

---

# Praktikum MATLAB®/Simulink® I

---

Prof. Dr.-Ing. U. Konigorski  
Skript



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

REGELUNGSTECHNIK *rtm*  
UND MECHATRONIK



---

# Praktikum MATLAB®/Simulink® I

Prof. Dr.-Ing. U. Konigorski

## Skript



Technische Universität Darmstadt  
Institut für Automatisierungstechnik und Mechatronik  
Fachgebiet Regelungstechnik und Mechatronik  
Prof. Dr.-Ing. U. Konigorski

Landgraf-Georg-Straße 4  
64283 Darmstadt  
Telefon 06151/16-25200  
[www.rtm.tu-darmstadt.de](http://www.rtm.tu-darmstadt.de)

Das Gesamtdokument ist unter CC BY-ND veröffentlicht:



<https://creativecommons.org/licenses/by-nd/4.0/>

Der Inhalt dieses Dokuments ausschließlich der Logos, des Layouts und der Schriftarten ist unter CC BY-SA veröffentlicht:



<https://creativecommons.org/licenses/by-sa/4.0/>



---

## Inhalt des Praktikums und Voraussetzungen

---

### Inhalt

Das Praktikum MATLAB/Simulink I wird an der Technischen Universität Darmstadt im Bachelorstudium angeboten und richtet sich im Wesentlichen an Studenten der Automatisierungstechnik und Mechatronik.

Das Praktikum MATLAB/Simulink I setzt keine Vorkenntnisse MATLAB und Simulink voraus.

Die Beispiele und Aufgaben, die im Rahmen dieses Praktikums bearbeitet werden, besitzen meist einen regelungstechnischen Hintergrund. Es wird davon ausgegangen, dass die Teilnehmer schon eine Grundlagenvorlesung zur Regelungstechnik gehört haben.

Inhaltlich werden in den ersten drei Versuchen die Themen

- Allgemeine Einführung in MATLAB,
- m-Files und Grafiken sowie
- Lösen von Differentialgleichungen

behandelt. In den Versuchen vier bis sechs werden diese – noch relativ allgemeinen – Grundkenntnisse erweitert und in Regelungsaufgaben angewandt:

- Analyse, Simulation und Reglerentwurf eines Ventilators mit Hilfe des regelungstechnischen Programmpaketes
- Simulation und Reglerentwurf für einen Pendelschrauber mit MATLAB und Simulink
- Reglerentwurf mit Hilfe der Wurzelortskurve unter Verwendung des SISO-Tools

### Unterlagen und Durchführung

Die Unterlagen zu diesem Praktikum bestehen aus

- dem vorliegenden Skript,
- den Hausaufgaben,
- den Versuchsunterlagen sowie
- zu manchen Versuchen aus vorbereiteten MATLAB-Funktionen.

Vor jedem Versuch sollte das entsprechende Kapitel im Skript durchgearbeitet werden. In diesem werden zum einen die in dem jeweiligen Versuch verwendeten regelungstechnischen Methoden erläutert. Dabei sind diese Erläuterung bei Themen eher knapp gehalten und sollen bewusst kein Skript zu diesem Thema ersetzen. Zum anderen werden die für den jeweiligen Versuch benötigten MATLAB-Funktionen und -Funktionalitäten erklärt.

Die Hausaufgaben sollen *vor* den Versuchsnachmittagen bearbeitet werden. Die in den Versuchsunterlagen gestellten Aufgaben sind während der einzelnen Versuchsnachmittage zu lösen.



---

# Inhaltsverzeichnis

<b>Versuch 1</b>	<b>1</b>
1.1. Allgemeine Einführung . . . . .	1
1.2. Systembefehle . . . . .	1
1.3. Grundrechenarten . . . . .	2
1.4. Matrizen . . . . .	4
1.4.1. Eingabe von Matrizen . . . . .	4
1.4.2. Mehrdimensionale Matrizen . . . . .	6
1.4.3. Matrixoperationen . . . . .	6
1.4.4. Vektorfunktionen . . . . .	9
1.5. Polynome . . . . .	10
1.5.1. Allgemeine Auswertung . . . . .	10
1.5.2. Polynomdivision und Multiplikation . . . . .	10
1.5.3. Partialbruchzerlegung . . . . .	11
<b>Versuch 2</b>	<b>13</b>
2.1. Einleitung . . . . .	13
2.2. m-Files und ihre Anwendung . . . . .	13
2.2.1. Skripte und Funktionen . . . . .	14
2.2.2. Schleifen . . . . .	15
2.2.3. Ein- und Ausgabe von Text . . . . .	17
2.3. Grafik in MATLAB . . . . .	19
2.3.1. Der plot-Befehl – 2D . . . . .	19
2.3.2. Der plot-Befehl – 3D . . . . .	21
2.3.3. Weitere Darstellungen . . . . .	22
2.4. Debugger . . . . .	22
<b>Versuch 3</b>	<b>25</b>
3.1. Einführung in Differentialgleichungen . . . . .	25
3.2. Lösen von Differentialgleichungen . . . . .	26
3.2.1. Analytische Lösung . . . . .	26
3.2.2. Approximation über Differenzengleichung . . . . .	26
3.2.3. Numerische Integration einer DGL . . . . .	27
<b>A. Einführung Versuch 4 – 6</b>	<b>33</b>
A.1. Der Pendelschrauber . . . . .	33
A.1.1. Das Teilsystem Ventilator . . . . .	33
A.1.2. Das Teilsystem Hebel . . . . .	35
A.1.3. Übertragungsfunktion des (linearisierten) Gesamtsystems . . . . .	36
A.2. Formelzeichen und Zahlenwerte . . . . .	38
A.3. Wichtige Befehle . . . . .	39
<b>Versuch 4</b>	<b>41</b>
4.1. Die Regelstrecke . . . . .	41
4.1.1. Blockschaltbild und Übertragungsfunktion der Regelstrecke . . . . .	41

