenntnisse unverändert auch auf die Programmierung mit anderen Programmiersprachen wie C, Java oder Python übertragen.

7.1 Addition, Subtraktion und die Reihenfolge

Wenn Sie im Supermarkt einkaufen, ist der zu zahlende Preis unabhängig davon, in welcher Reihenfolge Sie die Waren auf das Band gelegt haben. Grund dafür ist das Kommutativgesetz der Mathematik: beim Addieren und Subtrahieren ist das Ergebnis unabhängig von der Reihenfolge der Summanden.

Um zu überprüfen, ob dies auch für alle Rechnungen mit dem Computer gilt, werden Sie im folgenden Versuch eine Berechnung ausführen, deren Ergebnis Sie problemlos zum Vergleich im Kopf rechnen können.

Gegeben sind zwei Zahlen a und b . Daraus soll die folgende Summe berechnet werden:

$$s = a + b - a$$

Aus der Mathematik ist bekannt, dass Sie

1. dasselbe Ergebnis erhalten, wenn Sie die Summanden in einer anderen Reihenfolge anordnen: $s = a + b - a = a - a + b = b + a - a = b - a + a = \dots$
2. die Gleichung vereinfachen können: $s = b$.

Aufgabe 14: Addition, Subtraktion und die Reihenfolge

Diese Aufgabe wird während des Labortermins bearbeitet!

Schreiben Sie ein kleines Skript `V1_A14_Add_Sub_Reihenfolge.m`, mit dem Sie die Summe für verschiedene Werte a und b berechnen können.

- Löschen Sie zu Beginn alle Variablen mit `clear`
- Definieren Sie zwei Variablen `a` und `b`.
- Berechnen Sie die Summe in allen möglichen Reihenfolgen und geben Sie das Ergebnis formatiert (Formatstring `%g`) aus.
 - $a + b - a$
 - $-a + b + a$
 - $a - a + b$
 - $-a + a + b$
 - $b + a - a$
 - $b - a + a$

Führen Sie das Skript für die folgenden Werte für a und b aus:

- $a = 1, b = 10^{-16}$
- $a = 1, b = 10^{-15}$
- $a = 10^{16}, b = 1$

Diskutieren Sie die Ergebnisse basierend auf Ihrem Wissen aus der Vorlesung Rechnergestützte Mathematik.

7.2 Rundungsfehler – auch ohne große oder kleine Zahlen

Nach der Bearbeitung des Beispiels aus Abschnitt 7.1 sind Sie nun darauf sensibilisiert, dass Rundungsfehler in Abhängigkeit von der Reihenfolge der Rechnungen zu Fehlern führen können. Allerdings haben Sie dabei Zahlen addiert bzw. subtrahiert, die sich um viele Zehnerpotenzen unterscheiden. Daher kann man schnell zur Annahme kommen, dass die Rundungsfehler nur beachtet werden müssen, wenn man sehr große und/oder sehr kleine Zahlen einsetzt.

Leider ist dem nicht so: schon einige wenige einfache Rechnungen mit Zahlen im Bereich zwischen 0 und 1 können dazu führen, dass Ihr Programm nicht mehr das macht, was Sie eigentlich beabsichtigt haben.

Ein solches Programm werden Sie im nächsten Versuch erstellen. Die Aufgabe des Programms wird es sein, ausgehend vom Startwert $x = 1$ mehrmals einen Wert Δx zu subtrahieren. Sobald $x = 0$ erreicht ist, soll Ihr Programm die Meldung **NULL!** ausgeben. Das klingt nicht schwierig und in C könnten Sie dieses Programm auch problemlos schreiben. In der MATLAB-Sprache fehlen Ihnen dafür jedoch noch zwei Sprachkonstrukte:

- Eine kopfgesteuerte **while-Schleife**, die dafür sorgt, dass nur solange Δx subtrahiert wird, wie $x > 0$ ist.
- Eine **if-Abfrage**, die die Meldung **NULL!** ausgibt, wenn $x = 0$ ist.

Die entsprechenden MATLAB-Befehlen **while** und **if** funktionieren im Prinzip genauso, wie Sie es von C kennen. Die etwas andere Syntax wird in der MATLAB/Octave Kurzübersicht und – mit Beispielen – in der sehr ausführlichen MATLAB-Hilfe erläutert.

Sowohl bei der **while-Schleife** als auch bei der **if-Abfrage** muss eine boolesche Bedingung programmiert werden. Hierzu stellt MATLAB die in Tabelle 8 zusammengestellten Operatoren zur Verfügung:

Tabelle 8: Vergleichsoperatoren und logische Operatoren

Mathe	=	\neq	>	\geq	<	\leq	UND	ODER	NICHT
MATLAB	==	~=	>	>=	<	<=	&		~

Aufgabe 15: Rundungsfehler

Diese Aufgabe wird während des Laborterminals bearbeitet!

