

Lösen technisch- mechanischer Problemstellungen

mit den Computer- Algebra- Systemen:
Matlab und Mathematica

Projektbetreuer: Prof. Dr. Dieter Hackenbracht

Bearbeiter: Michael Kaufmann, René Timm

Frankfurt, 2015

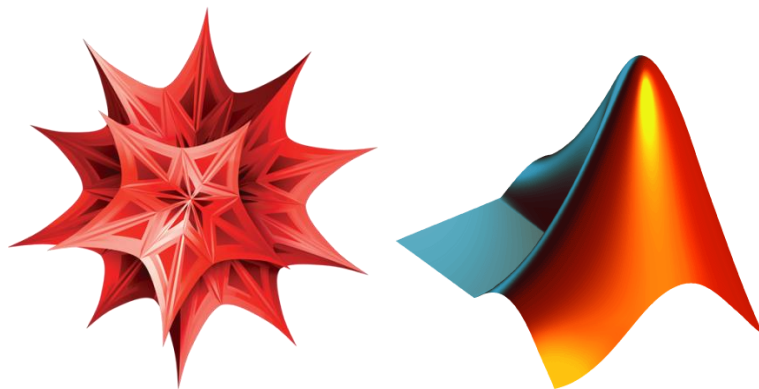


Abbildung 1: Logo(vlnr): Mathematica¹, Matlab²

¹ http://xahlee.info/comp/Mathematica_logo/Mathematica_8_logo.png

² http://www.mathworks.com/cmsimages/64848_wl_cc_logo_membrane_2002_wl.gif

Abstract

Um effizient am Arbeitsplatz mathematische Problemstellungen zu lösen, sind die beiden Computer-Algebra-Systeme Matlab und Mathematica geeignete Werkzeuge. Schließlich lassen sich mit ihnen komplexe Problematiken beschreiben. Auch die „Zettelwirtschaft“ nimmt dadurch deutlich ab. Rechnungen, die sich teilweise über mehr als zwei Seiten erstrecken, lassen sich so durch eine einzige Funktion innerhalb von wenigen Minuten erledigen. Aber nicht nur die Mathematik kommt auf ihre Kosten, denn man kann mit den Programmen auch Graphen, Grafiken und Darstellungen erstellen und diskutieren.

Doch welches dieser beiden Programme sollte man denn nun verwenden? Um diese Frage zu beantworten, haben wir beide Systeme verglichen und bewertet. Anhand mehrerer Aufgabenstellungen rund um die Themen: „Elastostatik“, „Kinetik“ und „Technische Schwingungslehre“ fühlten wir so der Software auf den Zahn.

Die technische Mechanik hat so ihre Tücken. Oftmals „versteht man nur Bahnhof“ oder muss viel Zeit für den Verständnisaufbau investieren. Mit Hinweisen und niedergeschriebenen Gedankengängen während der Bearbeitung der technischen Aufgabenstellungen laden wir die Leserin und den Leser ein, den einen oder anderen Blick in dieses Projekt zu riskieren. Warum werden periodische Erregungen in der Schwingungslehre über Fourier-Reihen dargestellt? Wie war das nochmal mit der Symmetrie? Was sind die Hintergründe beim allseits beliebten „Föppln“ im Modul TMII-Elastostatik?

Auch wenn keine Vorkenntnisse über die beiden Computer-Algebra-Systeme vorhanden sind, lassen sich auf schnellem Weg seitenlange Berechnungen mithilfe der hier vorgestellten Funktionen überprüfen. Wer schon einmal das Eigenverhalten eines Zwei-Masse-Schwingers per Hand berechnet hat, wird hier verstehen was wir meinen.

Studenten eines Ingenieurstudienganges, unentschlossene CAS-Benutzer, neugierige Menschen, die ihre Kenntnisse erweitern möchten, aber auch Ingenieure in der Berufswelt sowie Lehrende sind unsere Zielgruppen.

Danksagung

Bedanken möchten wir uns bei Herrn Prof. Dr. Dieter Hackenbracht, welcher uns das Projekt überhaupt möglich gemacht hat. Des Weiteren bedanken wir uns für seine Unterstützung bei der Programmierung mit Mathematica.

Ein weiterer Dank geht an Herrn Prof. Dr. Ulrich Becker. Dieser hat uns bei Fragen und Problemen mit Matlab zur Seite gestanden.

Ferner danken wir den Professoren aus der technischen Mechanik, da wir die Aufgaben direkt aus deren Modulen bereitgestellt bekamen.

Inhaltsverzeichnis

Danksagung	2
Abkürzungsverzeichnis	4
Abbildungsverzeichnis	5
Tabellenverzeichnis.....	8
Einführung.....	9
Motivation	10
Einarbeitung in die Programme	11
Mathematica	11
Matlab.....	13
Aufgabenauswahl	14
Mathematica	15
Aufgabe 1 – TMII – belasteter, gelagerter Balken	15
Aufgabe 2 – Fourier.....	20
Aufgabe 3 – Technische Schwingungen	24
Aufgabe 4 – TMII – Belastete, elastische Stäbe.....	29
Aufgabe 5 – TMIII – Impuls-, Energie-, Arbeitssatz	34
Aufgabe 6 – TMIII – Kinematische Bindung.....	38
Matlab.....	41
Aufgabe 1 – TMII – belasteter, gelagerter Balken	41
Aufgabe 2 – Fourier.....	45
Aufgabe 3 – Technische Schwingungen	48
Aufgabe 4 – TMII – Belastete, elastische Stäbe.....	50
Aufgabe 5 – TMIII – Impuls-, Energie-, Arbeitssatz	53
Aufgabe 6 – TMIII – Kinematische Bindung.....	55
Auswertung	57
Vergleich	58
Bewertung.....	61
Fazit – Schlusswort	62
Literatur	64
Anhang	65

Abkürzungsverzeichnis

CAS	Computer- Algebra- System
TMII	Technische Mechanik 2 - Elastostatik
TMIII	Technische Mechanik 3 - Kinetik
TS	Technische Schwingungslehre
LGS	Lineares Gleichungssystem

Abbildungsverzeichnis

Abbildung 1: Logo (vlnr.): Mathematica, Matlab	0
Abbildung 2: Bereichsweise definierte Funktion	15
Abbildung 3: Föppl-Klammer in Mathematica	15
Abbildung 4: Plot mit dem Befehl "Unitstep"	16
Abbildung 5: Plott mit dem Befehl "HeavisideTheta"	16
Abbildung 6: Plott mit dem Befehl "Piecewise"	17
Abbildung 7: Zelle mit Fehler	17
Abbildung 8: Plott der Querkraft mit Fehler	18
Abbildung 9: Zelle ohne Fehler.....	18
Abbildung 10: Plott der Querkraft ohne Fehler	19
Abbildung 11: Bereichsweise definierte Funktion	20
Abbildung 12: Darstellung mit UnitStep	20
Abbildung 13: Anwendung der Modulus Funktion	20
Abbildung 14: Plot der Modulus Funktion	21
Abbildung 15: FourierTrigSeries	21
Abbildung 16: Funktion für FourierTrigSeries	22
Abbildung 17: Berechnen der Eigenkreisfrequenz	22
Abbildung 18: Plot von FourierTrigSeries und Modulus-Funktion.....	22
Abbildung 19: Manipulate	23
Abbildung 20: Partielle Ableitung mit Fehler durch Indizes	24
Abbildung 21: Ableitungspunkte	25
Abbildung 22: Ableitungsstriche	25
Abbildung 23: Partielle Ableitung von "phi"	25
Abbildung 24: Zeitliches zuweisen	26
Abbildung 25: Differentiation nach der Zeit.....	26
Abbildung 26: MatrixForm	27
Abbildung 27: Rechnen mit Matrizen	27
Abbildung 28: Determinante	28
Abbildung 29: Eigenvektoren	28
Abbildung 30: Drawing Tool	29
Abbildung 31: Toggle Alignment Guides	29
Abbildung 32: Referenz	30

Abbildung 33: Kreis zeichnen.....	30
Abbildung 34: Plot ohne Option	31
Abbildung 35: AxesLabel	31
Abbildung 36: AspectRatio.....	32
Abbildung 37: PlotStyle und Filling	32
Abbildung 38: Plot-Funktion mit Optionen	32
Abbildung 39: Plot mit Optionen	32
Abbildung 40: Solve.....	34
Abbildung 41: Solve eines LGS.....	34
Abbildung 42: Liste von Listen.....	35
Abbildung 43: Solve - Liste	35
Abbildung 44: Ergebnis raus holen.....	35
Abbildung 45: Ergebnis rausgeholt.....	36
Abbildung 46: Listenbaum.....	36
Abbildung 47: Listenbaum.....	36
Abbildung 48: Beispiel einer komplexen Verschachtelung.....	37
Abbildung 49: Gleichungssystem nach 5 Unbekannten lösen.....	38
Abbildung 50: Gleichungen minimieren und auflösen	39
Abbildung 51: Anwendung der Matrixschreibform.....	39
Abbildung 52: Föppl-Symbolik aus TMII	41
Abbildung 53: Föppl-Realisierung in Matlab	41
Abbildung 54: Plot einer Funktion.....	42
Abbildung 55: Zwei Plots in einem.....	42
Abbildung 56: Subplot	43
Abbildung 57: Plot mit Zusätzen	43
Abbildung 58: Mehrere Sections.....	45
Abbildung 59: Fourier-Reihe	45
Abbildung 60: Rechteckfunktion mit Modulo Befehl.....	46
Abbildung 61: Symbolische Funktion.....	46
Abbildung 62: Nicht Symbolische Funktion.....	47
Abbildung 63: Matrixschreibweise	48
Abbildung 64: Rechnen mit Matrizen.....	48
Abbildung 65: Determinate	49
Abbildung 66: Eigenverhalten	49

Abbildung 67: Matlab	50
Abbildung 68: clc, clear	50
Abbildung 69: Workspace.....	51
Abbildung 70: subs.....	52
Abbildung 71: disp.....	52
Abbildung 72: Bestimmtes Integral	53
Abbildung 73: Unbestimmtes Integral	53
Abbildung 74: Differentiation.....	54
Abbildung 75: Partielle Differentiation.....	54
Abbildung 76: “assume” anhand einer Aufgabe	54
Abbildung 77: Mit solve Nullstellen gefunden	55
Abbildung 78: Lösung eines LGS	55
Abbildung 79: Vereinfachung.....	56
Abbildung 80: Hübsche Darstellung	56

Tabellenverzeichnis

Tabelle 1: Aufgabenauswahl	14
Tabelle 2: Plotoptionen	44
Tabelle 3: Vergleichstabelle	59
Tabelle 4: Bewertung	60
Tabelle 5: Ergebnis.....	60
Tabelle 6: Endergebnis.....	60

Einführung

Es gibt einige Veröffentlichungen über die Einführung und das Einarbeiten in Mathematica und Matlab. Diese Literatur beschäftigt sich mit verschiedenen Funktionen und allgemeinen Grundkenntnissen, nicht aber mit dem spezifischen Lösen mechanischer Aufgaben. Deshalb bauen wir auf die allgemeine Einleitung der Systeme auf und geben nur einen kurzen Einblick über die verschiedenen Einarbeitungsmöglichkeiten an.

Im Anschluss beschreiben wir die Auswahl der Aufgaben und erläutern den Grund dieser Auswahl. Es ist uns wichtig, dass wir ein breites Spektrum aus den Modulen TMII, TMIII und TS abdecken. Genauso wichtig ist es, die Aufgaben nicht nur programmiertechnisch, sondern auch verständlich für die Hörer dieser Module zu gestalten. Daher folgen immer jeweils vor, nach und zwischen dem Programmieren gedankliche Einschübe. Am Schluss jeder Aufgabe geben wir ein Fazit, um damit auf die Kernproblematiken der jeweiligen Aufgabe aufmerksam zu machen.

Bearbeitet werden 12 Aufgaben: sechs Aufgaben in Mathematica und die gleichen sechs Aufgaben in Matlab.

So können die Programme miteinander verglichen werden:

Welches Programm eignet sich besser für die technische Mechanik?

Kann man sich überhaupt festlegen, welches System sich besser eignet?

Gibt es von Aufgabe zu Aufgabe Unterschiede in der Eignung?

In welches Programm konnte man sich schneller Einarbeiten?

Sind ausreichend Medien zur Einarbeitung vorhanden?

Wer sind die Zielgruppen?

Wo sind die Systeme verbreitet?

Welches Programm nützt mir in meiner Zukunft mehr?

Motivation

Während unseres Studiums sind wir weniger mit Projekt-Aufgaben konfrontiert worden, somit bietet sich durch dieses Projekt eine hervorragende Chance uns weiter zu entwickeln.

Ursprünglich wollten wir mechanische Problemstellungen mit Mathematica ausdrücken. Doch nach einem Gespräch mit Herrn Prof. Dr. Stefan Dominico ist uns klar geworden, dass nicht nur Mathematica für die Mechanik in Frage kommen kann, sondern auch das schon etablierte CAS Matlab. Uns kam die Idee nach einem Vergleich der beiden Programme in Hinblick auf Einarbeitung, Verwendung, Verbreitung und Aktualität.

Unser selbst gewähltes Studien-Projekt bringt einige Herausforderungen mit sich, welchen wir uns stellen möchten. Als Ausgangssituation haben wir uns für das Projekt entschieden, da wir oft mit Problemen der technischen Mechanik konfrontiert wurden. Die Verständnisschwierigkeiten zogen sich von der Elastostatik über die Kinetik bis hin zur technischen Schwingungslehre.