4.1.2. Eigenschaften und Parameter von $PT_2$ -Gliedern . . . . .	41
4.2. Analyse des Systemverhaltens der Strecke . . . . .	45
4.3. Reglerentwurf . . . . .	48
4.3.1. Das Frequenzkennlinienverfahren . . . . .	48
4.3.2. Die Synthese nach dem Betragsoptimum . . . . .	51
<b>Versuch 5</b>	<b>55</b>
5.1. Kurzeinführung in Simulink . . . . .	55
5.2. Regelung des Ventilators unter Simulink . . . . .	58
5.2.1. Der Ventilator als Regelstrecke . . . . .	58
5.2.2. Reglerentwurf für den Ventilator . . . . .	59
5.3. Regelung des Pendelschraubers unter Simulink . . . . .	59
5.3.1. Der vollständige Pendelschrauber als Regelstrecke . . . . .	59
5.3.2. Reglerentwurf für den vollständigen Pendelschrauber . . . . .	61
<b>Versuch 6</b>	<b>63</b>
6.1. Die Wurzelortskurve . . . . .	63
6.1.1. Grundlagen . . . . .	63
6.1.2. Konstruktion der WOK . . . . .	65
6.1.3. Weiterführende Details . . . . .	66
6.2. Das SISO-Tool in MATLAB . . . . .	66
6.2.1. Grundlagen des SISO-Tools . . . . .	66
6.2.2. Besondere Funktionen des SISO-Tools . . . . .	70
6.3. Analyse von Parameteränderungen mit Hilfe der WOK . . . . .	70
6.3.1. Allgemeine Betrachtung von Parameterschwankungen . . . . .	70
6.3.2. Betrachtung des Arbeitspunkt winkels als Parameter . . . . .	72
<b>Literaturverzeichnis</b>	<b>75</b>



---

# Versuch 1

---

## 1.1 Allgemeine Einführung

---

MATLAB, die Abkürzung für **MAT**rix **LAB**oratory, wurde in den 70er Jahren entwickelt und ist heute ein universelles Werkzeug, das vor allem zur Programmierung technisch-wissenschaftlicher Probleme dient.

MATLAB führt eingegebene Befehle sofort aus und ist somit ein Interpreter und damit langsamer als ein kompiliertes, z. B. in C oder Fortran geschriebenes, Programm. Anders als beispielsweise in C müssen Variablen nicht deklariert werden.

Bei MATLAB stehen numerische Rechnungen und die Darstellung von Zahlenmaterial im Vordergrund. Der Grundbaustein ist eine Matrix, deren Dimensionen nicht explizit definiert werden müssen. Dadurch können numerische Probleme innerhalb kürzester Zeit gelöst werden. Alle Variablen sind Matrizen mit möglicherweise komplexen Elementen. Ein Vektor ist eine Matrix mit nur einer Zeile bzw. Spalte. Ein skalarer Wert ist für MATLAB entsprechend eine  $(1 \times 1)$ -Matrix. Zeichenketten werden ebenfalls als Vektoren behandelt, dessen Elemente entsprechend der ASCII-Zeichensatz-Tabelle abgespeichert werden.

Grundsätzlich gibt es zwei Anwendungsarten von MATLAB:

- Bei der interaktiven Verwendung werden Anweisungen direkt über die Tastatur eingegeben und sofort ausgeführt.
- Für umfangreichere Probleme ist es empfehlenswert, MATLAB als Programmiersprache einzusetzen. Dabei werden mehrere Anweisungen als sogenannte **m-Files** abgespeichert. m-Files sind ASCII-Files und werden mit einem Texteditor geschrieben. Sobald sie im Commandfenster aufgerufen werden, führt sie der MATLAB-Interpreter wie ein Programm aus. Darüber hinaus können auch C- und Fortran-Programme in Form von sogenannten **mex-Files** von MATLAB ausgeführt werden.

Für spezielle Anwendungen gibt es Toolboxes. Das sind Bibliotheken von m-Files zu bestimmten Aufgabenbereichen. So sind zum Beispiel elementare Befehle zum symbolischen Rechnen in der *Symbolic Math Toolbox* enthalten.

Dieses Skript zum Praktikum MATLAB/Simulink I orientiert sich an der MATLAB/Simulink-Version 2015b. Insbesondere bei den Versuchen 5 (Simulink) und 6 (SISO-Tool) kann die Bedienoberfläche bei älteren und neueren MATLAB-Versionen leicht abweichen.

---

## 1.2 Systembefehle

---

Nach dem Starten von MATLAB öffnet sich das Commandfenster (Command Window) mit einem Prompt. Im Workspace werden immer die aktuellen Variablen abgespeichert. Mit dem Befehl `whos` lässt sich jederzeit der Inhalt des Workspace auslesen.

Mithilfe von `doc` und `help` lassen sich die Dokumentation bzw. Hilfe zu den einzelnen Befehlen in MATLAB aufrufen. `doc` öffnet hierbei die HTML Dokumentation, mit `help` wird im Command Window eine kurze Beschreibung des Befehls ausgegeben.