Schreiben Sie ein kleines Skript **v1_A15_Rundungsfehler.m**, mit dem ausgehend von $x = 1$ solange $x > 0$ ist immer wieder der Wert Δx subtrahiert wird.

- Definieren Sie den Startwert $x = 1$
- Definieren Sie Δx
- Schreiben Sie eine while-Schleife, die solange $x > 0$ ist den Wert Δx subtrahiert
- Nach der Subtraktion geben Sie den aktuellen Wert von x formatiert (%g) aus.
- Prüfen Sie, ob $x = 0$ ist, und geben Sie dann den Text **NULL!** aus.

Testen Sie dann das Programm mit $\Delta x \in \{0,1; 0,125; 0,2; 0,25\}$ und diskutieren Sie das Ergebnis. Überlegen Sie sich auch einen Lösungsvorschlag zur Beseitigung des auftretenden Problems.

7.3 IEEE-Fließkommazahlen

Sie haben die IEEE-Fließkommazahlen in der Vorlesung Rechnergestützte Mathematik kennengelernt und selbst die Umrechnungen zwischen Fließkommazahl und Binärdarstellungen „von Hand“ vorgenommen. Damit Sie in Zukunft diese Rechnungen mit MATLAB / Octave überprüfen können, lernen Sie die hierfür notwendigen Befehle kennen.

7.3.1 Fließkommazahl in Hexadezimaldarstellung

Bei der Verwendung von MATLAB / Octave als Taschenrechner in Kapitel 4.2.1 haben Sie bereits mit dem Befehl **format** die Art der Ausgabe der Werte verändert. Nach Eingabe von

```
format hex
```

werden alle Größen im Hexadezimalformat ausgegeben:

```
>> pi
ans =
    400921fb54442d18
```

Ohne spezielle Vorgaben rechnet MATLAB / Octave mit 64 Bit IEEE-Fließkommazahlen. Entsprechend wird eine 16-stellige Hexadezimalzahl ausgegeben. Um eine 64 Bit Zahl als eine 32 Bit IEEE-Fließkommazahl darzustellen, muss diese zuerst in den Datentyp **single** umgewandelt werden:

```
>> single(pi)
ans =
    40490fdb
```

Durch Eingabe von **format** ohne weitere Argumente gibt MATLAB die Ergebnisse wieder als Fließkommazahl aus.

```
>> format
>> ans
ans =
    3.1416
```

Ohne die Ausgabe aller Zahlen umzuschalten erzeugt der Befehl **num2hex** einen String mit der Hexadezimaldarstellung der IEEE-Fließkommazahl:

```
>> num2hex(pi)
ans =
    400921fb54442d18

>> num2hex(single(pi))
ans =
    40490fdb
```

7.3.2 Hexadezimaldarstellung in Fließkommazahl

Für die Umwandlung des Strings der Hexadezimaldarstellung in eine 64 Bit IEEE-Fließkommazahl können Sie die Funktion **hex2num** verwenden:

```
>> hex2num('400921fb54442d18')
ans =
    3.1416
```

Leider funktioniert diese Funktion nicht mit 32 Bit IEEE-Fließkommazahlen. Hier erfolgt die Umwandlung in zwei Schritten:

1. Berechnung einer 32 Bit Integerzahl aus dem String:
xint = uint32(hex2dec(xhex));
2. Umwandeln der Integerzahl in eine 32 Bit IEEE-Fließkommazahl, ohne dabei die Bits zu verändern:
x = typecast(xint, 'single');

Vollständiges Beispiel:

```
>> xint = uint32(hex2dec('40490fdb'))
xint =
    1078530011

>> x = typecast(xint, 'single')
x =
    3.1416
```

7.3.3 Bequemes Umwandeln von 32 Bit IEEE-Fließkommazahlen

Aufgabe 16: Umrechnung von IEEE-Fließkommazahlen

Diese Aufgabe wird während des Laborterminals bearbeitet!

Um den Aufwand in Grenzen zu halten, wird in der Vorlesung Rechnergestützte Mathematik mit 32 Bit IEEE-Fließkommazahlen gerechnet. Schreiben Sie daher basierend auf den Erläuterungen in den Abschnitten 7.3.1 und 7.3.2 die beiden Funktionen

- **single2ieee**
- **ieee2single**

um diese Umrechnungen bequem vornehmen zu können. Testen Sie die Funktionen anhand einiger Beispiele aus der Vorlesung / dem Vorlesungsskript.