Wir wollen den Leserinnen und Lesern die Anwendung der Programme im Bereich der technischen Mechanik zeigen und ihnen die Möglichkeit geben sich für ein System zu entscheiden.

Wir streben zwei gemeinsame Ziele an: eigene Erfahrungen mit den beiden Programmen gewinnen und einen Wegweiser für unsere Kommilitonen schaffen. Wir beide wissen selbst, dass die Kernfächer Technische Mechanik zu den kräftezehrendsten Fächern zählen. Durch unsere Arbeit wollen wir Kommilitonen das Bewältigen von Problemen veranschaulichen.

Am Ende der Studien-Arbeit soll für jede Interessierte und jeden Interessierten klar sein, welches Programm das richtige für sich ist.

In der Dokumentation wollen wir verschiedene Lösungsansätze für die Programmierung geben und am Ende die Anwendungsmöglichkeiten diskutieren, um eine solide Aussage geben zu können.

Einarbeitung in die Programme

Mathematica

Vor dem Projekt benutzten wir wolframalpha.com um Graphen zu plotten und Ergebnisse zu überprüfen. Dort ist es möglich ohne Programmierkenntnisse zum Ergebnis zu gelangen.

Die Verbindung zwischen wolframalpha.com und Mathematica half uns bei den ersten Programmierversuchen. Die von der Freeform umgewandelte Eingabe bot uns den ersten Einblick in die Programmierung von Mathematica.

Den Einstieg rundete das von Wolfram bereitgestellte *Hands-on Start to Mathematica*³ ab.

Die Devise lautete “learning by doing”, wir haben parallel zu den Video-Tutorials die Übungen mit dem Programm nachgeahmt. Für das weitere Vorgehen bedienten wir uns der Literatur und der Internetseite youtube.com⁴.

Mit Hilfe der verwendeten Medien erlernten wir die Grundlagen, wie man Zellen evaluiert, Funktionen korrekt notiert und den Hinweis auf das Mathematica interne Helpcenter.

Weitere Informationen konnten aus den Medien nicht gezogen werden, da unsere mechanischen Aufgabenstellungen spezifische Funktionen benötigen.

Um eine Struktur zur Lösung der Aufgaben darzustellen, erstellten wir ein Musterbeispiel. Wir bauten uns die Aufgabe nach dem Prinzip auf, wie wir sie damals per Hand in unserem Studium lösten. Im Laufe der Bearbeitung der Musteraufgabe stießen wir auf diverse Probleme.

Bei Fragestellungen, welche wir über unsere Einarbeitungsmedien nicht lösen konnten, griffen wir auf Foren im Internet zurück. Die Hinweise und auch gerechneten Beispiele in den Foren sind gute Hilfestellungen.

Letztendlich ist es eine Frage des Benutzers, welches Einarbeitungsmedium effektiver ist.

Sowohl Videos, Literatur und Foren konnten uns zum Vollenden der Musteraufgabe nicht ausreichend helfen. Da Mathematica sehr viele umfangreiche Funktionen besitzt, ist es notwendig sich zunächst mit deren Anwendung vertraut zu machen. Durch das Mathematica

³ <http://www.wolfram.com/broadcast/screencasts/handsonstart/>

⁴ <https://www.youtube.com/watch?v=D4IcCkON13o>

Hilfcenter konnten wir uns benötigte Funktionen beschaffen. Der letzte Schritt war die Absprache mit Herrn. Prof. Dr. Hackenbracht. Dieser gab uns abschließende Hinweise und Verbesserungsvorschläge.

Mit dem Beenden der Musteraufgabe, in der wir auch die Struktur und die Formatierung klärten, starteten wir in die Programmierung der sechs Aufgaben. Bis zum Abschluss der Aufgaben lernten wir immer wieder neue Funktionen kennen und vertieften das bisher Gelernte.

Matlab

Durch die gewonnenen Programmierkenntnisse von Mathematica ging die Einarbeitung in das CAS Matlab schneller. Eine Reihe von Funktionen besitzt in beiden Programmen den gleichen Namen. Somit reichte es bis zu einem gewissen Grad aus, nur die Syntax zu ändern. Für alles weitere bedienen wir uns Tutorial-Videos und Einführungen im PDF-Format aus dem Internet.

Auch diesmal klärten wir im Vorfeld durch Bearbeiten der ersten Aufgabe die Struktur, um eine einheitliche Bearbeitung zu erzielen.

Im Matlab internen Hilfecenter konnten wir einige Hinweise heraus ziehen und durch einen Besuch bei Herrn Prof. Dr. Becker erlangten wir den letzten Schliff.

Ohne Vorkenntnisse durch andere Computer Algebra Systeme bietet das Internet durch Videos, PDF-Einführungen und Foren gute Quellen zur Einarbeitung. Des Weiteren gibt es in der Bibliothek einige Literatur zum Thema Einführung in Matlab.

Aufgabenauswahl

Wir haben aus den Modulen TMII, TMIII und TS einige Übungs- und Klausuraufgaben ausgewählt. Wir achteten darauf, dass sich kein Aufgabentyp wiederholt und somit wesentliche Bereiche der Module abgedeckt sind.

Modul	Verweis	Kurzbeschreibung
TMII Elastostatik	- Klausuraufgabe SS02	Stab: Stabkräfte, Stabverlängerung, Verschiebungsplan
TMII Elastostatik	- Klausuraufgabe II/2 WS2010	Biegelinie, Durchbiegung am freien Ende, Querkraft- und Momentenverlauf, Stelle des größten Schnittmomentes
TMIII Kinetik	- Klausuraufgabe D/2	Kinematische Bindung von 2 Rollen: Relation der Bewegungen, Beschleunigung, Reibungskoeffizient
TMIII Kinetik	- Klausuraufgabe B/2	Impuls-, Arbeits-, Energiesatz: Geschwindigkeiten an verschiedenen Punkten berechnen
Technische Schwingung	-	Verständnis für die Fourier-Reihe aufbauen, Plotten von immer höher werdenden mehreren harmonischen bis zur Fourier-Reihe, Symmetrie, allgemeine Vorgehensweise
Technische Schwingung	Übungsblatt 1, Aufgabe 3	MFHG : Lagrange 2. Gleichung aufstellen, Eigenkreisfrequenzen, Eigenvektoren

Tabelle 1: Aufgabenauswahl

Mathematica

Aufgabe 1 – TMII – Belasteter, gelagerter Balken

In dieser Aufgabe ist die Problematik das **Föppl**-Symbol. Das aus der Elastostatik beliebte Symbol zur Darstellung von Mehrfeldbalken ist in Mathematica mit der Funktion “**UnitStep**” realisierbar. UnitStep ist der Einheitssprung.

$$\langle x - a \rangle^n = \begin{cases} 0 & \text{für } x < a \\ (x - a)^n & \text{für } x > a \end{cases}$$

Abbildung 2: Bereichsweise definierte Funktion

Mit $n = 0$ folgt für die Föppl-Klammer $\langle x - a \rangle^0$:

$$\langle x - a \rangle^0 = 0 \text{ für } x < a, \quad \langle x - a \rangle^0 = 1 \text{ für } x > a.$$

Das Gleiche macht auch UnitStep. Aber das ganze sieht etwas anders aus.

$$\langle x - a \rangle^0 * \text{UnitStep}[x - a]$$

Abbildung 3: Föppl-Klammer in Mathematica

Mit dieser Notation kann man in Mathematica das Föppl-Symbol anwenden:

$$\langle x - a \rangle^n = (x - a)^n \text{UnitStep}[x - a].$$

In der Abbildung haben wir die Funktion “Unitstep” benutzt, um einen Sprung darzustellen. UnitStep erzeugt den Einheitssprung, das heißt ein Sprung um “1”. In der eckigen Klammer sagt man der Funktion wann der Einheitssprung an bzw. ausgeschaltet ist. Sobald der Ausdruck in der Klammer negativ oder gleich Null entspricht, wird die Funktion “ausgeschaltet”. Ergibt der Ausdruck ein positives Ergebnis, springt die Funktion um “1” und ist somit angeschaltet.

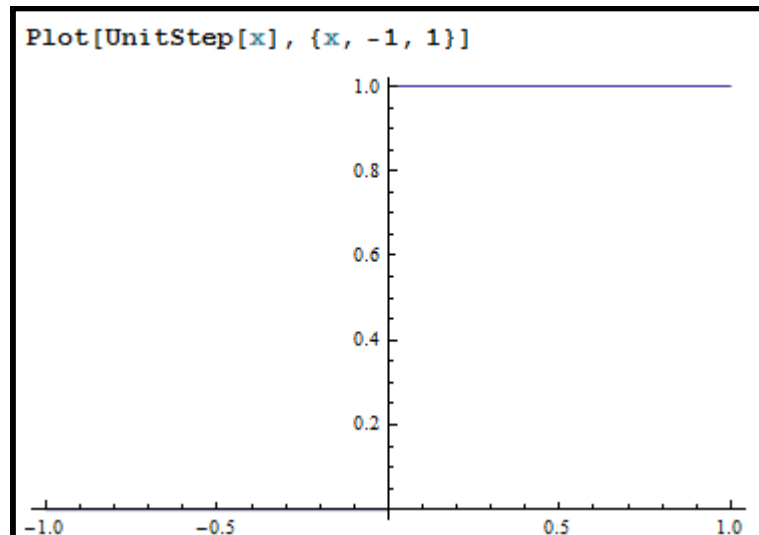


Abbildung 4: Plott mit dem Befehl "Unitstep"

Weitere Möglichkeiten zur Realisierung des Föppl-Symbols in Mathematica:

Auf der eine Seite gibt es den Befehl "HeavisideTheta". Die Funktion HeavisideTheta ist im Grunde das Gleiche wie der Befehl UnitStep, der sogenannte Einheitssprung.

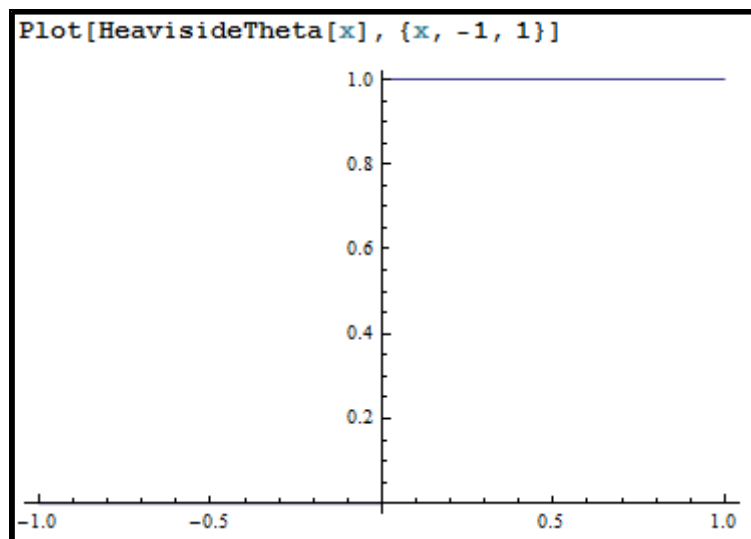


Abbildung 5: Plott mit dem Befehl "HeavisideTheta"

Auf der anderen Seite gibt es den Befehl "Piecewise".

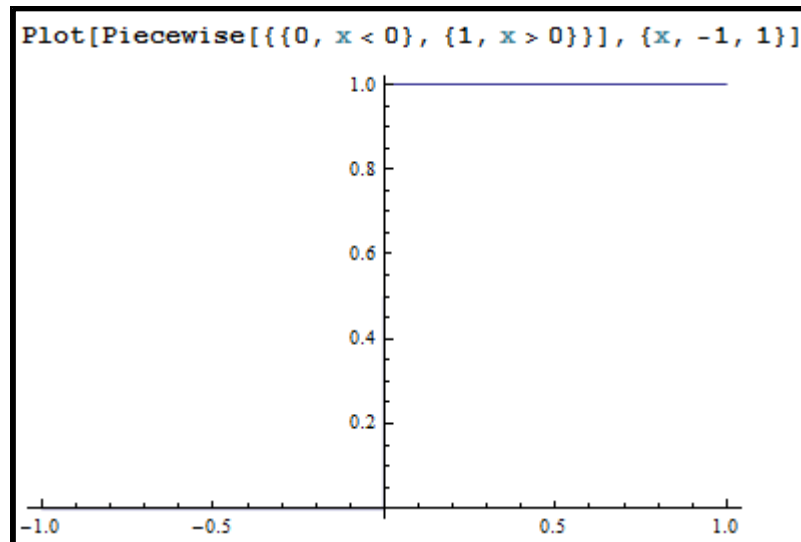


Abbildung 6: Plott mit dem Befehl "Piecewise"

Mit der Funktion Piecewise erstellt man eine stückweise definierte Funktion. Aber die so definierte Funktion lässt sich nicht integrieren, daher ist sie nutzlos für uns.

Ob man nun HeavisideTheta oder UnitStep benutzt spielt für uns hier keine Rolle. Wir haben uns eben für UnitStep entschieden.

EXKURS: Verborgenes Zeichen

Ein weiteres Problem tauchte bei dem Plotten der Funktionen im Aufgabenteil c) auf.

Ein verborgenes Zeichen sorgte dafür, dass beim Plotten Q ab $x = 2$ nicht angezeigt wurde. Daher untersuchten wir den Fehler beim Berechnen der Lagerkraft B mit dem Funktionsbefehl **FullForm**.

Er gibt Aufschluss über die Darstellung in den Ausdrücken.