---

Mit `edit` wird der MATLAB-Editor gestartet. Hier können Befehlsfolgen eingegeben und als m-File gespeichert werden. MATLAB unterscheidet dabei zwischen zwei Arten von Dateien: *Skripte* und *Funktionen*. Ein *Skript* kennt alle aktuellen Variablen (den Workspace) und unterscheidet sich somit nicht von einer Eingabe per Hand direkt in die MATLAB-Oberfläche.

Eine *Funktion* kennt den Workspace nicht, alle Variablen (Eingangs- und Ausgangsgrößen) müssen explizit übergeben werden.

Der Befehl `clear` löscht alle Variablen aus dem Workspace, `clear all` löscht zusätzlich auch alle Funktionen, globale Variablen und MEX Links. `clc` löscht dagegen nur das Command Window.

---

### 1.3 Grundrechenarten

---

Der Befehl `help ops` zeigt alle in MATLAB bekannten Operationen an. Neben den arithmetischen Operationen sind das Vergleichsoperationen, logische Operationen und noch einige spezielle, auf die im weiteren Verlauf des Kurses eingegangen wird.

Die arithmetischen Standardoperatoren, die für Zahlen (Skalare) und Matrizen gelten, lauten:

Operator	Operation
+	Addition
-	Subtraktion
*	Multiplikation
/	Rechtsdivision
\	Linksdivision
^	Potenzieren
.'	Transponieren
'	konjugiert-komplexes Transponieren

**Hinweis:** Beachten Sie den Unterschied zwischen `'` (konjugiert-komplexes Transponieren) und `.'` (normales Transponieren). Es empfiehlt sich immer, wenn normales Transponieren gemeint ist, den Operator `.'` zu verwenden, auch wenn nur mit reellen Matrizen gerechnet wird.

Für elementweise Operationen an Matrizen gibt es spezielle Punkt-Operatoren:

`.*`      `./`      `.\`      `.^`

Vergleiche und logische Operationen haben in MATLAB `„true“` oder `„false“` als Ergebnis. Werden diese logischen Werte in arithmetischen Ausdrücken verwendet, werden diese von MATLAB als 1 bzw. 0 interpretiert.

Vergleichsoperationen können auch mit ganzen Matrizen durchgeführt werden, das Ergebnis ist dann eine Matrix mit `true`- und `false`-Werten als Einträge. Damit lassen sich sehr einfach Filter für Daten realisieren.

MATLAB kennt u. a. die folgenden logischen Operatoren:

<code>&amp;</code>	und	siehe: <code>help and</code>
<code> </code>	oder	siehe: <code>help or</code>
<code>~</code>	nicht	siehe: <code>help not</code>
<code>xor</code>	entweder oder	siehe: <code>help xor</code>

Daneben gibt es noch die abkürzende logische Operatoren `&&` und `||`, die jedoch nur auf skalare Größen angewandt werden können. Bei diesen wird der rechte Operator nur dann ausgewertet, wenn dies nötig ist.

---

Als Vergleichsoperatoren stehen zur Verfügung:

==	gleich	siehe: help eq
~=	nicht gleich	siehe: help ne
<	kleiner als	siehe: help lt
>	größer als	siehe: help gt
<=	kleiner oder gleich	siehe: help le
>=	größer oder gleich	siehe: help ge

Mit `help elfun`, `help elmat` und `help specfun` erfolgt die Ausgabe einer Liste aller in MATLAB vordefinierten Funktionen. `elfun` gibt hierbei die trigonometrischen, exponentiellen, komplexen und Funktionen zum Runden aus, `specfun` eine Liste spezieller Funktionen (Bessel, Airy, Fehlerfunktion, ...) und `elmat` Funktionen für Matrizenrechnung. Vordefinierte Konstanten sind `pi`, `1i` („Eins-i“) bzw. `1j` für die imaginäre Einheit  $\sqrt{-1}$  sowie die Double-Werte `inf`, `-inf` und `nan`.

**Hinweis:** MATLAB lässt zu, dass alle Funktionen und Konstanten durch Variablen überschrieben werden können. Dies zeigt das folgende Beispiel:

```
>> pi
ans =
    3.1416
>> pi = 3;
>> pi
pi =
    3
```

Erst wenn man mit `clear pi` die Variable `pi` löscht, kann wieder auf die ursprüngliche Funktion zurückgegriffen werden.

```
>> clear pi
>> pi
ans =
    3.1416
```

Auf gleiche Weise würde eine Funktion mit dem Namen `pi.m` dazu führen, dass die ursprüngliche Konstante „versteckt“ ist. Dieses Verhalten kann zu schwer zu findenden Fehlern führen.

Hat man den Verdacht, dass ein solcher Fall auftritt, kann mit `which name -all` geprüft werden, welche Funktion oder Variable `name` aktuell von MATLAB verwendet werden würde:

```
% Im aktuellen Pfad ist eine Funktion "pi.m" vorhanden:
% function r = pi(), r = 1; end
>> pi = 3
>> which pi -all
pi is a variable.
D:\Matlab\startupdirs\2016a\pi.m % Shadowed
built-in (C:\Program Files\MATLAB\R2016a\toolbox\matlab\elmat\pi) % Shadowed
```

**Hinweis:** Um für die imaginäre Einheit dieses Problem zu umgehen, welches immer dann auftreten kann, wenn man eine Variable `i` oder `j` genannt hat, sollte für die imaginäre Einheit grundsätzlich `1i` oder `1j` anstelle von `i` bzw. `j` geschrieben werden!