8 Lineare Algebra

Bereits bei der Einarbeitung in die MATLAB-Sprache haben Sie beim Thema Matrizen und Vektoren in Kapitel 5 gesehen, dass sich MATLAB ganz besonders für die Lösung von Aufgaben aus der linearen Algebra eignet. In Mathematik 1 haben Sie die dafür notwendigen mathematischen Grundlagen gelernt und festgestellt, dass zum Beispiel das Lösen von linearen Gleichungssystemen mit einem großen Rechenaufwand verbunden ist. Diese Rechenarbeit können Sie nun MATLAB bzw. Octave überlassen.

8.1 Lösung linearer Gleichungssysteme

8.1.1 Beispiel aus Mathematik 1

Die Lösung von linearen Gleichungssystemen kennen die meisten von Ihnen bereits aus der Schule. In der Vorlesung Mathematik 1 wurde unter anderem das folgende lineare Gleichungssystem betrachtet:

$$\begin{aligned}x + 2y - z &= 4 \\ -x + y + 3z &= -1 \\ 2x + 7y - z &= 11\end{aligned}$$

Sowohl für die Lösung von Hand mit dem Gaußalgorithmus als auch für die Bearbeitung mit MATLAB wird das Gleichungssystem in der folgenden Matrix-Vektor-Notation geschrieben:

$$\underbrace{\begin{pmatrix} 1 & 2 & -1 \\ -1 & 1 & 3 \\ 2 & 7 & -1 \end{pmatrix}}_A \cdot \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{\vec{x}} = \underbrace{\begin{pmatrix} 4 \\ -1 \\ 11 \end{pmatrix}}_{\vec{c}} \quad \Leftrightarrow \quad A \cdot \vec{x} = \vec{c}$$

In Mathematik 1 haben Sie dann die erweiterte Koeffizientenmatrix gebildet,

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 4 \\ -1 & 1 & 3 & -1 \\ 2 & 7 & -1 & 11 \end{array} \right)$$

und mit dem Gaußalgorithmus so lange umgeformt bis links die Einheitsmatrix steht:

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array} \right)$$

Auf der rechten Seite können Sie dann die Lösung ablesen:

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

In der MATLAB-Sprache formt der Befehl **rref** die erweiterte Koeffizientenmatrix solange um, bis links die Einheitsmatrix steht:

```
>> A = [1 2 -1; -1 1 3; 2 7 -1];  
>> c = [4; -1; 11];
```

```
>> EKM = [A c]
EKM =
     1     2    -1     4
    -1     1     3    -1
     2     7    -1    11
>> rref(EKM)
ans =
     1     0     0     2
     0     1     0     1
     0     0     1     0
```

Bei der Handrechnung kann man sich die Arbeit etwas erleichtern, indem man den Gaußalgorithmus nur solange anwendet, bis links eine obere Dreiecksmatrix steht

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 4 \\ 0 & 3 & 2 & 3 \\ 0 & 0 & -1 & 0 \end{array}\right)$$

und dann die Lösung schrittweise durch Lösen der einzelnen Gleichungen berechnet.

Die Umformung mit dem Gaußalgorithmus funktioniert auch dann, wenn das LGS keine oder unendlich viele Lösungen hat:

Keine Lösung	Unendlich viele Lösungen
$\left(\begin{array}{ccc c} 1 & 2 & -1 & 4 \\ -1 & 1 & 3 & -1 \\ 2 & 7 & 0 & 10 \end{array}\right)$ <pre>>> A = [1 2 -1; -1 1 3; 2 7 0]; >> c = [4; -1; 10]; >> EKM = [A c]; >> rref(EKM) 1.0000 0 -2.3333 2.0000 0 1.0000 0.6667 0.0000 0 0 0 1.0000</pre> <p>Widerspruch in der letzten Zeile: $0 = 1$</p>	$\left(\begin{array}{ccc c} 1 & 2 & -1 & 4 \\ -1 & 1 & 3 & -1 \\ 2 & 7 & 0 & 11 \end{array}\right)$ <pre>>> A = [1 2 -1; -1 1 3; 2 7 0]; >> c = [4; -1; 11]; >> EKM = [A c]; >> rref(EKM) 1.0000 0 -2.3333 2.0000 0 1.0000 0.6667 0.0000 0 0 0 0</pre> <p>Letzte Zeile: $0 = 0$</p>

8.1.2 Lösung der Matrix-Vektor-Gleichung

Wenn das lineare Gleichungssystem

$$A \cdot \vec{x} = \vec{c}$$

eine eindeutige Lösung hat, kann es durch einfache Umformungen nach dem gesuchten Vektor \vec{x} aufgelöst werden. Hierzu werden beide Seiten von links³⁸ mit A^{-1} multipliziert.