Hier taucht der Ausdruck "TextCell" auf.

```
(* Output-Zelle mit Fehler *)
FullForm[B]
Times[Rational[3, 2], a, q0, TextCell[""]]
```

Abbildung 7: Zelle mit Fehler

Somit kann Mathematica den Befehl nicht auswerten.

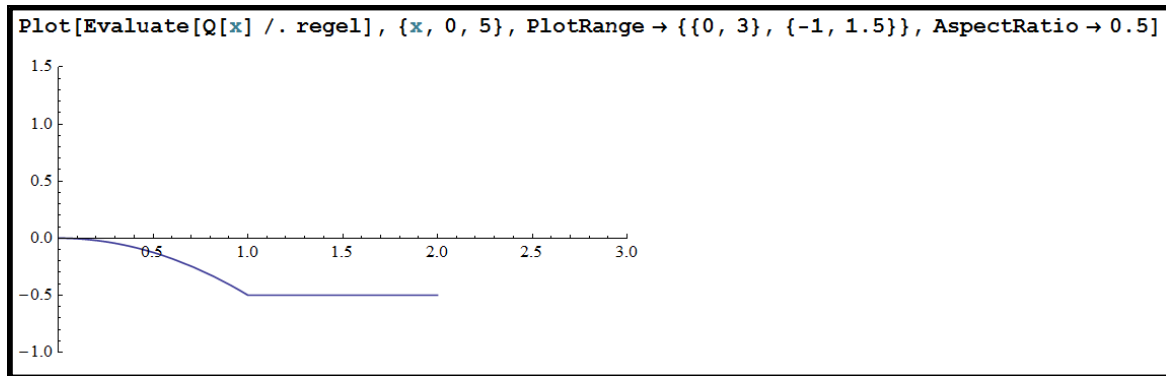


Abbildung 8: Plott der Querkraft mit Fehler

Im Graphen kann man erkennen, dass Q für $x > 2$ nicht ausgewertet wird.

An der Stelle $x=2$ sollte ein Sprung durch die Lagerkraft "B" auftreten.

Nachdem wir den Befehl der Zelle erneut eingegeben haben, ist dadurch der Fehler beseitigt und der Graph kann vollständig geplottet werden.

```
(* Output-Zelle ohne Fehler *)
FullForm[B]
Times[Rational[3, 2], a, q0]

$$\frac{3 a q_0}{2}$$

```

Abbildung 9: Zelle ohne Fehler

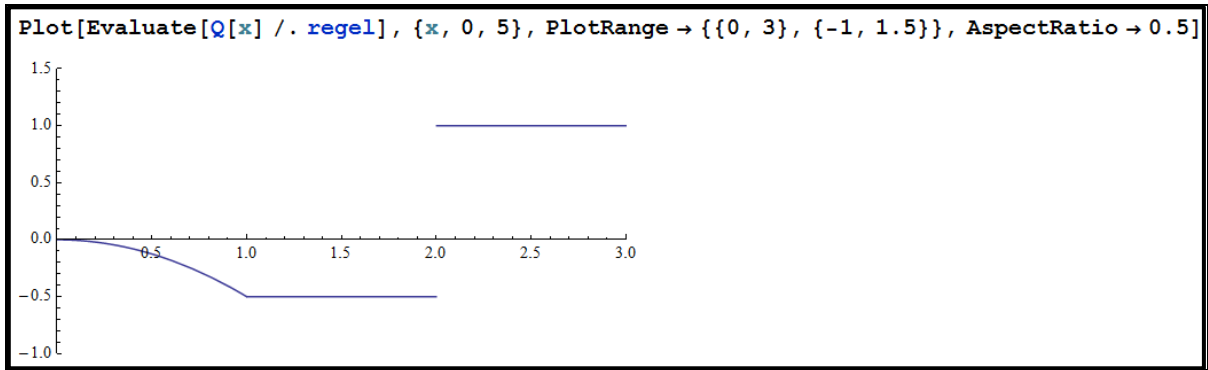


Abbildung 10: Plott der Querkraft ohne Fehler

Aufgabe 2 – Fourier

In der technischen Schwingung ist zum Aufstellen einer Fourier-Entwicklung oft eine bereichsweise definierte Funktion in einem Grundintervall vorgegeben., z.B.:

$f(x) = -1$ für $-1 < x \leq 0$, $f(x) = +1$ für $0 < x \leq 1$. Über das Grundintervall hinaus ist die Funktion periodisch fortgesetzt. Dies könnte man in Mathematica mit dem Befehl Piecewise durchführen:

```
fp[x_] = Piecewise[{{-1, -1 < x ≤ 0}, {1, 0 < x ≤ 1}}, "Periodisch fortgesetzt"]
{
-1                                -1 < x ≤ 0
1                                0 < x ≤ 1
Periodisch fortgesetzt True
```

Abbildung 11: Bereichsweise definierte Funktion

Aber damit kann man nicht weiterrechnen. Daher gehen wir anders vor.

Zuerst wird die gegebene Funktion durch zwei separate Funktionsvorschriften erfasst: die erste Vorschrift gilt für $x < 0$, die zweite für $x > 0$. Die Verwendung von UnitStep (statt einfach -1 bzw. +1 zu nehmen) geschieht schon im Hinblick auf die periodische Fortsetzung im nächsten Schritt.

```
f1[x_] := -UnitStep[x + 1]
f1[x_] := UnitStep[x]
```

Abbildung 12: Darstellung mit UnitStep

Eine in einem Grundintervall definierte Funktion kann in Mathematica mit “Modulus” periodisch fortgesetzt werden. Die Modulus-Funktion generiert den Funktionswert an beliebiger Stelle durch Rückführung auf das Grundintervall.

```
f1[x_] := -UnitStep[(Mod[x, p] - p) + 1] /; Mod[x, p] ≥ p / 2
f1[x_] := UnitStep[Mod[x, p]] /; Mod[x, p] < p / 2
```

Abbildung 13: Anwendung der Modulus Funktion

$/; \text{Mod}[x, p] < \frac{p}{2}$ bedeutet: “Diese Funktionsvorschrift wird nur dann benutzt, wenn $\text{Mod}[x, p] < \frac{p}{2}$.”

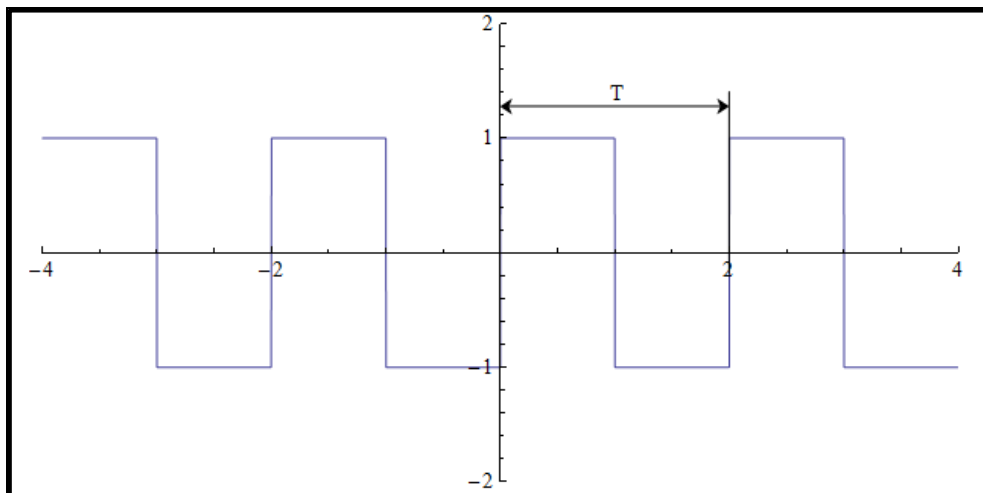


Abbildung 14: Plot der periodisch fortgesetzten Funktion

Die Funktionsweise der Modulus Funktion lässt sich durch die Frage: “Woher kennt Mathematica den Funktionswert an jeder beliebigen Stelle?” und der folgenden Erklärung beschreiben. Von einer beliebigen Stelle $x > 0$ wird die Periodendauer abgezogen (bei negativem x wird addiert), bis der Wert im Grundintervall, hier $(-1,1]$, liegt. Dieser Wert entspricht dem Wert an der beliebigen Startstelle. Somit ist jede Stelle definiert und kann gezeichnet werden.

Aber es gibt einen Haken: Mathematica kann damit nicht rechnen und somit dient die Überlegung hier nur zur Darstellung periodisch fortgesetzter Funktionen.

Zum Aufstellen der Fourier-Reihe kann die Funktion **FourierTrigSeries** helfen.

Mit der Funktion `FourierTrigSeries` berechnet Mathematica die Fourier-Reihe bis zur gewünschten Ordnung.

```
fr4[x_] = FourierTrigSeries[f2[x], x, 7, FourierParameters -> {1, ω}]
```

Abbildung 15: `FourierTrigSeries`

In diesem Beispiel wurde die Fourier-Reihe bis zur 7. Ordnung berechnet. Wie bereits beschrieben ist es nicht möglich die Fourier-Reihe für die periodisch fortgesetzte Funktion zu

berechnen. Aber da ist auch gar nicht notwendig, denn es reicht dazu ja die Angabe der zu entwickelnden Funktion im Grundintervall, die wir so beschrieben haben:

```
f2[x_] := -UnitStep[x + 1] + 2 UnitStep[x]
```

Abbildung 16: Funktion für FourierTrigSeries

Anschließend wird mit Hilfe der Periodendauer die Kreisfrequenz berechnet, welche über die **FourierParameters** einbezogen wird. Dies dient der korrekten Darstellung, denn somit kennt Mathematica das Intervall, in welchem entwickelt werden soll.

$$\omega = 2 \pi / p$$

Abbildung 17: Berechnen der Kreisfrequenz

In der nächsten Abbildung ist der Plot der 7. Ordnung und der periodisch fortgesetzten Funktion zu sehen.

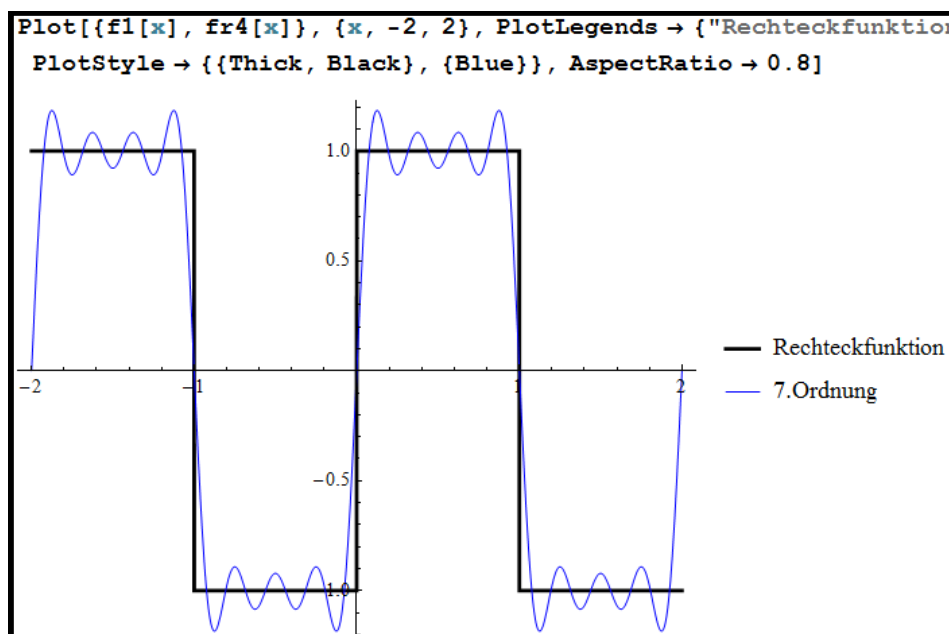


Abbildung 18: Plot von FourierTrigSeries und Modulus-Funktion

Als letzten Punkt in dieser Aufgabe behandeln wir den Umgang mit **Manipulate**. Mit diesem Befehl kann man in Echtzeit eine Funktion manipulieren. Als Beispiel nehmen wir x^n . Die Variable n wird manipuliert.

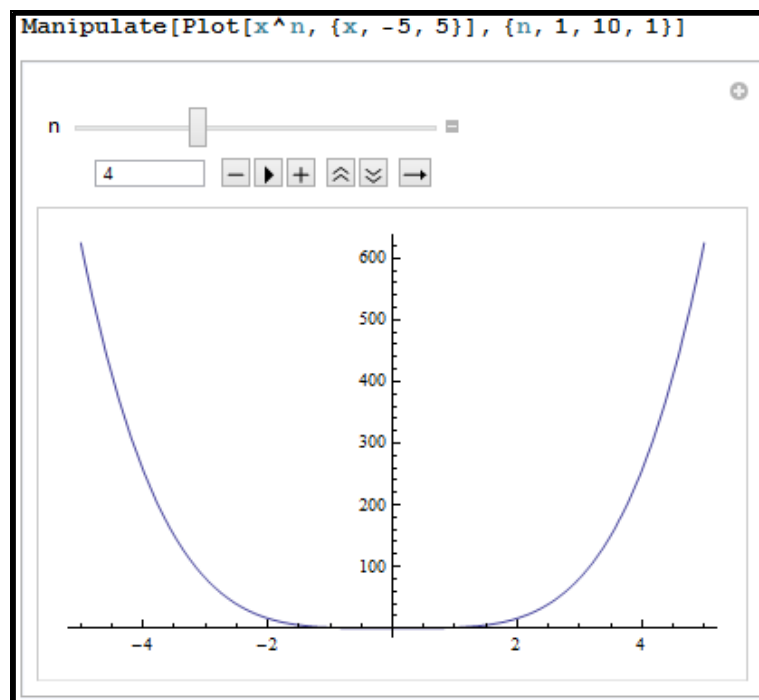


Abbildung 19: Manipulate

In Manipulate stehen die Plot-Funktion und der Hinweis, dass n manipuliert werden soll. Der Manipulationsbereich erstreckt sich von 1 bis 10 und ist geteilt in Einer-Schritten ($n, 1, 10, 1$). Im Notebook zur Fourier-Reihe kann man die Ordnung der Entwicklung manipulieren.