Das Grundkonstrukt der MATLAB-Sprache ist eine Zuweisung:

---

[Variable = ] Ausdruck

Ein Ausdruck ist zusammengesetzt aus Variablennamen, Operatoren und Funktionen. Das Resultat des Ausdrucks ist eine Matrix, welche der angeführten Variablen auf der linken Seite des Gleichheitszeichens zugeordnet wird. Bei fehlender Angabe einer Variablen wird das Resultat stets in die Variable *ans* gespeichert. Endet die Zuweisung mit einem Semikolon, wird die Operation zwar ausgeführt, das Resultat jedoch nicht auf den Bildschirm ausgegeben. Bei Zuweisungen, die länger als eine Zeile umfassen, müssen die fortzusetzenden Zeilen mit drei Punkten beendet werden.

Bevor man beginnt zu rechnen, muss zuerst das gewünschte Ausgabeformat festgelegt werden. Neben den Standardformaten `format short` und `format long` können auch andere verwendet werden (`format short e`, `format short g`, ...). Hierbei sei auf das entsprechende help-File verwiesen (`help format`). Anzumerken ist, dass `format` das Ausgabeformat festlegt, die Berechnungen von MATLAB aber nicht beeinflusst werden (durch Runden o. Ä.).

MATLAB rechnet standardmäßig mit double-Werten (64 bit-Gleitkommazahlen). Dadurch treten prinzipiell Rundungsfehler bei der Eingabe von Zahlen und bei den Berechnungen auf. So ergibt

```
>> sin(pi)
ans =
    1.2246e-16
```

und nicht exakt Null. Der ausgegebene Wert kann aber im Rahmen der Rechengenauigkeit als Null angesehen werden. Bei den Berechnungen im Rahmen dieses Praktikums ist die begrenzte Rechengenauigkeit jedoch unkritisch. Dagegen können beim Umgang mit großen oder ungünstig parametrisierten Gleichungssystemen durchaus Probleme auftreten, insbesondere wenn man numerisch ungünstige Verfahren verwendet.

Mit der optionalen „Symbolic Toolbox“ besitzt MATLAB auch die Möglichkeit symbolisch zu rechnen. (Jedoch hilft das in der Regel nicht, Probleme zu umgehen, die im Zusammenhang mit der endlichen Rechengenauigkeit der numerischen Berechnungen auftreten. In diesem Fall sind die Dimensionen der Matrizen meist so groß, dass eine symbolische Berechnung nicht durchführbar ist.)

---

## 1.4 Matrizen

---

---

### 1.4.1 Eingabe von Matrizen

---

Prinzipiell können Matrizen auf verschiedene Arten eingegeben werden:

1. Explizite Eingabe der Matrixelemente,
2. Aufruf einer matrixgenerierenden Funktion, oder
3. Laden aus einer externen Datei.

---

#### Explizite Eingabe

---

Die Grundregeln bei der Eingabe von Matrixelementen sind:

- Elemente einer Matrixzeile werden durch Leerzeichen oder Komma getrennt.
- Das Ende einer Zeile wird durch ein Semikolon ; angegeben.
- Die ganze Liste von Zahlen wird in eckige Klammern [ ] geschrieben.

---

Da Spaltenvektoren ( $n \times 1$ )-Matrizen und Zeilenvektoren ( $1 \times n$ )-Matrizen sind, gelten die Vorgaben in analoger Weise für Vektoren.

```
>> u = [3; 1; 4], v = [2 0 -1], s = 7
u =
     3
     1
     4

v =
     2     0    -1

s =
     7
```

Ein weiteres nützliches Feature ist der „:“-Operator (help colon). So erzeugt

```
(i : j : k)
```

einen Vektor mit erstem Element  $i$ , gefolgt von  $i + j$ ,  $i + 2j$ , ... für  $j > 0$  bis zu einem Element, das gerade noch  $\leq k$  ist und für  $j < 0$  bis zu einem Element, das gerade noch  $\geq k$  ist.

```
>> 100:-7:50
ans =
    100     93     86     79     72     65     58     51
```

---

## Matrixgenerierende Funktionen

---

Matixgenerierende Funktionen sind unter help elmat angegeben, die Wichtigsten sind dabei

- eye: Einheitsmatrix
- zeros: Nullmatrix
- ones: Einser-Matrix
- rand: zufällige Matrix (gleichverteilte Zufallsvariablen)
- randn: zufällige Matrix (normalverteilte Zufallsvariablen)

In einer nachgestellten Klammer erfolgt die Vorgabe der Matrixdimension. zeros( $n$ ,  $m$ ) generiert dabei eine ( $n \times m$ )-Matrix mit Nullen, eye( $n$ ) eine ( $n \times n$ )-Einheitsmatrix.

Der Befehl diag( $A$ ) speichert die Diagonale der Matrix  $A$  in einem Vektor. Mithilfe von triu( $A$ ) und tril( $A$ ) werden das obere bzw. untere Dreieck der Matrix ausgegeben, wobei der Rest mit Nullen aufgefüllt wird.

Der Befehl size( $A$ ) zeigt die Anzahl der Zeilen und Spalten an.