³⁸ Bei der Matrizenrechnung ist die Multiplikation nicht kommutativ, das heißt es kommt auf die Reihenfolge an.

$$\underbrace{A^{-1} \cdot A}_E \cdot \vec{x} = A^{-1} \cdot \vec{c}$$

Das Produkt $A^{-1} \cdot A$ ist gleich der Einheitsmatrix E . Mit $E \cdot \vec{x} = \vec{x}$ kann damit der gesuchte Vektor wie folgt berechnet werden:

$$\boxed{\vec{x} = A^{-1} \cdot \vec{c}}$$

Diese Rechnung ist nur dann möglich, wenn es eine eindeutige Lösung \vec{x} für das lineare Gleichungssystem gibt. Dies ist – wie aus der Vorlesung Mathematik 1 bekannt – dann der Fall, wenn $\det(A) \neq 0$ ist.

Aufgabe 17: Lösung des linearen Gleichungssystems

Diese Aufgabe wird während des Labortermins bearbeitet!

Schreiben Sie ein Skript `v1_A17_LGS_loesen.m`, das die Lösung des obigen linearen Gleichungssystems berechnet:

- Geben Sie die Matrix **A** und den Vektor **c** ein.
- Prüfen Sie durch Berechnung von $\det(A)$, ob die Matrix invertierbar ist.
- Falls ja (if-Abfrage) berechnen Sie den Lösungsvektor $\vec{x} = A^{-1} \cdot \vec{c}$.
- Nutzen Sie dabei beide in Abschnitt 5.1 vorgestellten Methoden zur Inversion der Matrix. Was erwarten Sie bezüglich Rechenaufwand bei diesen beiden Verfahren?
- Neben den beiden Methoden aus Abschnitt 5.1 können Sie die Lösung auch mit $\mathbf{x} = \mathbf{A} \backslash \mathbf{c}$ berechnen. Setzen Sie auch diese Methode um und schauen Sie in der Hilfe nach (`doc mldivide`), was dieser Befehl macht.
- Überprüfen Sie, ob die Lösung stimmt, indem Sie $A \cdot \vec{x}$ berechnen.

8.2 Beispiel: Widerstandsnetzwerk

8.2.1 Aufgabenstellung

Als Beispiel für die Lösung linearer Gleichungssysteme wird die in Abbildung 6 gezeigte elektrische Schaltung untersucht.

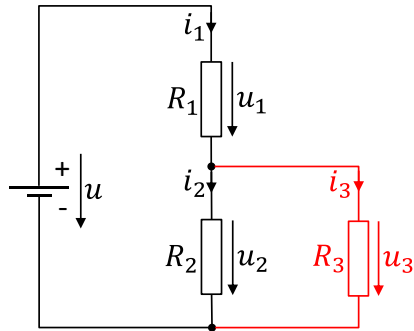


Abbildung 6: Schaltplan

Eine Batterie mit der konstanten Spannung u ist mit einem Spannungsteiler aus den beiden Widerständen R_1 und R_2 verbunden. Parallel zu R_2 wird zusätzlich das in rot in den Schaltplan eingezeichnete Heizelement R_3 angeschlossen. Für die Auslegung des Heizelements stellen sich die folgenden Fragen:³⁹

- Wie hängt die elektrische Leistung $P_3 = R_3 \cdot i_3^2$ des Heizelements von der Wahl des Widerstands R_3 ab?
- Für welchen Widerstandswert R_3 wird die Heizleistung P_3 maximal?

8.2.2 Herleitung der Gleichungen

Um die Fragen beantworten zu können, muss die Leistung $P_3 = R_3 \cdot i_3^2$ in Abhängigkeit des noch frei wählbaren Widerstands R_3 und den gegebenen Widerständen R_1 und R_2 berechnet werden. Hierzu wird der Schaltplan aus Abbildung 6 in ein mathematisches Modell überführt. Grundlage hierfür sind die aus der Elektrotechnik bekannten Kirchhoffschen Regeln:

- Linke Masche $-u + u_1 + u_2 = 0 \quad \Leftrightarrow \quad u_1 + u_2 = u$
- Rechte Masche $-u_2 + u_3 = 0$
- Knoten $i_1 = i_2 + i_3 \quad \Leftrightarrow \quad i_1 - i_2 - i_3 = 0$

³⁹ Studierende der Elektrotechnik/Informationstechnik kennen diese Aufgabe in ähnlicher Form aus dem Elektrotechnischen Grundlagenlabor.