Aufgabe 3 – Technische Schwingungen

Zu Beginn der Aufgabe haben wir für die potentielle sowie kinetische Energie beide Gleichungen aufgestellt. Dabei sind wir auf die ersten Probleme gestoßen. Mathematica erkennt den Ableitungsstrich bei den Variablen “x” und “φ” nicht. Dadurch entsteht beim Differenzieren ein Fehler. In der unten aufgeführten Abbildung verwenden wir die Lagrangesche Gleichung 2. Art. Im zweiten Schritt (bei “a01”) benutzen wir die partielle Differentiation. Durch Eingabe des Großbuchstabens “D” startet man die Funktion des Differenzierens. Dort gibt man die Funktion ein, in diesem Fall “L”. Zuletzt wählt man die Variable aus, nach der partiell differenziert werden soll. Mathematica hat die Symbolik, die in der Abbildung 20 gezeigt wird, missverstanden.

$$\begin{aligned}
 L[\underline{x}, \varphi] &= E_k[\underline{x}, \varphi] - E_p[\underline{x}, \varphi] \\
 &= -\frac{1}{2} k_1 r^2 \varphi^2 - \frac{1}{2} k_2 (-x + r \varphi)^2 + \frac{1}{2} m (\dot{x})^2 + \frac{1}{2} \Theta (\dot{\varphi})^2 \\
 a01 &= D[L[\underline{x}, \varphi], \varphi] \\
 &= -k_1 r^2 \varphi - k_2 r (-x + r \varphi) + \Theta (0 \& \dot{\varphi})
 \end{aligned}$$

Abbildung 20: Partielle Ableitung mit Fehler durch Indizes

Hier ist nun der Fehler aufgetreten. Der letzte Teil der Lösung, welcher rot umrahmt ist, müsste “null” ergeben. Mathematica kann die Indizes nicht verarbeiten und “φ” wird als weitere unbekannte Variable angesehen.

Um das Ganze zu veranschaulichen folgen Beispiele.

Beispiel A:

Bei diesem Beispiel haben wir Punkte über den Variablen verwendet. Mathematica gibt den Befehl “OverDot” als Output heraus. Das bedeutet, dass der Befehl als Symbol erkannt wird. Der Punkt wird in der Regel zur Darstellung der Ableitung einer Größe nach der Zeit verwendet, da Mathematica dies nicht erkennt hilft es uns nicht weiter. Da wir für unser Beispiel einen Index für die Ableitung benötigen, bringt uns diese Möglichkeit nicht weiter.

```
A01[x_] := 6 x'' - 3 x' + 8
A02 = D[A01[x], x]
-3 OverDot(1,0)[x, 1] + 6 OverDot(1,0)[x, 2]
```

Abbildung 21: Ableitungspunkte

Beispiel B

In diesem Beispiel haben wir Ableitungsstriche benutzt. Mathematica kann die Ableitungsstriche nicht verarbeiten. Das Ergebnis “3” kommt dadurch zustande, da Mathematica beide Variablen gleichstellt und anschließend verrechnet. Die Anzahl von Ableitungsstrichen macht da keinen Unterschied. Das zweite Fallbeispiel hat ebenfalls nicht zur Lösung beigetragen.

```
B01[x_] := 6 x'' - 3 x' + 15
B02 = D[B01[x], x]
3 (0 &)
```

Abbildung 22: Ableitungsstriche

Um der Problematik entgegen zu wirken, haben wir ein “p” an den beiden Variablen “x” und “φ” beigefügt. Das “p” steht für die erste partielle Ableitung. Dadurch tritt kein Fehler beim Differenzieren mehr auf, da Mathematica nun versteht, dass “φp” und “φ” unterschiedliche Variablen sind.

```
L[x_, φ_, xp_, φp_] = Ek[xp, φp] - Ep[x, φ]
m xp^2 / 2 - 1/2 k1 r^2 φ^2 - 1/2 k2 (-x + r φ)^2 + Θ φp^2 / 2
Lφp = D[L[x, φ, xp, φp], φp]
Θ φp
```

Abbildung 23: Partielle Ableitung von “phi”

Nun geht es um das zeitliche Ableiten. Dort haben wir eine Zuordnung (Regeln) erstellt, um Mathematica klar zu machen, dass diese Variable letztlich auch von der Zeit abhängt (siehe Abb. 24).

```
Zeitabhängig = {xp → x' [t], φp → φ' [t]}
{xp → x' [t], φp → φ' [t]}
```

Abbildung 24: Zeitliches zuweisen

Daraufhin haben wir die “Regel” angewendet, welche in Mathematica durch “/.” ausgeführt wird. Das Programm erkennt die Ableitung und wendet die Regel an. Im zweiten Schritt wird das ganze zeitlich differenziert.

```
Lxpt[t_] = Lxp /. Zeitabhängig
m x' [t]
d01 = Lxpt' [t]
m x'' [t]
```

Abbildung 25: Differentiation nach der Zeit

Matrix:

In der technischen Schwingung werden Matrizen benutzt.

Um eine Matrix in Mathematica zu erstellen, kann man zum einen eine Liste von Listen erstellen, welche man anschließen mit dem Befehl **MatrixForm** in eine symbolische Matrix umwandelt. Zum andere gibt es über Typesetting vorgefertigte Matrix Darstellung. Verwendet man den Rechtsklick (insert Matrix/Table) kann man beliebig große Matrizen erstellen.

```

mat = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}

MatrixForm[mat]


$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$


```

Abbildung 26: MatrixForm

Das Rechnen mit Matrizen erfolgt ebenfalls recht einfach, welches wir an einem weiteren Beispiel klar machen. Um zum Beispiel eine Multiplikation eines Skalars mit einer Matrix durchzuführen, genügt es, das Skalar, in unserem Beispiel “3”, vor die Funktion zu schreiben (Mathematica erkennt, dass es eine Multiplikation ist). Anschließend berechnet Mathematica das Ergebnis.

```

mat1 =  $\begin{pmatrix} a & b \\ c & c \\ e & f \end{pmatrix}$ 

3 mat1 // MatrixForm


$$\begin{pmatrix} 3 a & 3 b \\ 3 c & 3 c \\ 3 e & 3 f \end{pmatrix}$$


```

Abbildung 27: Rechnen mit Matrizen

In unserem Beispiel benötigen wir die Determinante einer Matrix. Über den Befehl “**Det**” und die dazugehörige Funktion berechnet Mathematica die Determinante.

$$\begin{aligned}
 mM &= \begin{pmatrix} m & 0 \\ 0 & 1/2 * m * r^2 \end{pmatrix} \\
 kM &= \begin{pmatrix} k2 & -k2 * r \\ -k2 * r & k1 * r^2 + k2 * r^2 \end{pmatrix} \\
 m01 &= \text{Det}[-\omega^2 * mM + kM] \\
 &= k2^2 r^2 - \frac{5}{2} k2 m r^2 \omega^2 + \frac{1}{2} m^2 r^2 \omega^4
 \end{aligned}$$

Abbildung 28: Determinante

In der Aufgabenstellung ist nach dem Eigenverhalten der zwei Freiheitsgrade gefragt. In Mathematica benutzt man den Befehl “**Eigenvalues**”. Anschließend haben wir den Befehl in Form einer Liste eingegeben. Somit kann Mathematica das Ergebnis auswerten. Zuletzt haben wir den Listenausdruck in eine Matrixform dargestellt.

$$\begin{aligned}
 e01 &= -\omega^2 * mM + kM \\
 &= \left\{ \left\{ k2 + \frac{1}{2} (-5 k2 + \sqrt{17} k2), -k2 r \right\}, \left\{ -k2 r, 2 k2 r^2 - \frac{1}{4} (5 k2 - \sqrt{17} k2) r^2 \right\} \right\} \\
 e02 &= \text{Eigenvalues}[e01] \\
 &= \left\{ \left\{ \frac{1}{4} (3 r + \sqrt{17} r), 1 \right\}, \left\{ -\frac{-3 + \sqrt{17}}{2 r}, 1 \right\} \right\} \\
 \text{Eigenverhalten1} &= \text{MatrixForm}[e02[[1]]] \\
 &= \begin{pmatrix} \frac{1}{4} (3 r + \sqrt{17} r) \\ 1 \end{pmatrix}
 \end{aligned}$$

Abbildung 29: Eigenvektoren

Aufgabe 4 – TMII – Belastete, elastische Stäbe

In dieser Aufgabe behandeln wir die **Erstellung von Graphiken**, sowie **Optionen für Plots**. Mathematica bietet zur Erstellung von Graphiken ein Zeichnungs-Tool an. Damit besteht die Möglichkeit Formen wie Rechtecke, Kreise, Linien und Punkte zu erstellen. Des Weiteren kann man sich der Frei-Hand Funktion bedienen.

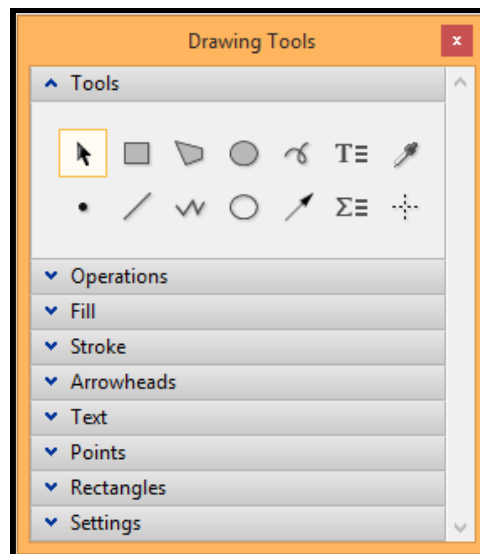



Abbildung 30: Drawing Tool

Wie in der Abbildung zu sehen ist, ist es auch möglich, extzellen, mathematische Zellen, Pfeile und gefüllte Formen einzufügen. Mit dem Fadenkreuz  kann man Koordinaten ablesen, welche für Programmierungen von Graphiken benötigt werden.

Das Programmieren von Graphiken werden wir in dem Projekt nicht behandeln.

Eine sehr nützliche Option dieses Tools befindet sich unter dem Reiter “Settings”.

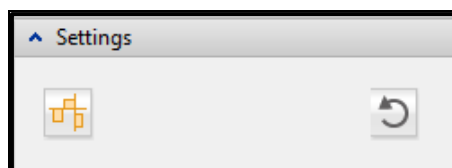


Abbildung 31: Toggle Alignment Guides

Man kennt diese Funktion unter anderem aus Microsoft Anwendungen (Word und PowerPoint). Jeder gezeichnete Strich, sowie der Mittelpunkt und die Mittelachsen werden als Raster fest gehalten und können als Referenz für neue Formen verwendet werden (siehe Abb. 31).

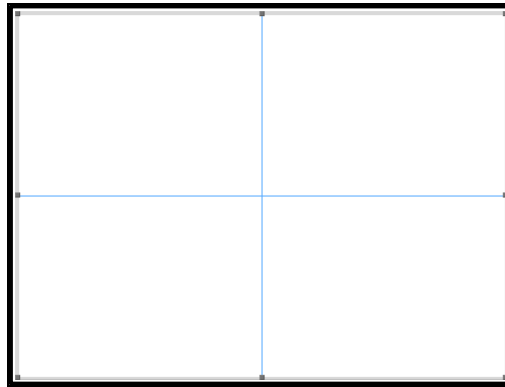


Abbildung 32: Referenz

Beim Zeichnen eines Punktes wird die Mitte der Zeichnungsebene automatisch erfasst (siehe Abb. 32).

Unter dem Reiter “Operations” kann man mehrere Formen zu einer Gruppe fügen, die Referenzen feiner einstellen, sowie einzelne Formen oder Gruppen in der Ebene nach vorn oder hinten verschieben.

Unter allen anderen Reitern sind Optionen für Strichstärke, Strichfarbe, Füllfarbe, Punktgröße, Schriftart, Schriftfarbe, usw. zu finden.

Als Beispiel und gleichzeitigem Hinweis erstellen wir einen Kreis (siehe Abb. 33).

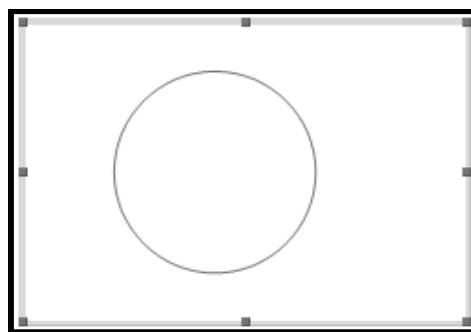


Abbildung 33: Kreis zeichnen

Ein Kreis zu erstellen scheint nicht schwer zu sein. Linke Maustaste gedrückt halten und Kreis aufspannen. Aber wie bekommt man die Ellipse nun rund? Die Lösung auf die Frage ist: Mit gedrückter Shift-Taste lassen sich Kreise, Quadrate und senkrechte bzw. waagerechte Striche erzeugen.