---

## Laden aus einer externen Datei

---

Beim Laden einer Matrix aus einer externen Datei muss unterschieden werden, ob von einem m-File oder aus binären Dateien bzw. Textdateien geladen wird.

Wenn innerhalb eines Scripts „matrix.m“ eine Matrix  $A$  über

---

```
A = [ 1 2 3; 4 5 6; 7 8 9 ]
```

definiert ist, dann wird vom Command Window aus mit der Eingabe von `>> matrix` erreicht, dass der Variablen A die vorher definierte Matrix A zugeordnet wird.

Über `load dateiname` kann der Inhalt einer beliebigen Textfile geladen werden. Hierbei muss darauf geachtet werden, dass die Datei eine rechteckige Tabelle mit Zahlen enthält, die durch Leerzeichen getrennt sind und eine Textzeile pro Matrixzeile enthalten. Außerdem muss jede Matrixzeile eine gleiche Anzahl von Elementen haben.

Der Import Wizard (File→Import Data...) erlaubt, weitere verschiedene Formate in MATLAB zu laden.

---

### 1.4.2 Mehrdimensionale Matrizen

---

MATLAB lässt auch Matrizen mit mehr als zwei Dimensionen zu. (Wobei dann nicht mehr alle Funktionen auf diese mehrdimensionalen Matrizen anwendbar sind.) Diese sind beispielsweise geeignet, um Werte einer matrixwertigen Funktion zu verschiedenen Zeitpunkten in einer Variablen zu speichern.

Folgender Befehl generiert z. B. eine  $(n \times m \times d)$ -Matrix, deren Elemente gleichverteilte Zufallszahlen mit Werten zwischen 0 und 1 sind:

```
>> M = rand(2, 4, 3)
M(:,:,1) =
    0.8147    0.1270    0.6324    0.2785
    0.9058    0.9134    0.0975    0.5469

M(:,:,2) =
    0.9575    0.1576    0.9572    0.8003
    0.9649    0.9706    0.4854    0.1419

M(:,:,3) =
    0.4218    0.7922    0.6557    0.8491
    0.9157    0.9595    0.0357    0.9340
```

---

### 1.4.3 Matrixoperationen

---

Ein Element der Matrix A in Zeile *i* und Spalte *j* wird durch `A(i, j)` bezeichnet. Entsprechend kann auf das jeweilige Element zugegriffen und für weitere Berechnungen benutzt werden. Eine weitere Möglichkeit bietet der bereits erwähnte „:“-Operator. Er greift auf ganze Teile einer Matrix zu.

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Die Eingabe von

```
>> A(2,1:3)
ans =
     5    10    11
```

---

liefert somit die Elemente 1 bis 3 der zweiten Zeile,

```
>> A(2:3,2:3)
ans =
    10    11
     6     7
```

die Elemente 2 und 3 der zweiten und dritten Zeile,

```
>> A(3:end,3)
ans =
     7
    14
```

die Elemente 3 bis zum Schluss (hier 4) der dritten Spalte und

```
>> A(4,:)
ans =
     4    15    14     1
```

liefert alle Elemente der vierten Zeile.

---

## Grundrechenarten

---

Die prinzipiellen Grundrechenarten werden bereits in Abschnitt 1.3 vorgestellt. Addition und Subtraktion von Matrizen geschieht elementweise, demzufolge müssen die Matrizen dieselben Dimensionen besitzen. Für die elementweise Berechnung muss ein Punkt (.) vor den jeweiligen Operator gesetzt werden.

---

## Elementare Matrixfunktionen

---

### Vektorprodukte und Transponierte

Ein Zeilenvektor und ein Spaltenvektor können miteinander multipliziert werden. Als Ergebnis kommt entweder ein Skalar (das innere Produkt oder Skalarprodukt) oder eine Matrix, das äußere Produkt, heraus.

```
>> u = [3; 1; 4];
>> v = [2 0 -1];
>> x = v * u
x =
     2
>> X = u * v
X =
     6     0    -3
     2     0    -1
     8     0    -4
```

Bei Matrizen spiegelt die Transposition die Elemente an der Diagonale. Seien die Elemente der Matrix **A** mit  $(m \times n)$  bezeichnet als  $a_{i,j}$ ,  $1 \leq i \leq m$  und  $1 \leq j \leq n$ , dann ist die Transponierte von **A** die  $(n \times m)$ -Matrix **B** mit den Elementen  $b_{j,i} = a_{i,j}$ . Zur Berechnung der Transponierten gibt es in MATLAB den Befehl `()'`.

```
>> A = [1 2 3; 4 5 6; 7 8 9];  
>> B = A.'
```

```
B =  
     1     4     7  
     2     5     8  
     3     6     9
```

Die Anwendung der Transposition macht aus einem Zeilenvektor einen Spaltenvektor. Für die Berechnung der konjugiert-komplex transponierten Matrix  $A^H$  gibt es den Befehl `()'` (`A'`). Eine elementweises Bestimmen der konjugiert-komplexen Werte (ohne die Durchführung der Transposition) kann über den Befehl `conj()` erfolgen.

### Inverse und Determinante

**A** ist genau dann invertierbar, wenn die Determinante von **A** ungleich Null ist. Mit `det(A)` ist in MATLAB der Determinantenwert einer Matrix **A** erhältlich. Die Inverse kann mit `inv(A)` berechnet werden.

### Lösung von linearen Gleichungssystemen

Existiert die Inverse  $A^{-1}$ , so ist die Lösung **x** von  $A \cdot x = b$  gegeben durch  $x = A^{-1} \cdot b$ .