Ziel ist die Berechnung der Leistung $P_3 = R_3 \cdot i_3^2$ und damit des Stroms i_3 in Abhängigkeit der Spannung u sowie der Widerstände R_1 , R_2 und R_3 . Damit kann der Widerstandswert R_3 bestimmt werden, für den die Leistung $P(R_3)$ maximal wird.

Es werden die Spannungen u_1 , u_2 und u_3 mithilfe des Ohmschen Gesetzes in Abhängigkeit der Ströme berechnet:

$$u_1 = R_1 \cdot i_1, \quad u_2 = R_2 \cdot i_2, \quad u_3 = R_3 \cdot i_3$$

Durch Einsetzen in die drei Kirchhoffschen Gleichungen erhält man ein lineares Gleichungssystem, das in eine Matrix-Vektor-Notation überführt werden kann:

$$\begin{array}{rcl} R_1 \cdot i_1 + R_2 \cdot i_2 & = & u \\ -R_2 \cdot i_2 + R_3 \cdot i_3 & = & 0 \\ i_1 - i_2 - i_3 & = & 0 \end{array} \Leftrightarrow \underbrace{\begin{pmatrix} R_1 & R_2 & 0 \\ 0 & -R_2 & R_3 \\ 1 & -1 & -1 \end{pmatrix}}_R \cdot \underbrace{\begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix}}_{\vec{i}} = \underbrace{\begin{pmatrix} u \\ 0 \\ 0 \end{pmatrix}}_{\vec{u}}$$

8.2.3 Ermittlung des optimalen Widerstandswertes

Der Zusammenhang zwischen den Strömen \vec{i} , den Widerständen R und der Spannung \vec{u} wird somit durch das lineare Gleichungssystem

$$\boxed{R \cdot \vec{i} = \vec{u}}$$

beschrieben. Zur Berechnung der Leistung am Widerstand R_3 muss daher

- das Gleichungssystem nach \vec{i} aufgelöst,
- der Strom i_3 aus dem Lösungsvektor \vec{i} entnommen,
- und dann damit die Leistung $P_3 = R_3 \cdot i_3^2$ berechnet werden.

Aufgabe 18: Berechnung der Leistung**Diese Aufgabe wird während des Labortermins bearbeitet!**

Es werden jetzt folgende Parameter für den in Abbildung 6 schwarz eingezeichneten Teil der Schaltung gewählt:

$$u = 10 \text{ V}, \quad R_1 = 10 \, \Omega, \quad R_2 = 15 \, \Omega$$

Programmieren Sie eine Funktion **P3 = Leistung(R3)**, die die Leistung P_3 abhängig vom Widerstand R_3 berechnet:

- Erzeugen Sie die Koeffizientenmatrix R und den Vektor \vec{u} innerhalb der Funktion
- Lösen Sie das Gleichungssystem nach \vec{i} auf.
- Berechnen Sie die Leistung $P_3 = R_3 \cdot i_3^2$.

Schreiben Sie dann ein Skript **v1_A18_Auswertung_Leistung.m**, das für Widerstandswerte

$$0 \, \Omega \leq R_3 \leq 30 \, \Omega$$

die Leistung P_3 am Widerstand R_3 berechnet und in einem Schaubild grafisch darstellt.

Hinweis:

Da der Widerstandswert R_3 in die Matrix R eingetragen wird, können Sie im Unterschied zu den bisherigen Funktionen keinen Vektor mit mehreren Werten für R_3 an die Funktion **Leistung** übergeben. Es kann daher eine **for-Schleife** programmiert werden:

- Machen Sie sich mithilfe der MATLAB / Octave Kurzreferenz [4] und der MATLAB-Hilfe [5] mit der Syntax der for-Schleife vertraut.
- Definieren Sie für die Widerstandswerte den Vektor **R3array = 0:0.1:30**
- Das Array **P3array** zum Speichern der Leistungswerte initialisieren Sie mit dem Befehl **zeros** mit einer entsprechenden Anzahl von Nullelementen und füllen es dann in der for-Schleife mit den berechneten Werten.

Ermitteln Sie dann anhand der Grafik den optimalen Widerstandswert für R_3 und die damit erzielte Leistung.