Optionen für Plotts

Ein Graph muss vollständig sein. Die Achsenbeschriftungen dürfen nicht fehlen und auch die Skala, sowie die Auflösung müssen ordentlich gewählt sein. Deshalb folgen hier Optionen für Plots.

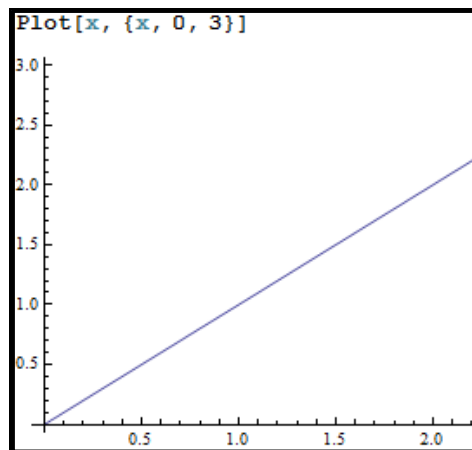


Abbildung 34: Plot ohne Option

Als erstes benennen wir die Achsen, die waagerechte nennen wir x und die senkrechte y (siehe Abb. 35).

```
AxesLabel -> {"x", "y"}
```

Abbildung 35: AxesLabel

Da uns eventuell die y-Achse detaillierter interessiert ändern wir das Verhältnis beider Achsen (siehe Abb. 36).


```
AspectRatio → 1.5
```

Abbildung 36: AspectRatio

Mit den zwei weiteren Optionen PlotStyle und Filling lassen wir den Plot etwas origineller aussehen. (siehe Abb. 37).

```
PlotStyle → {Red, Thick}, Filling → Bottom
```

Abbildung 37: PlotStyle und Filling

Alle Optionen kommen in die Plot Funktion (siehe Abb. 38).

```
Plot[x, {x, 0, 3}, AxesLabel → {"x", "y"}, AspectRatio → 1.5, PlotStyle → {Red, Thick}, Filling → Bottom]
```

Abbildung 38: Plot-Funktion mit Optionen

Das Ergebnis dieser Optionen sieht so aus (siehe Abb. 39).

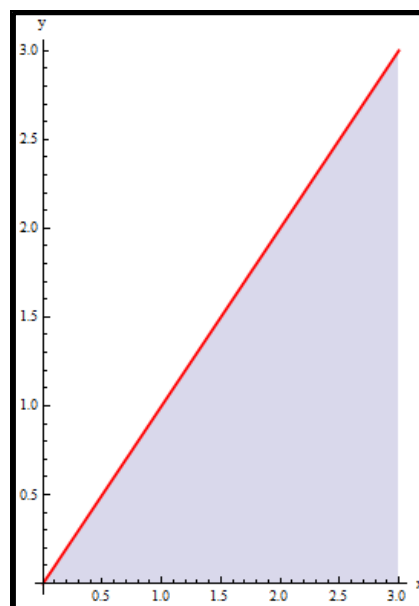


Abbildung 39: Plot mit Optionen

Es gibt noch eine ganze Reihe weiterer Optionen, die alle auf die gleiche Weise implementiert werden.

Aufgabe 5 – TMIII – Impuls-, Energie-, Arbeitssatz

In dieser Aufgabe behandeln wir die Funktion **Solve** und wie man das resultierende Ergebnis dieser Funktion aus der Klammer holt.

Die Funktion Solve dient der Lösung von Gleichungen und Gleichungssystemen (linearer und nicht-linearer).

In der folgenden Abbildung werden die Nullstellen eines Polynoms bestimmt (siehe Abb. 40):

```
f[x_] := x^2 + x - 2
Solve[f[x] == 0, x]
{{x -> -2}, {x -> 1}}
```

Abbildung 40: Solve

Zu beachten ist:

Gleichungen mit `==` formulieren.

Ebenso können lineare Gleichungssysteme als Liste eingegeben werden.

```
lgs = {2 x + 3 y + z == 1, x - y - z == 4, 3 x + 7 z == 5};
Isol = Solve[lgs]
{{x -> 101/41, y -> -49/41, z -> -14/41}}
```

Abbildung 41: Solve eines LGS

Bei diesem Beispiel wurde nach den drei Variablen (x, y, z) aufgelöst (siehe Abb. 41).

Da Ergebnisse von Solve immer in geschweiften Klammern erscheinen, muss man diese aus der Klammer holen, um mit ihnen weiter rechnen zu können.

In der Programmierung arbeitet man häufig mit Listen. Zum Beispiel ist eine Matrix eine Liste von Listen (siehe Abb. 42).

```

(a b)
(c d)

{{a, b}, {c, d}}

FullForm[%]

List[List[a, b], List[c, d]]

```

Abbildung 42: Liste von Listen

Genauso wie das Ergebnis des ersten Beispiels zur Solve Funktion (siehe Abb.43).

```

f[x_] := x^2 + x - 2

Solve[f[x] = 0, x]

{{x -> -2}, {x -> 1}}

```

Abbildung 43: Solve - Liste

Will man nun mit dem Ergebnis der Nullstellenberechnung $x=1$ weiter rechnen, muss man das Ergebnis aus der Klammer holen (siehe Abb. 44).

```

FullForm[{{x -> -2}, {x -> 1}}]

List[List[Rule[x, -2]], List[Rule[x, 1]]]

```

Abbildung 44: Ergebnis raus holen

Zur Darstellung nennen wir die Lösung "a01" und holen in einem weiteren Schritt die erste Nullstelle daraus (siehe Abb. 45).

```

a01 = {{x → -2}, {x → 1}};
ErsteNullstelle = a01[[2, 1, 2]]
1
  
```

Abbildung 45: Ergebnis rausgeholt

Dies geschieht mit den eckigen Klammern [[Aus der zweiten Liste, Den ersten Eintrag, Und davon der zweite Teil]]

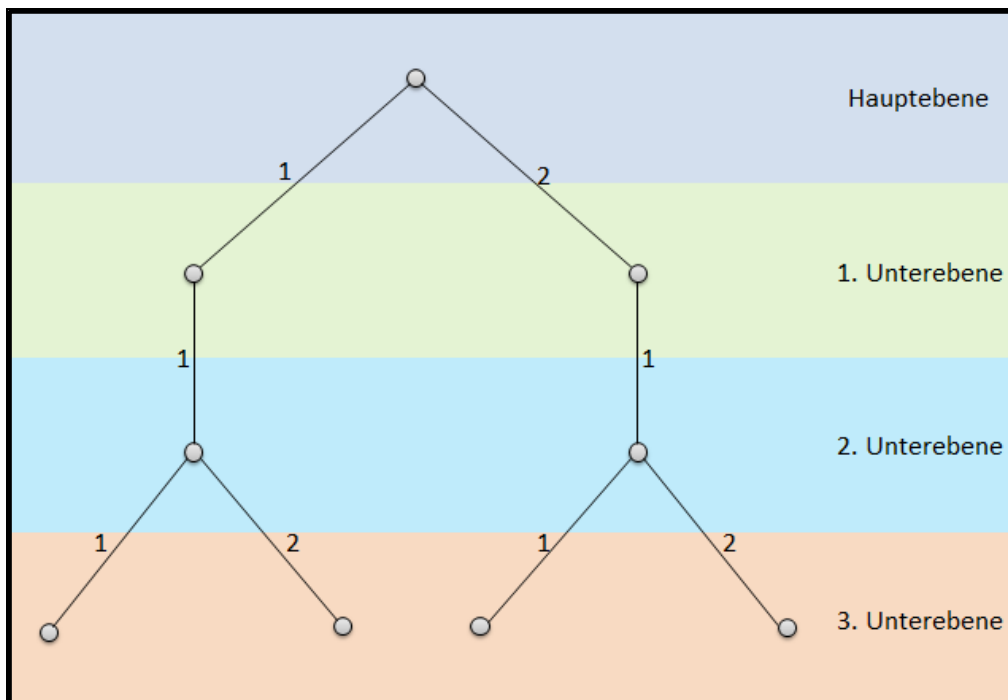


Abbildung 46: Listenbaum

Dieser Baum kann beliebig lang fortgeführt werden (siehe Abb. 46). In den eckigen Klammern [[]] sind von links nach rechts die Ebenen eingetragen. Jede Ebene trennt sich durch eine Komma: [[1.Unterebene, 2.Unterebene, 3.Unterebene]].

```

Liste = {{1, 2}, {1, 2}}
{{1, 2}, {1, 2}}
Liste[[1, 2]]
2
  
```

Abbildung 47: Listenbaum

Die Hauptebene ist die mit der äußeren geschweiften Klammer. In diesen Klammern befinden sich weitere “4“ Klammern. Das ist die erste Unterebene. Die Einträge der ersten Unterebene sind die zweite Unterebene. Will man nun an die “2“ der ersten Klammer gelangen nimmt man den ersten Eintrag der ersten Unterebene und von dort den zweiten Eintrag der zweiten Unterebene [[1,2]] (siehe Abb. 47).

In der nächsten Abbildung folgt ein übertriebenes Beispiel der Verschachtelung. Möchte man dort an einen gezielten Eintrag gelangen sind viel mehr Ebenen notwendig (siehe Abb. 48).

```
FullForm[{{x → -2, a}, {x → 1, {a, {t, {c, {y, x}}}}}]
List[List[Rule[x, -2], a], List[Rule[x, 1], List[a, List[t, List[c, List[y, x]]]]]
```

Abbildung 48: Beispiel einer komplexen Verschachtelung

Aufgabe 6 – TMIII – Kinematische Bindung

Lineares Gleichungssystem lösen mit dem Befehl “Solve”

In der Aufgabe “Kinematische Bindung” entstehen zu Beginn mehrere Gleichungen. Die Umstellung nach einer Variablen ist gesucht. Mathematica bietet hier verschiedene Möglichkeiten das Gleichungssystem zu Lösen. Wir haben uns den Befehl “Solve” mit einer Liste zu eigen gemacht.

```
u01 = Solve[m1 * a1 == m1 * g - S1 && -S2 * r + S3 * r == 0 && m2 * g - Nk + S2 == 0 && theta * aphi2 == -R * r + S2 * r && R == mu * Nk, {a1, S3, S2, R, Nk}]
{{a1 -> - (2 g - 5 g mu) / (-5 + 2 mu), S3 -> - (6 m (g + g mu) / (-5 + 2 mu)), S2 -> - (6 m (g + g mu) / (-5 + 2 mu)), R -> - (21 g m mu / (-5 + 2 mu)), Nk -> 3 g m - (6 m (g + g mu) / (-5 + 2 mu))}}
```

```
a1 = u01[[1, 1, 2]]
```

$$-\frac{2g - 5g\mu}{-5 + 2\mu}$$

Abbildung 49: Gleichungssystem nach 5 Unbekannten lösen

In unserem Beispiel haben wir fünf Gleichungen in einer List-Form beschrieben (siehe Abb. 49). Nach jeder Gleichung haben wir “&&” eingefügt, damit erkennt Mathematica, dass eine neue Gleichung folgt. Das Doppelte “=”, bedeutet Test auf Gleichheit. Nach dem alle fünf Gleichungen formuliert sind, erstellt man eine Liste mit den unbekannt Variablen. Es geht aber auch ohne &&, s. Abb. 51.

Eliminate

Eine weitere Variante zu dem “Solve” Befehl ist das Kommando “**Eliminate**” (siehe Abb. 50). Mit diesem Befehl löst Mathematica nach einer Unbekannten “c” auf und setzt direkt den berechneten Wert in die Gleichung ein. Es ist ein vorbereitender Schritt für “Solve”, um ein System vor dem Lösen auf den für die jeweilige Anwendung wesentlichen Teil zu reduzieren. Dadurch wird “Solve” die Arbeit erleichtert.

```

glsys = {4 x + 11 y + 4 z == a, 4 x - 5 y - 1 z == b, 2 x + 5 y + 6 z == c}
{4 x + 11 y + 4 z == a, 4 x - 5 y - z == b, 2 x + 5 y + 6 z == c}

{2 x + 3 y + z == a, x - y - z == b, 3 x + y + 7 z == c};

Eliminate[glsys, z]

4 b == -a + 20 x - 9 y && 8 x == 3 a - 2 c - 23 y
  
```

Abbildung 50: Gleichungen minimieren und auflösen

Gleichungssystem in Matrixform

Eine weitere Möglichkeit ist, das ganze Gleichungssystem in eine Matrixform zu schreiben (siehe Abb. 51).

```

lgs = {2 x + 4 y + z == 9, x - y - 6 z == 8, 4 x + 9 z == 2};
Isol = Solve[lgs]
{{{x -> 415/146, y -> 159/146, z -> -76/73}}}

mat = {{2, 4, 1},
       {1, -1, -6},
       {4, 0, 9}}

{{2, 4, 1}, {1, -1, -6}, {4, 0, 9}}

a01 = Solve[mat.{{x},
                {y},
                {z}} == {{9},
                        {8},
                        {2}}]

{{{x -> 415/146, y -> 159/146, z -> -76/73}}}
  
```

Abbildung 51: Anwendung der Matrixschreibform

Bei dieser Variante zur Lösung eines Gleichungssystems haben wir zunächst eine Liste mit den Gleichungen erstellt. Das System hat eine eindeutig bestimmbare Lösung. Darauf folgend haben wir die Matrixschreibform angewendet. Sobald eine Unbekannte nicht auftritt, in diesem Fall das “y” in der dritten Gleichung, schreiben wir in der Matrix eine “0”.