Aus numerischen Gründen sollte jedoch die Lösung *nicht* über  $x = \text{inv}(A) * b$  bestimmt werden!

Eine numerisch sicherere Methode wird von MATLAB automatisch angewandt, wenn  $x = A^{-1} \cdot b$  als

```
x = A \ b
```

geschrieben wird.

Das Symbol `\` bezeichnet die sogenannte Linksdivision (left division). Der Name kommt daher, dass nun der Nenner links steht und der Bruchstrich nach links geneigt ist. Hier steht `A\` sozusagen für die Inverse von **A**:  $A \backslash B \hat{=} \text{inv}(A) * B$ . Analog gilt für die Rechtsdivision  $A/B \hat{=} A * \text{inv}(B)$ .

Im Fall von Skalaren ist die Linksdivision  $2 \backslash 3$  gleich der Rechtsdivision  $3/2$  (die Multiplikation von Skalaren ist kommutativ). Bei Matrizen aber ist  $\text{inv}(A) * B$  (= Linksdivision) im Allgemeinen nicht das Gleiche wie  $B * \text{inv}(A)$  (= Rechtsdivision).

**Hinweis:** Die Verwendung von  $x = A \backslash b$  anstelle von  $x = \text{inv}(A) * b$  hat verschiedene Vorteile:

- $A \backslash b$  verwendet aus numerischer Sicht bessere Verfahren. Dies betrifft zum einen die Geschwindigkeit, vor allem aber die Genauigkeit der Lösung.
- Zweitens liefert die Eingabe von  $x = A \backslash b$  auch dann eine Lösung, wenn das Gleichungssystem nicht eindeutig lösbar ist (und daher die Inverse nicht existiert). Im Fall eines überbestimmten Systems wird eine Lösung angegeben, bei der der quadratische Fehler von  $Ax - b$  minimal ist.

### Eigenwertberechnung

Mit dem Befehl `eig(A)` können die Eigenwerte einer Matrix **A** bestimmt werden. Wenn `eig` mit einer Rückgabewerten aufgerufen wird, dann gibt `eig` einen Vektor mit den Eigenwerten zurück. Wird die Funktion mit zwei Rückgabewerten aufgerufen, dann ist der erste Rückgabewert die Rechtseigenvektormatrix, die spaltenweise die Eigenvektoren enthält, und der zweite Rückgabewert eine Diagonalmatrix mit den Eigenwerten als Diagonalelemente.

```
>> A = [-1, 1, 1; 0, -2, -3; 4, 0, 3];  
>> D = eig(A)
```



```

D =
    3.4040
   -0.1824
   -3.2217

>> [V, D] = eig(A)
V =
   -0.0880   -0.3811    0.5060
    0.4835   -0.7907   -0.7988
   -0.8709    0.4791   -0.3253

D =
    3.4040         0         0
         0   -0.1824         0
         0         0   -3.2217

>> V * D / V      % = V * D * inv(V), "Gegenprobe": muss wieder A ergeben
ans =

   -1.0000    1.0000    1.0000
         0   -2.0000   -3.0000
    4.0000    0.0000    3.0000

```

## Weitere Funktionen

Für mathematische Berechnungen sind weitere Funktionen wichtig, wie z. B. `trace()`, die die Spur einer Matrix liefert, `rank()`, welche den Rang liefert, und `norm()`, um die Norm einer Matrix zu bestimmen.

`null()` liefert den Kern der zugehörigen Abbildung. Falls die Matrix vollen Rang hat, ist die Determinante von 0 verschieden und die zugehörige Abbildung besitzt einen trivialen Kern. Da der Befehl `null()` eine Basis des Kerns zurückliefert, ist das Ergebnis in diesem Fall eine leere Matrix.

Für die Berechnung von linearen Gleichungssystemen sind neben den bereits erwähnten Befehlen weitere gebräuchlich. Mit `cond()` lässt sich die Konditionszahl berechnen und `chol()`, `lu()` und `qr()` erzeugen die *Cholesky*-, *LQ*- und die *QR-Zerlegung*.

---

### 1.4.4 Vektorfunktionen

---

Nützliche Funktionen sind `prod()` und `sum()`. Mithilfe dieser können die Produkte bzw. die Summe der Werte eines Vektors berechnet werden. Zum Bestimmen des maximalen und minimalen Eintrages gibt es die Funktionen `max()` und `min()`. Der arithmetrische Mittelwert lässt sich mit `mean()` berechnen und die Standardabweichung mit `std()`.

---

## 1.5 Polynome

---

### 1.5.1 Allgemeine Auswertung

---

Polynome sind besonders einfache mathematische Funktionen, die vielfach Anwendung finden. Ein Polynom, etwa  $y = -x^3 + 3 \cdot x^2 + 4$ , kann über Punktoperationen für einen bestimmten Vektor  $x$  errechnet werden. Einfacher kann ein Polynom in MATLAB durch einen Vektor mit seinen Koeffizienten angegeben werden.

Die Nullstellen des Polynoms lassen sich mit

```
>> roots([-1 3 0 4])
```

bestimmen. Hierbei ist zu beachten, dass in der Klammer links der Koeffizient der höchsten Potenz des Polynoms steht. Entsprechend kann aus beliebigen Nullstellen mit

```
>> poly ([1 2 3])
```

ein Polynom bestimmt werden. Angewendet auf eine Matrix liefert der Befehl entsprechend die Koeffizienten des charakteristischen Polynoms. Die Auswertung von Polynomen erfolgt mit dem Befehl `polyval`.