8.2.4 Numerische Optimierung

Statt den optimalen Widerstandswert aus dem Schaubild abzulesen, kann man auch ein numerisches Verfahren verwenden, dass für die gegebene Funktion **Leistung** das Maximum sucht. Numerische Verfahren zur Suche nach Extrempunkten werden in der Vorlesung Rechnergestützte Mathematik nicht behandelt. Zur Suche nach einem **Minimum** gibt es in MATLAB / Octave die Funktion

fminsearch(@fun, x0)

@fun Handle auf eine MATLAB-Funktion **y = fun(x)**

- **x** ist die Eingangsgröße der Funktion (hier der Widerstand R_3)
- **y** ist die skalare Ausgangsgröße (hier die Leistung P_3)

x0 Startwert der Suche

Zur Lösung der gegebenen Aufgabe soll statt einem Minimum das Maximum der Funktion **Leistung** gefunden werden. Hierzu wird zuerst eine Hilfsfunktion programmiert, die genau dann minimal wird, wenn die Leistung P_3 maximal ist.

Aufgabe 19: Numerische Optimierung

Diese Aufgabe wird während des Laborterminals bearbeitet!

Programmieren Sie eine Funktion **Leistung_min**, die abhängig vom Widerstand R_3 genau dann minimal wird, wenn die Leistung P_3 maximal ist.

Erweitern Sie dann Ihr Skript **V1_A19_Auswertung_Leistung.m** zur Suche nach dem Minimum. Mit dem Startwert $R_3 = 10 \Omega$ (versuchen Sie auch andere Startwerte) müssen Sie dafür die Zeile

R3opt = fminsearch(@Leistung_min, 10);

ergänzen. Geben Sie das Ergebnis formatiert aus.

9 Funktionen in der Anwendung

In MATLAB können Funktionen mehrere Ein- und Ausgabeparameter besitzen.⁴⁰

Mit der folgenden Aufgabe können Sie Ihre Erfahrung im Umgang mit Funktionen vertiefen. Dort, wo es sinnvoll ist, sollen innerhalb einer Funktion Arrays statt Skalare als Funktionsparameter übergeben werden, um weniger Schleifen beim Funktionsaufruf einzusetzen.

Aufgabe 20: Schiefer Wurf

In Tabelle 9 sind die Formeln angegeben, mit denen die in Abbildung 7 dargestellte Bewegung des schiefen Wurfs einer Kugel aus der Höhe s_0 durch zweidimensionale Vektoren physikalisch beschrieben wird.⁴¹

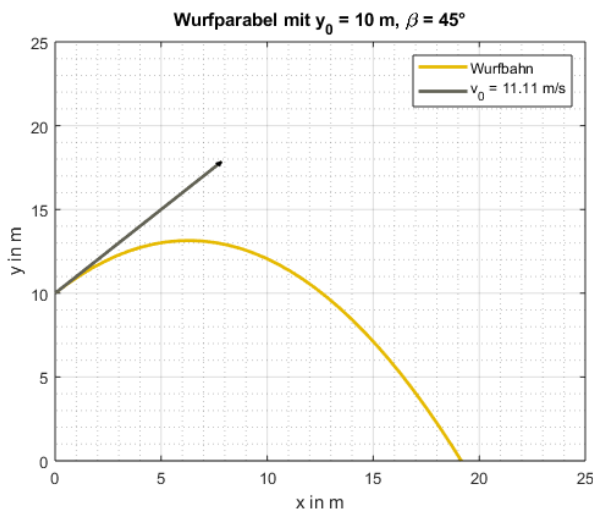


Abbildung 7: Schiefer Wurf aus Höhe y_0

Tabelle 9: Formeln zur Beschreibung des schiefen Wurfes

Beschleunigungsvektor	Geschwindigkeitsvektor	Ortsvektor
$\mathbf{a}(t) = \begin{pmatrix} 0 \\ -g \end{pmatrix}$	$\mathbf{v}(t) = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$ $= \begin{pmatrix} v_0 \cos(\beta_0) \\ v_0 \sin(\beta_0) - g \cdot t \end{pmatrix}$	$\mathbf{r}(t) = \begin{pmatrix} x \\ y \end{pmatrix}$ $= \begin{pmatrix} v_0 \cdot t \cdot \cos(\beta_0) \\ v_0 \cdot t \cdot \sin(\beta_0) - \frac{1}{2} \cdot g \cdot t^2 + y_0 \end{pmatrix}$
Wurfparabel $y(x) = x \cdot \tan(\beta_0) - \frac{g}{2 \cdot v_0^2 \cdot (\cos(\beta_0))^2} \cdot x^2 + y_0$		Größen $g \approx 9,81 \frac{\text{m}}{\text{s}^2}$ Erdbeschleunigung v_0 Anfangsgeschwindigkeit in m/s β_0 Abwurfwinkel in rad t Zeit in s y_0 Anfangshöhe in m

⁴⁰ Beim Aufruf von Funktionen mit mehreren Ausgabeparametern werden die Variablen, denen die Ausgabeparameter zugewiesen werden, in []-Klammern übergeben. Stimmt die Anzahl der Variablen nicht mit der Anzahl der Ausgabeparameter überein, so gehen beim Funktionsaufruf Informationen verloren.