Hinweis: die Unbekannten sowie die rechte Seite sind in Vektor Darstellung aufgeführt.

Der Aufwand für unsere Aufgabe wäre sehr viel höher, da wir eine 5x5 Matrix erstellen müssten. Zudem dürfen keine unbekannt Größen in der Matrix auftauchen und die rechte Gleichungsseite muss definiert sein. Beim Erstellen der Matrix treten daher häufig Fehler auf. Allerdings gibt es auch hierfür einen Befehl: `Normal[CoefficientArrays[...]]`.

Der schnellste Weg für uns war der `Solve`-Befehl, welchen wir in unserem Beispiel auch verwendet haben.

Matlab

Aufgabe 1 – TMII – Belasteter, gelagerter Balken

In dieser Aufgabe behandeln wir die Problematik des **Föpplns** (siehe Abb. 52) und geben Informationen für das Plotten von Funktionen. Aber zuerst die Föppl-Problematik. Das Werkzeug zur Beschreibung von Mehrfeldbalken wird in Matlab mit dem Befehl “**heaviside**” realisiert (siehe Abb. 53). Dieser generiert einen Einheitssprung bei $x=0$, soweit keine Spezifikation eingegeben wurde.

$$\langle x - a \rangle^n = \begin{cases} 0 & \text{für } x < a \\ (x - a)^n & \text{für } x > a \end{cases}$$

Abbildung 52: Föppl-Symbolik aus TMII

$$(x-a)^n * heaviside(x-a)$$

Abbildung 53: Föppl-Realisierung in Matlab

Zur Verdeutlichung: Die spitzen Klammern werden durch $heaviside(x-a)$ ersetzt.

Mit dieser Schreibweise lassen sich Balken mit der Föppl-Symbolik in Matlab beschreiben.

Alternativ ist der “**piecewise**”-Befehl zu erwähnen. Mit diesem lassen sich Funktionen bereichsweise beschreiben. Allerdings lässt er sich nicht integrieren und ist somit nicht geeignet.

Als nächstes folgt das Thema “**Plot**”. Funktionen werden mit dem Befehl “Plot” visualisiert (siehe Abb. 54).

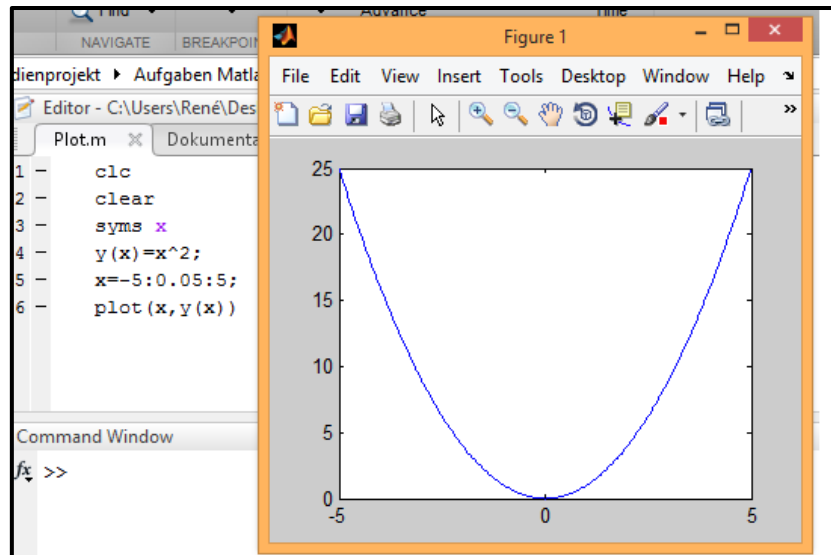


Abbildung 54: Plot einer Funktion

Um einen zweiten Graphen in den Plot zu zeichnen kann man dies auf zwei Arten machen. Bei der ersten Art wird der Graph in den gleichen Plot gezeichnet (siehe Abb. 55).

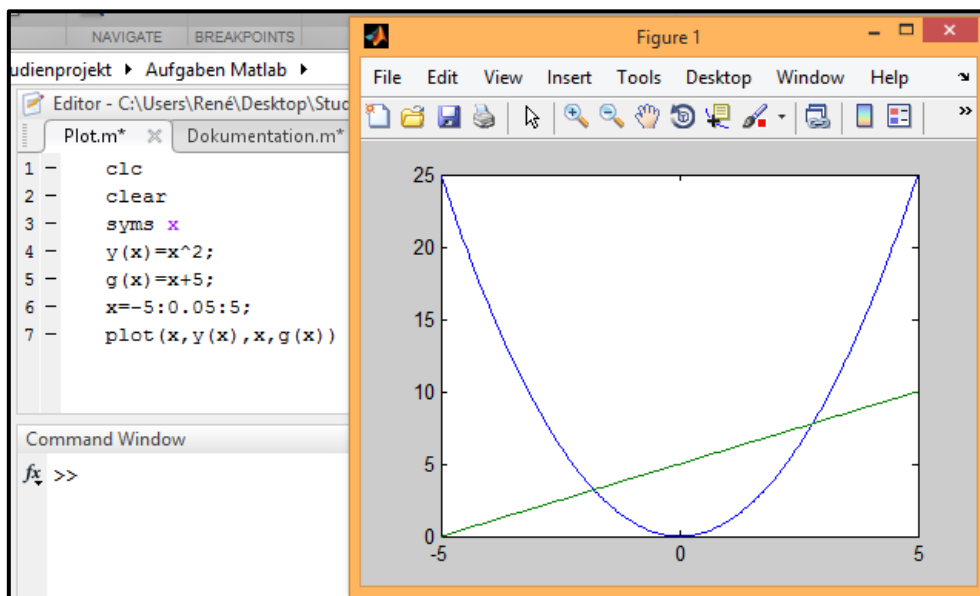


Abbildung 55: Zwei Plots in einem

Die zweite Möglichkeit ist **Subplots** zu erstellen, also die Graphen in unterschiedlichen Koordinatensystemen zu erstellen (siehe Abb. 56).

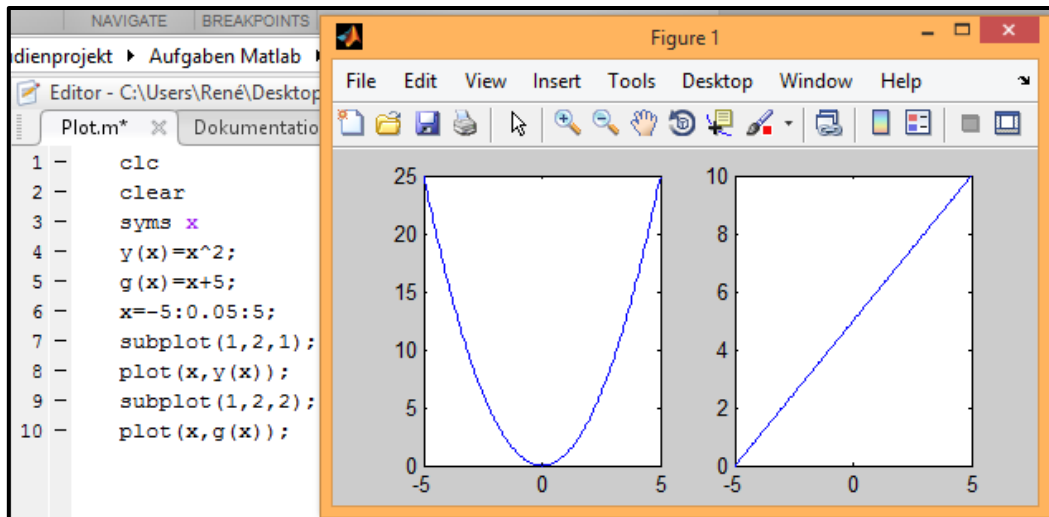


Abbildung 56: Subplot

Subplots funktionieren folgendermaßen: subplot(Zeilen, Reihen, Position). Also in dem Beispiel aus der Abbildung: eine Zeile, zwei Spalten. $y(x)$ auf Position 1 und $g(x)$ auf Position 2. Zeilen und Spalten können beliebig erweitert werden.

Nun geht es darum, ein Gitternetz, Achsenbeschriftungen, einen Titel und ggf. eine Legende einzufügen (siehe Abb. 57).

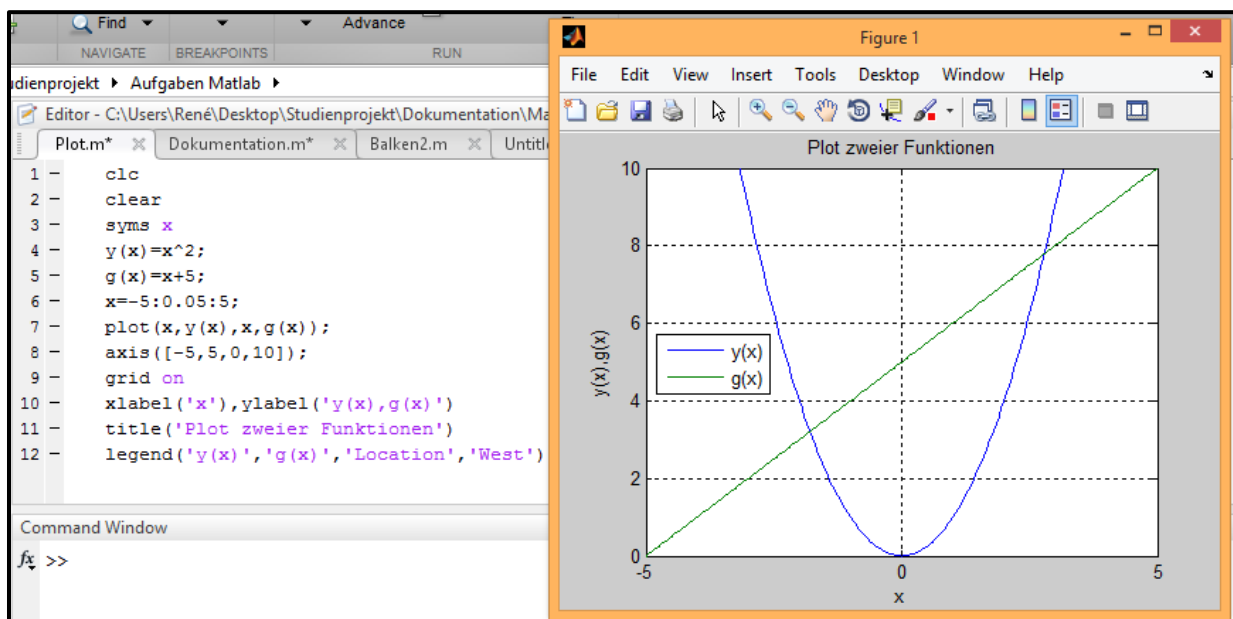


Abbildung 57: Plot mit Zusätzen

Eine Tabelle verdeutlicht die Abbildung:

Zusatzoption	Matlab-Befehl	Beschreibung
Gitternetz	grid on	erzeugt ein Gitternetz
Achsenbeschriftung	xlabel, ylabel	erzeugt Achsenbeschriftungen
Titel	title	erzeugt einen Titel
Legende	legend	erzeugt eine Legende und durch 'Location', 'West' erscheint sie links am Rand innerhalb der Zeichnungsebene
Anzeigebereich	axis	Bestimmt den Bereich der angezeigt werden soll

Tabelle 2: Plotoptionen

Aufgabe 2 – Fourier

Bei der Fourier Aufgabe haben wir zwei **Sections** erstellt. Das Ganze funktioniert indem man in dem Editor ein “%%” eingibt. Hierfür gibt es die Alternative über den Reiter Editor → Insert → New Sektion (siehe Abb. 58).

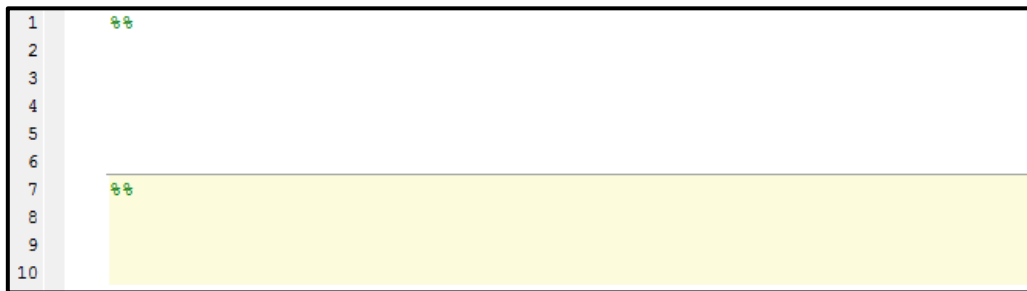


Abbildung 58: Mehrere Sections

Der Grund, weshalb wir zwei Sections eingefügt haben ist, dass mehrere Sections unabhängig voneinander evaluierbar sind. Sobald man eine Grundrechnung, wie zum Beispiel die Fourier-Entwicklung, eingibt, hat es den Vorteil, dass nur ein bestimmter Teil der Grundrechnung verändert wird. Dieser bestimmte Teil kann die Ordnung sein, welche man erhöhen möchte. In der ersten Section haben wir die Rechteckfunktion sowie den Fourier Term festgelegt. In der darauffolgenden Section haben wir die Plots programmiert. Sobald man “Run Section” durchführt, erscheinen die Plots, durch erneutes evaluieren haben wir die Ordnung der Fourier-Reihe erhöht. Hierfür haben wir zuerst $sum=0$ und $n=0$ festgelegt. In der nächsten Section haben wir den Ausdruck: $n=n+1$, $sum=sum+f(x,n)$ verwendet (Siehe Abb. 59).

```

4      % Rechteckfunktion
5 -    y=2*pi*(mod(x,2*pi)<pi)-pi;
6      % Fourier-Reihe
7 -    f=@(x,n) (2-2*cos(n*pi))/n*sin(n*x);
8 -    sum=0;
9 -    n=0;
10     %%
11 -    n=n+1;
12     % Höher werdende Fourier-Reihe
13 -    sum=sum+f(x,n);
  
```

Abbildung 59: Fourier-Reihe

Um eine periodisch fortgesetzte Funktion zu erstellen benutzen wir den Befehl “**mod**” (**Modulo**). Dieser Befehl verwendet den Rest einer Division.