Im Folgenden soll das Polynom  $y = -x^3 + 3 \cdot x^2 + 4$  für die Werte  $x = -1, 0, 1, 2, 3, 4$  und  $5$  berechnet werden. Hierfür wird  $x$  als Vektor definiert.

```
>> x = (-1:5);  
>> koeffizienten = [-1 3 0 4];  
>> polyval(koeffizienten, x)  
ans =  
      8      4      6      8      4     -12     -46
```

Die Ableitung eines Polynoms  $p$  kann mit `polyder(p)` bestimmt werden. Zum *Fitten* eines Polynoms in vorgegebene Punkte gibt es den Befehl `polyfit(x, y, n)`, wobei  $x$  und  $y$  Vektoren der  $x$ - bzw.  $y$ -Koordinaten der Punkte sind und  $n$  die Ordnung des zurückgelieferten Polynoms angibt.

---

### 1.5.2 Polynomdivision und Multiplikation

---

Wenn zwei Polynome vorhanden sind, kann eine Multiplikation oder Division der Polynome mit MATLAB folgendermaßen durchgeführt werden:

Der Operator `conv(a, b)` führt die Polynommultiplikation von  $a$  und  $b$  durch. Das Ergebnis ist ein Polynom in Form eines Zeilenvektors, der die Koeffizienten des Ergebnis-Polynoms enthält.

Der Operator `[q, r] = deconv(a,b)` führt eine Polynomdivision  $a/b$  aus. Das Ergebnis sind die beiden Vektoren  $q$  und  $r$  zusammengefasst zu einer Matrix. Hierbei ist  $q$  das entstehende Polynom und  $r$  enthält den bei der Polynomdivision entstehenden Rest.

---

### 1.5.3 Partialbruchzerlegung

---

Die Partialbruchzerlegung wird in MATLAB über `residue` berechnet. Der Befehl `[r, p, k] = residue(b, a)` entspricht folgender Darstellung:

$$\frac{b(x)}{a(x)} = \sum_{i=1}^n \frac{r_i}{x - p_i} + \sum_{j=1}^{m-n+1} k_j x^{m-n+1-j}$$

Hierbei ist  $n$  der Nennergrad und  $m$  der Zählergrad. Wenn  $m < n$  ist, existiert nur die erste Summe und  $k$  ist ein leerer Vektor.

Als Beispiel wird die Partialbruchzerlegung von

$$\frac{4x}{x^4 + 2x^3 + 2x^2 + 2x + 1}$$

berechnet.

```
>> b = [4 0];
>> a = [1 2 2 2 1];
>> [r, p, k] = residue(b, a)
r =
    0.0000 + 0.0000i
   -2.0000 + 0.0000i
   -0.0000 - 1.0000i
   -0.0000 + 1.0000i

p =
   -1.0000 + 0.0000i
   -1.0000 + 0.0000i
   -0.0000 + 1.0000i
   -0.0000 - 1.0000i

k =
[]
```

Der Befehl liefert die Darstellung mit konjugiert komplexem Polpaar

$$\frac{0}{x+1} + \frac{-2}{(x+1)^2} + \frac{-j}{x-j} + \frac{j}{x+j}$$



---

# Versuch 2

---

## 2.1 Einleitung

---

Für umfangreiche Berechnungen ist es empfehlenswert, MATLAB als Programmiersprache einzusetzen. Dabei werden mehrere Anweisungen als sogenannte m-Files abgespeichert. m-Files sind ASCII-Files und werden mit einem Texteditor geschrieben. Sobald sie im Command Window aufgerufen werden, führt sie der MATLAB-Interpreter wie ein Programm aus.

Eine weitere Stärke von MATLAB ist die Darstellung zweidimensionaler (2D) und dreidimensionaler (3D) Daten. Es gibt sehr viele Funktionen, um diese Daten darzustellen und die Grafiken nach Wunsch zu editieren. Diese sind in ihrer Basisfunktionalität sehr einfach auf Daten in Vektor- oder Matrixform anzuwenden.

---

## 2.2 m-Files und ihre Anwendung

---

Es ist sinnvoll, spätestens bei größeren Berechnungen bzw. Ablaufstrukturen, den Code innerhalb von m-Files zu erstellen und auszuführen (Stapelbetrieb), als jeden Befehl einzeln in das Command Window einzugeben (interaktiver Betrieb). Dateien mit MATLAB Anweisungen erhalten den Suffix \*.m und werden dementsprechend als m-Files bezeichnet. So können die MATLAB-Anweisungen mit jedem beliebigen Editor in eine ASCII-Datei geschrieben werden. Es empfiehlt sich, den in MATLAB integrierten Editor zu verwenden.

Vom Command Window aus wird über *File* → *New* → *m-File* der MATLAB-Editor aufgerufen, der sich mit einem eigenen Eingabe-Fenster öffnet. Abgespeichert wird die Datei über *File* → *Save* direkt unter einem gewünschten Namen. Hierbei sollte darauf geachtet werden, dass die Datei mit dem Suffix \*.m abgespeichert wird.