⁴¹ Größen, die in MATLAB als Array angelegt werden, sind dabei fettgedruckt.

- a) Markieren Sie in Tabelle 9 die Formelabschnitte, in denen bei der Implementierung elementweise Operatoren zum Einsatz kommen.
- b) Vervollständigen und implementieren Sie die Funktion **geschwindigkeitsvektor()**, welche x- und y-Komponenten des Vektors $\mathbf{v}(t)$ berechnet und zurückgibt.

```
function [vx,vy] = geschwindigkeitsvektor(v0, beta0,t)
%geschwindigkeitsvektor berechnet x- und y-Komponente des
%Geschwindigkeitsvektors für alle Zeitpunkte des Vektors t
% Eingabeparameter: v0:   Anfangsgeschwindigkeit in m/s
%                   beta0:Abwurfwinkel in rad
%                   t:    Vektor mit Zeitpunkten in s
% Ausgabeparameter: vx:   x-Komponente Geschwindigkeitsvektor
%                   vy:   y-Komponente Geschwindigkeitsvektor
% Autor, Datum
...
end
```

- c) Schreiben Sie eine kommentierte Funktion **grad2rad()**, mit der Winkelangaben im Gradmaß (Grad) ins Bogenmaß (Radiant) umgerechnet werden können.⁴² Diese Funktion können Sie im weiteren Verlauf der Aufgabe einsetzen.
- d) Erstellen Sie ein Skript **V1_A20_SchieferWurf.m**. In diesem Skript werden alle Funktionsaufrufe der nächsten Teilaufgaben erfasst. Verwenden Sie **%%Abschnitte**, um das Skript zu strukturieren.
- Rufen Sie **geschwindigkeitsvektor()** mit folgenden Werten auf:
 $v_0 = 36 \frac{\text{km}}{\text{h}}, \beta_0 = 45^\circ$ für $0 \leq t \leq 10\text{s}$ mit einer Schrittweite von $0,1\text{ s}$.
Weisen Sie das Ergebnis des Funktionsaufrufs den Variablen **v_x** und **v_y** zu.⁴³
 - Nutzen Sie den Befehl **size()**, um die Anzahl der Zeilen und Spalten des Zeitvektors, sowie der Arrays **v_x** und **v_y** zu bestimmen. Machen Sie sich hierzu mit den verschiedenen Syntaxoptionen von **size()** vertraut.
 - sz = size(A)**,
 - szdim = size(A,dim)**
 - [m,n] = size(A)**
- Wieso existieren mehrere Syntaxoptionen, um die Größe einer Matrix zu bestimmen?

⁴² Zusatz: Prüfen Sie, ob der eingegebene Winkel im Intervall $[0, 360^\circ)$ liegt (**doc if, doc Short-circuit && ||**). Ist dies nicht der Fall, nutzen Sie den Modulo-Operator, um den Winkel in dieses Intervall zu verschieben (**doc mod**). Die Modulo Funktion in Matlab kann auch auf negative Operanden angewendet werden. Für die vorliegende Aufgabe ist physikalisch gesehen eine Einschränkung des Winkels auf $[0, 90^\circ)$ sinnvoll.

⁴³ Die lokalen Variablennamen innerhalb der Funktion können natürlich von den Namen der Variablen, die beim Funktionsaufruf genutzt werden, abweichen.

- Zeichnen Sie die y-Komponente des Geschwindigkeitsvektors in Abhängigkeit vom Zeitvektor in eine beschriftete Abbildung.
 - Zusatz: Zeichnen Sie die x-Komponente des Geschwindigkeitsvektors in Abhängigkeit vom Zeitvektor in eine beschriftete Abbildung.⁴⁴
- e) Vervollständigen und implementieren Sie die MATLAB-Funktion `ortsvektor()`, mit der x- und y-Komponente des Ortsvektors der Wurfparabel bestimmt und in zwei Parametern zurückgegeben werden. Überprüfen Sie die Aussagekraft des Kommentars, indem Sie die Hilfe zu dieser Funktion aufrufen.

```
function [ , ] = ortsvektor( )
%ORTSVEKTOR berechnet x- und y-Komponente des Ortsvektors bei einem
% schiefen Wurf
% Eingabeparameter:  v0:      Anfangsgeschwindigkeit in m/s
%                   beta0:   Abwurfwinkel in rad
%                   y0:      Anfangshöhe in m
%                   t:       Vektor mit Zeitschritten in s
%
% Ausgabeparameter:  rx : x-Komponente des Ortsvektors
%                   ry : y-Komponente des Ortsvektors
% Autor, Datum

g = 9.81; % Erdbeschleunigung in m/s^2
rx = ; %x-Komponente
ry = ; %y-Komponente des Ortsvektors
end
```