Beispiel: Die Periode hat eine Dauer von 5. Der Funktionswert an der Stelle 23 ergibt sich dann zu: $23/5=4$ Rest 3

→ Der Funktionswert an der Stelle 23 ist identisch zu dem Funktionswert an der Stelle 3.

Somit kann Matlab den Funktionswert an beliebiger Stelle selbst ermitteln.

Die nächste Abbildung zeigt eine periodisch fortgesetzte Rechteckfunktion (siehe Abb. 60).

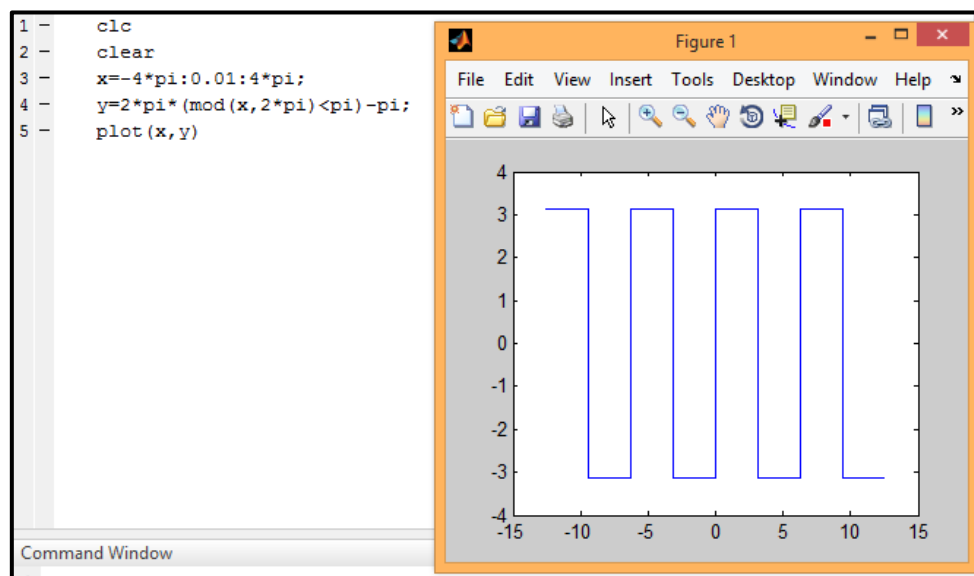


Abbildung 60: Rechteckfunktion mit Modulo Befehl

Für die Erstellung von symbolischen Funktionen benötigen wir den Befehl “**syms**” (siehe Abb. 61).

```

1      clc
2      clear
3      syms x
4      Gp(x)=x^2;
5      Gp(2)
  
```

Abbildung 61: Symbolische Funktion

Für die Erstellung von nicht symbolischen Funktionen wird der Befehl ”syms” nicht mehr benötigt. Nur die Funktionsdefinition ändert sich. Aus $Gp(x)$ wird $Gp=@(x)$ (Siehe Abb. 62).

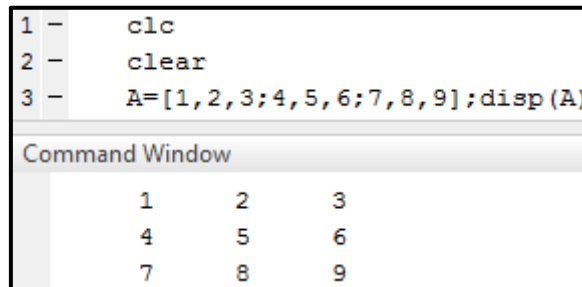
```
1  clc
2  clear
3  Gp=@(x) x^2;
4  Gp(2)
```

Abbildung 62: Nicht Symbolische Funktion

Aufgabe 3 – Technische Schwingungen

In der Aufgabe technische Schwingungen benötigen wir die **Matrizen**.

Um generell eine Matrix zu erstellen werden statt der runden Klammern, eckigen Klammern verwendet. Ein Komma trennt die Einträge einer Zeile (Die Anzahl der Einträge legen die Spaltenanzahl fest). Das Semikolon steht für den Eintrag einer neuen Zeile (siehe Abb. 63).



```

1 -   clc
2 -   clear
3 -   A=[1,2,3;4,5,6;7,8,9];disp(A)

```

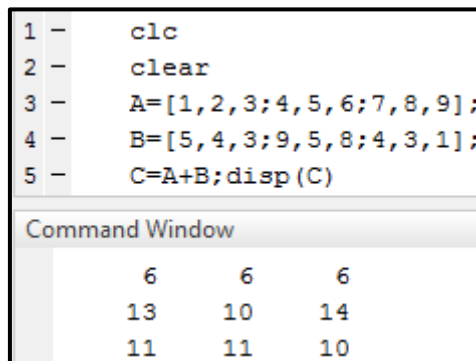
Command Window

1	2	3
4	5	6
7	8	9

Abbildung 63: Matrixschreibweise

1 Matrixschreibweise

Das Rechnen mit Matrizen ist in der Abbildung 64 dargestellt. Zum Beispiel das Addieren von zwei Matrizen. Hierbei genügt es zwischen beiden Matrizen einer der vier mathematischen Operationen Addition, Subtraktion, Multiplikation und Division beizufügen.



```

1 -   clc
2 -   clear
3 -   A=[1,2,3;4,5,6;7,8,9];
4 -   B=[5,4,3;9,5,8;4,3,1];
5 -   C=A+B;disp(C)

```

Command Window

6	6	6
13	10	14
11	11	10

Abbildung 64: Rechnen mit Matrizen

Für die Berechnung der **Determinante** in Matlab verwenden wir die Befehlseingabe “**det**”. (Siehe Abb. 65).

```

1 -   clc
2 -   clear
3 -   A=[1,2;4,5];
4 -   d=det(A);disp(d)

```

Command Window

```
-3
```

Abbildung 65: Determinate

Desweiteren haben wir das **Eigenverhalten** bestimmt. Der Befehl dafür heißt in Matlab “**eig**”. (siehe Abb. 66).

```

1 -   clc
2 -   clear
3 -   A=[1,2,3;4,5,6;7,8,9];
4 -   d=eig(A);disp(d)

```

Command Window

```

16.1168
-1.1168
-0.0000

```

Abbildung 66: Eigenverhalten
4 Eigenverhalten

Aufgabe 4 – TMII – Belastete, elastische Stäbe

Mit Hilfe dieser Aufgabe werden die Themen, die zum symbolischen Rechnen mit Matlab notwendig sind, behandelt. Außerdem geben wir Ihnen Hinweise, wie man ein m-File strukturiert aufbaut.

Nach dem Start von Matlab befindet man sich im Command Window. Dort lassen sich alle Befehle ausführen, allerdings nicht abspeichern. Um die Arbeit abzuspeichern bedienen wir uns dem Editor. In den Editor geben wir die Eingabe ein. Über den Button “Run Section” (siehe Abb. 67) erscheint die Ausgabe im Command Window.

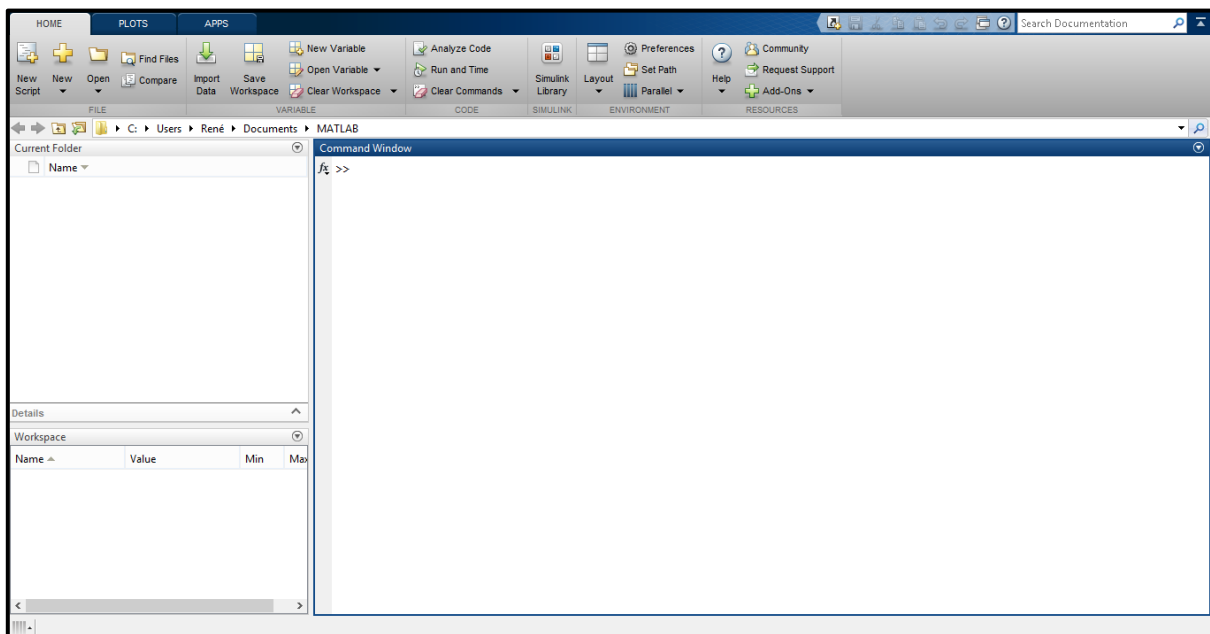


Abbildung 67: Matlab

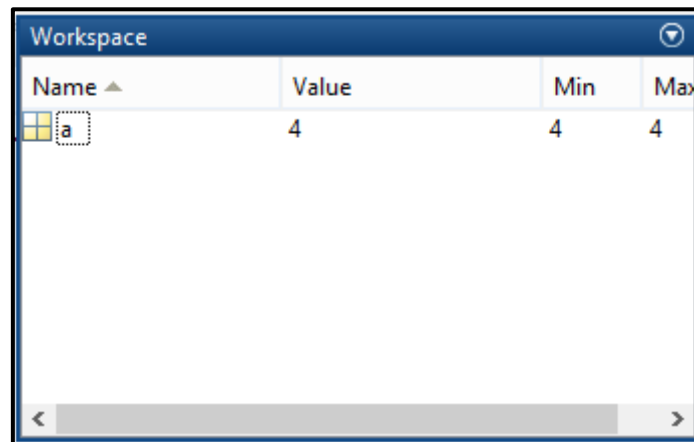
Zu Beginn jedes m-Files (Abk.) kommen die Befehle “**clc**” und “**clear**”. Mit dem Befehl “**clc**” werden alle Einträge im Command Window gelöscht (siehe Abb. 68).

```

1 -   clc
2 -   clear
  
```

Abbildung 68: clc, clear

Der “clear”-Befehl löscht beim Ausführen den Workspace. Im Workspace befinden sich alle deklarierten Variablen. Des Weiteren erscheint mittels Doppelklick auf eine Variable (im Workspace) eine Tabelle. Dort lassen sich die Werte dieser Variablen ändern (siehe Abb. 69).



Name ▲	Value	Min	Max
a	4	4	4

Abbildung 69: Workspace

Matlab basiert auf der numerischen Berechnung, das symbolische Rechnen wurde erst später zum Programm hinzugefügt. Mit der Anweisung “syms” werden Variablen als Symbole verstanden. Die Variable ist solange ein Symbol, bis ihr ein Zahlenwert zugeordnet wird.

Hinweise:

Um gefundene Werte für Variablen einzusetzen wird der Ausdruck “subs” benutzt. In der folgenden Abbildung (siehe Abb. 70) wird die Gleichung $a=b+c$ beschrieben. Nachdem b die Zahl 3 zugewiesen wurde, ändert a die Ausgabe nicht automatisch auf $a = b+3$, sondern bleibt bestehen. Erst durch die Eingabe “subs” werden alle Variablen sofern möglich substituiert.

```

>> syms a b
>> a = 2*b
a =
2*b
>> b = 1
b =
    1
>> a
a =
2*b
>> a = subs(a)
a =
    2
  
```

Abbildung 70: subs

Wir möchten noch auf einen weiteren Befehl hinweisen. Um ein Ergebnis oder Text in der Ausgabe zu erstellen, wird der Befehl “**disp**” genutzt (siehe Abb. 71).

```

>> x = pi^2;
>> disp(x)
    9.8696
>> disp('x ist Pi zum Quadrat')
x ist Pi zum Quadrat
  
```

Abbildung 71: disp

Aufgabe 5 – TMIII – Impuls-, Energie-, Arbeitssatz

Bei nahezu allen Aufgaben der Technischen Mechanik, sowie der Technischen Schwingungslehre, wird das Integrieren und Differenzieren benötigt.

Der Befehl Integrieren wird in Matlab durch den Ausdruck “**int**” beschrieben.

Zum einen das Bestimmte Integral. Hier ist zu beachten, dass die Funktion nach dem Befehl “int”, in runden Klammern angegeben wird. Ebenso sind die Grenzen anzugeben. In der Abbildung 72 sind die Grenzen von 0 bis 1 gewählt.

```

1      clc
2      clear
3      syms x
4      a=int(sin(x)+cos(x),x,0,1);disp(a)

```

Command Window

```

sin(1) - cos(1) + 1

```

Abbildung 72: Bestimmtes Integral

Zum anderen das unbestimmte Integral (siehe Abb. 73).

```

1      clc
2      clear
3      syms x
4      a=int(sin(x)^2,x);disp(a)

```

Command Window

```

x/2 - sin(2*x)/4

```

Abbildung 73: Unbestimmtes Integral

Beim Differenzieren wird der Befehl “**diff**” verwendet. Anschließend folgt die Funktion in Runden Klammern und die Ableitungsvariable x (siehe Abb. 74).

```

1   clc
2   clear
3   syms x
4   a=diff(x^2+3*x+4,x);disp(a)

```

Command Window

2*x + 3

Abbildung 74: Differentiation

Differenzieren in Matlab ist generell partiell. Durch den Befehl “diff” ist das Programm auch in der Lage, eine Gleichung mit zwei Variablen zu differenzieren (siehe Abb. 75).

```

1   clc
2   clear
3   syms x y
4   a=diff(x^2+y^2,x);disp(a)

```

Command Window

2*x

Abbildung 75: Partielle Differentiation

Der Befehl “**assume**” wird in Matlab zur Annahme verwendet (z.b. x soll nur positive Werte annehmen: $x > 0$). Da wir in der technischen Mechanik meist nur das positive Ergebnis eines Polynoms benötigen, wird “assume” verwendet.

Der Befehl “assume” gilt für die folgenden Einträge im Editor, daher ist es ratsam den Befehl zu Beginn der Rechnung anzuwenden (siehe Abb. 76).

```

1 -   clc
2 -   clear
3 -   syms x
4 -   assume(x>0)
5 -   f(x)= x^2+4*x-21;
6 -   Lsg=solve(f(x),x);disp(Lsg)

```

Command Window

3

Abbildung 76: “assume” anhand einer Aufgabe

Aufgabe 6 – TMIII – Kinematische Bindung

In der technischen Mechanik werden sehr oft Lösungen von Gleichungssystemen gesucht. Deshalb gehen wir im Folgenden auf die Befehle “**solve**”, “**simplify**” und “**pretty**” ein.

Mit “**solve**” werden Gleichungen nach einer gewünschten Variablen umgestellt. Somit können Lösungen für Gleichungen und Gleichungssystemen gefunden werden (siehe Abb. 77).

```

1 -   clc
2 -   clear
3 -   syms x
4 -   Lsg=solve(x^2+4*x-5,x);disp(Lsg)

```

Command Window

```

1
-5

```

Abbildung 77: Mit solve Nullstellen gefunden

In Abbildung 78 ist die Lösung eines linearen Gleichungssystems zu sehen. Nicht-lineare werden analog zum LGS berechnet.

```

1 -   clc
2 -   clear
3 -   syms x y z
4 -   [x,y,z]=solve(x+4*y-5*z==0,y-3*z==2,2*x+5*y==1);disp(x);disp(y);disp(z)

```

Command Window

```

-57
23
7

```

Abbildung 78: Lösung eines LGS

Der Befehl “**simplify**” wird bei unübersichtlichen Gleichungen verwendet. Er versucht die Gleichung zu vereinfachen (siehe Abb. 79).


```

1 -   clc
2 -   clear
3 -   syms a b
4 -   G1=simplify(a^2+2*a*b+b^2);disp(G1)

```

Command Window

```
(a + b)^2
```

Abbildung 79: Vereinfachung

Als letztes möchten wir ihnen die Funktion “pretty” vorstellen. Diese erstellt aus der gewohnten Schreibweise eine “hübschere” Darstellung einer Gleichung (siehe Abb. 80).

```

1 -   clc
2 -   clear
3 -   syms a b
4 -   G1=(a-b^4)/(b*a^2);
5 -   pretty(G1)

```

Command Window

```

      4
    - b + a
    -----
      2
     a b

```

Abbildung 80: Hübsche Darstellung

Auswertung

Ein fairer Vergleich zeichnet sich durch identische Voraussetzungen aus. Daher wurden die gleichen Aufgaben jeweils von beiden Programmen gelöst.

Durch die Bearbeitung der Aufgaben haben wir ausreichende Erfahrungen mit beiden Programmen gesammelt. In der folgenden Tabelle stellen wir detailliert jede Funktion und Tatsache gegenüber.

Dabei wird jeder Punkt bewertet und am Ende aufsummiert.

Vergleich

		Mathematica		Matlab	
Bezeichnung		-	Bewertung	-	Bewertung
Allgemeines	Kosten	300€ pro Jahr	0	2000€	0
	Lizenzdauer	1 Jahr	/	unbegrenzt	/
	Kompatibilität	Windows, Mac	++	Windows, Mac	++
	Aktualität	Mathematica 10	/	Matlab R2014b	/
	Einarbeitungs- medien	Hands-on to Start Mathematica	++	Step-by-Step Guides to MATLAB and Simulink	0
	Hilfecenter	Dokumentation Center	+	Dokumentation Center	0
Bezeichnung		Syntax	Bewertung	Syntax	Bewertung
Funktionen	Gleichung lösen	Solve[...]	++	solve(...)	+
	Vereinfachung	Simplify[...]	+	simplify(...)	+
	Sections	Alt+F4	++	%%	++
	Graph zeichnen	Plot[...]	+	plot(...)	0
	Integrieren	Integrate[...]	++	int(...)	++
	Differenzieren	D[...]	++	diff(...)	++
	Föppl-Symbol	UnitStep[...], HeavisideTheta[...]	-	heaviside(...)	-
	Matrix	{{...},{...},{...}} }	++	[...;...;...]	++
	Determinante	Det[...]	++	det(...)	++
	Eigenvektoren	Eigenvectors[...]]	++	eig(...)	++
Bezeichnung		Anmerkung	Bewertung	Anmerkung	Bewertung
Sonstiges	Oberfläche	Ein Fenster, Paletten ergeben neue Fenster	/	Mehrere Fenster in einem, Plots ergeben neue Fenster	/
	Indizes	Teilweise Möglich	-	Nicht möglich	--

Effizienz	Paletten	0	Reiter, Workpiece, Command Historie, Pfade	++
Erweiterungen	Packages	0	Apps	+
Plattformübergreifend	Mathematica online	++	Handy-App	++
Formatierung	Palette	++	/	--
Befehl Eingabe	Eingabehilfe	+	/	-
Fehlersuche	Fehler werden in der Eingabe nicht angezeigt	-	Erkennung der Fehler bei der Eingabe	+
Symbolisches Rechnen	/	++	/	0
Numerisches Rechnen	/	+	/	++
Interaktivität	Manipulate [...]	++	input(...)	+
Text darstellen	Format → Style → Text	++	disp‘...‘	0
Syntax verschönern	automatisch oder über Paletten	+	pretty(...)	+
Zeichnen (FKB)	Drawing Tools	+	/	--
Syntax	Funktionsname groß, Eingabe in [...]	/	Funktionsname klein, Eingabe in (...)	/
Annahmen	Assumption	++	assume	+
Funktionen deklarieren	Name[Variable_]:=...	++	syms Variable; Name=@(Variable)	+
Mit bestimmtem Ergebnis weiter rechnen	[...,...,...]	++		0

Tabelle 3: Vergleichstabelle

Nun summieren wir die Bewertungen auf. Dabei gilt:

Punkte	Bewertungssymbol
- 2 Punkte	--
-1 Punkt	-
0 Punkte	0
1 Punkt	+
2 Punkte	++

Tabelle 4: Bewertung

Daraus folgt folgende Punktevergabe:

Mathematica	38 Punkte
Matlab	21 Punkte
Differenz:	17 Punkte

Tabelle 5: Ergebnis

Da wir uns mit Mathematica wesentlich länger beschäftigten (Grund: siehe Einarbeitung) und dadurch mehr kennen lernten, gleichen wir das mit +5 Punkten für Matlab und -5 Punkten für Mathematica aus. Das führt zum Endergebnis:

Mathematica	33 Punkte
Matlab	26 Punkte
Differenz:	7 Punkte

Tabelle 6: Endergebnis

Bewertung

Das Endergebnis kann nicht als absolutes Ergebnis angesehen werden, da wir die Programme aus unserem Standpunkt bewerteten. Aus diesem Grund führen wir nun eine Fallunterscheidung durch.

Beide Systeme lassen sich entsprechend programmieren, dass dadurch eine Benutzeroberfläche entsteht. Zum Beispiel kann damit der Laie ein und dieselbe Rechnung mit unterschiedlichen Werten immer wieder berechnen.

So eine Situation kann von uns in diesem Projekt nicht überprüft werden, daher wird die folgende Beurteilung eher subjektiv als objektiv behandelt.

In Mathematica kann ohne Vorkenntnisse mittels „Manipulate“ programmiert werden, somit kommt jeder Anwender damit zurecht. Der Aufwand zum Programmieren vom Manipulieren von Daten beginnt sehr gering, wächst aber schnell auf ein hohes Niveau.

Wenn man dagegen Matlab in Betracht zieht, ist es dort möglich mit Hilfe von „input“, Werte beim Anwender abzufragen. Dies ist simpel gehalten und auch der Programmieraufwand einer solchen Abfrage bleibt gering.

Aus unserer Sicht ist der Programmieraufwand solcher „Programme“ bei beiden CAS gleich. Bei Mathematica erscheint jede Zeile in „Manipulate“. Matlab hingegen besitzt eine unübersichtlichere Oberfläche. Bei Anwendung der „Programme“ sehen wir Matlab jedoch im Vorteil, da die Abfrage eines Wertes effizient und transparent gehalten ist.

Wir halten fest:

Bei beiden Programme kommt es drauf an, wer sie benutzen möchte und zu welchem Zweck. In großen Firmen sind CAS bereits etabliert, zudem wurden „Anwendungsprogramme“ dort schon programmiert.

Fazit – Schlusswort

Zum Schluss eine rein subjektive Beurteilung der CAS, sowie Anregungen und Wünsche für die Zukunft.

Beide Programme bieten bei der Lösung von Problemstellungen Hilfe zur Findung eines geeigneten Befehls an. Durch das Arbeiten mit den CAS fiel uns auf, dass der Rechenaufwand stark minimiert wird. Ebenso konnten wir in Erfahrung bringen, dass der Verständnisaufbau zu Lösung der Aufgaben gefördert wurde.

Bei beiden Programmen war eine gewisse Einarbeitungszeit notwendig. Aber sobald man einmal verstanden hat wie die Syntax funktioniert, lief die weitere Bearbeitung einfacher ab. Mathematica konnte in dem Punkt Veranschaulichung der Aufgaben deutlich besser abschneiden als Matlab, da sich Matlab rein auf das Programmieren beschränkt.

Da wir unsere Arbeit mit dem Programm Mathematica begonnen haben, neigen wir zu der Meinung, dass es für unser Projekt das bessere Programm ist.

Gestützt wird diese Meinung durch anbieten einer breiten Bearbeitungsmöglichkeit. Zudem konnten wir in Mathematica einfacher dokumentieren, visualisieren und gegebenenfalls noch manipulieren. Das „Alles in einem Paket“ erspart eine Menge Arbeit, da wir keine zusätzlichen Programme benutzen mussten um die Formatierung so hin zu bekommen (siehe Anlage A).

Natürlich ist Matlab auch nicht zu unterschätzen, da es ein breites Spektrum an Programmiermöglichkeiten bietet. Wir haben uns lediglich mit dem Programmieren im Editor und Command Window bedient. Mit den zahlreichen Apps haben wir uns nicht beschäftigt, dennoch sehen wir dies als Vorteil an.

Die Darstellungen von Texten sowie das Einfügen von Bildern ist ein Nachteil von Matlab.

Für die Zukunft wünschen wir uns, dass beide Programme an unserer Hochschule etabliert werden. Am besten in Form eines eigenständigen Moduls.

Die Kenntnisse, welche durch CAS vermittelt werden sind sehr umfangreich. Wir beide sprechen aus Erfahrung, dass die Technische Mechanik ein komplexer Bereich ist, mit

welchem einige Studenten an der Hochschule zu kämpfen haben. Das Erlernen und der Umgang mit den CAS Programmen kann hier Abhilfe schaffen.

Zudem wäre es angemessen, wenn die Hochschule sich für die Lizenz für Studenten einsetzt, da beide Programme kostenintensiv sind.

Schlusswort:

Mathematica überwiegt bei der Bearbeitung unseres Projektes.

Matlab ist in der deutschen Industrie weit verbreitet.

Beide Computer-Algebra Systeme zu beherrschen ist von Vorteil.

Literatur

Christian H. Weiß: Mathematica - Eine Einführung 2. Aufl. Deutschland, Universität Würzburg, Institut für Mathematik, Lehrstuhl für Statistik, 2008.

Michael Kofler, Hans-Gert Gräbe: Mathematica Einführung, Anwendung, Referenz , 4. Aufl.

Addison-Wesley Verlag, Deutschland, München, 2001. ISBN: 3-8273-1894-7

Anhang

Im Anhang befinden sich die Programmierungen der Aufgaben. Diese wurden als PDF aus den CAS exportiert.

Beginnend mit Mathematica, im Anschluss folgt Matlab.