- f) Rufen Sie `ortsvektor()` in einem neuen Abschnitt Ihres Skripts `V1_A20_SchieferWurf.m` auf, um die Ortskurve eines schiefen Wurfs für $y_0 = 0 \text{ m}$, $v_0 = 30 \frac{\text{m}}{\text{s}}$, $\beta_0 = 60^\circ$ und $0 \text{ s} \leq t \leq 5,5 \text{ s}$ zu berechnen. Plotten Sie die Ortskurve in eine beschriftete Abbildung mit geeigneter Schrittweite. In welchem Bereich ist die Darstellung der berechneten Daten physikalisch sinnvoll?
- g) Der Funktionsaufruf von `geschwindigkeitsvektor()` wird nun in Kontrollstrukturen eingebunden.

```
v0g = 4; beta0g = 30; tg = 0.1:0.2:0.5;
v = zeros(2,size(tg,2));
[v(1,:),v(2,:)] = geschwindigkeitsvektor(v0g,grad2rad(beta0g), tg);
for k = 1:size(v,2)
    if (v(2,k) > eps)
        fprintf('Bewegung nach oben bei t = %1.2f s\n', tg(k));
    elseif (v(2,k) < eps)
        fprintf('Bewegung nach unten bei t = %1.2f s\n', tg(k));
    else
```

⁴⁴ Aus einem Skalar können Sie einen Vektor erzeugen, indem Sie eine Matrix mit 1-Einträgen mit dem gewünschten Skalar multiplizieren.

```
    fprintf('Bewegung horizontal bei t = %1.2f s\n', tg(k));  
end  
end
```

- Wie werden hier die Rückgabeparameter der Funktion gespeichert?
- Welche Ausgabe erzeugt obiger Code in der Standardausgabe?
- Erweitern Sie die formatierte Ausgabe mit `fprintf()` so, dass eine Ausgabe der folgenden Form erreicht wird:

Bewegung nach oben bei $t = \text{X.XX}$ s mit $v_x = \text{X.XX}$ m/s und $v_y = \text{X.XX}$ m/s

Literatur

- [1] Frank Thueselt, Felix Paul Gennrich: Praktische Mathematik mit MATLAB, Scilab und Octave für Ingenieure und Naturwissenschaftler, Springer Verlag, 2013.
Als E-Book verfügbar!
- [2] Frank Thueselt: Das Arbeiten mit Numerik Programmen MATLAB, Scilab und Octave in der Anwendung. Beiträge der Hochschule Pforzheim, Nr. 129, 2009.
Download von der Laborseite.
- [3] Rainer Dietz: Skript zur Vorlesung Rechnergestützte Mathematik, 2016.
Download von der Vorlesungsseite.
- [4] Stefan Hillenbrand: MATLAB / Octave Kurzreferenz.
Download von der Laborseite.
- [5] MATLAB Online-Hilfe.
Verfügbar auf de.mathworks.com – Support – Documentation
- [6] Lothar Papula: Mathematik für Ingenieure und Naturwissenschaftler, Band 1, Springer Verlag, 2014.
Als E-Book verfügbar!
- [7] Lothar Papula: Mathematik für Ingenieure und Naturwissenschaftler, Band 2, Springer Verlag, 2015.
Als E-Book verfügbar!

Hinweis zu den E-Books

Die Hochschule hat zahlreiche Lehrbücher insbesondere vom Springer-Verlag abonniert, die Sie kostenlos herunterladen können. Weitere Informationen zu den Ebooks finden Sie auf den Webseiten der Bibliothek:

<https://www.hs-pforzheim.de/hochschule/organisation/bibliothek/>

Die hier genannten E-Books können Sie auch direkt bei Springer herunterladen:

1. Verbinden Sie sich mit dem Hochschulnetz:
 - Rechner der Hochschule oder
 - Eigener Rechner über VPN
2. Rufen Sie <http://link.springer.com> auf.
3. Suchen Sie nach dem Buch.
4. Abonnierte Bücher können Sie kapitelweise oder vollständig als PDF herunterladen.