Das Aufrufen eines m-Files erfolgt durch Eingabe des Dateinamens ohne Suffix. MATLAB prüft dann der Reihe nach, ob dieser Name:

1. zu einer Variablen,
2. zu einem m-File im aktuellen Arbeitsverzeichnis,
3. zu einem m-File in einem Verzeichnis des MATLAB Suchpfades
4. zu einer Standardfunktion (`built in function`),

gehört. Das erste Objekt, auf das dies zutrifft, wird genommen und von MATLAB ausgeführt. m-Files entsprechen Programmen, Funktionen, Subroutinen oder Prozeduren in anderen Programmiersprachen. Sie bestehen zu einem großen Teil aus einer Folge von MATLAB-Befehlen. Durch die Benutzung von m-Files ergeben sich eine Reihe von Möglichkeiten, die das Befehlsfenster nicht bietet:

- Experimentieren mit einem Algorithmus durch Editieren des m-Files anstelle der permanenten Eingabe langer Befehlssequenzen
- Permanente Aufzeichnung eines Experiments
- Aufbau von Dienstprogrammen zur späteren Benutzung

Prinzipiell gibt es zwei Arten von m-Files: *Skripte* und *Funktionen*.

---

## 2.2.1 Skripte und Funktionen

---

Skripte bestehen aus Befehlsfolgen, die ohne Angabe von Argumenten aufgerufen werden. Der Aufruf geschieht entweder durch Eingabe des Namens in der Kommandozeile oder durch Klicken auf das *run-Icon* in der Kopfzeile des MATLAB-Editorfensters. Skriptfiles werden einmalig durchlaufen und die darin enthaltenen Befehle so ausgeführt (interpretiert), als ob sie am Bildschirm eingetippt werden. Sie haben keine Eingabe- oder Ausgabeparameter.

Während ein Skript somit immer das gleiche Ergebnis produziert, kann eine Funktion mit einem Argument aufgerufen und die Ergebnisse von Berechnungen als Variablen zurückgeben werden. Es ist wichtig, dass das m-File den gleichen Namen wie die Funktion hat. Prinzipiell unterscheiden sich Skripte und Funktionen nicht in der Dateinamenserweiterung (\*.m).

Mit dem Befehl `what` erfolgt die Auflistung der m-Files im gegenwärtigen Verzeichnis. Der Befehl `type` listet den Inhalt eines m-Files auf.

Funktionen sind durch das Wort `function` in der ersten Zeile des m-Files gekennzeichnet und besitzen Eingabe- und Ausgabeparameter. Sie arbeiten mit einem eigenen Speicherbereich, unabhängig vom Arbeitsspeicher, der vom MATLAB-Befehlsfenster sichtbar ist. Innerhalb von Funktionen definierte Variablen sind lokal und gehen somit beim Rücksprung aus der Funktion verloren. Neben dem Stichwort `function` erscheinen auf der ersten Zeile des m-Files die Ausgabe- und die Eingabeparameter. MATLAB-Funktionen können mit einer variablen Anzahl von Parametern aufgerufen werden.

In diesem Beispiel wird eine Funktion deklariert, die zwei Eingabeparameter `parameter1` und `parameter2` sowie die Ausgabeparameter `rueck1` und `rueck2` besitzt:

```
function [rueck1, rueck2] = myfunction(parameter1, parameter2)
```

Wenn diese Funktion in einem m-File „`myfunction.m`“ gespeichert wird, kann diese beispielsweise über

```
[value1, value2] = myfunction(par1, par2)
```

aufgerufen werden. Die Variablennamen in der Deklaration und im Aufruf müssen also nicht gleich sein, sondern die Zuordnung geschieht einzig über die Position. (So wird der Wert von `par1` der Variablen `parameter1` innerhalb der Funktion zugewiesen. Umgekehrt wird der Wert der innerhalb der Funktion berechneten Variablen `rueck1` der Variablen `value1` zugewiesen.)

Eine Funktion darf in MATLAB mit weniger Eingabeparametern als in der Deklaration aufgerufen werden. Dies würde erst dann zu einem Fehler führen, wenn innerhalb der Funktion auf eine Variable zugegriffen wird, der beim Funktionsaufruf kein Wert zugewiesen wurde. Innerhalb einer Funktion steht die Größe `nargin` zur Verfügung, die angibt, mit wie vielen Eingabeparametern die Funktion aufgerufen wurde.

Genauso kann eine Funktion in MATLAB mit weniger Ausgabeparametern als in der Deklaration aufgerufen werden. Mit `nargout` lässt sich innerhalb einer Funktion feststellen, mit wie vielen Ausgabeparametern diese aufgerufen wurde.

Als Beispiel sei folgender MATLAB Code gegeben:

---

### Listing 2.1: Funktion `mittelwerte`.

---

```
1 function [out1, out2] = mittelwerte(x, wahl)

    if nargin == 1
```

---

```

        out1=mean(x);
        out2=prod(x).^(1/length(x));
6
    elseif nargin == 2
        if wahl == 'g'
            out1 = prod(x).^(1/length(x));
        else
11            out1 = mean(x);
        end
        out2=[ ];
    end
16 end

```

---

Wird nur die Variable x übergeben, werden beide Mittelwerte ausgegeben:

```

>> [g, a] = mittelwerte([1, 2, 3])
g =
    2

a =
    1.8171

```

Sobald auch der zweite Parameter wahl übergeben wird, wird nur ein Mittelwert berechnet, und zwar der geometrische Mittelwert, wenn 'g' übergeben wird, ansonsten der arithmetische Mittelwert:

```

>> m = mittelwerte([1, 2, 3], 'g')
m =
    1.8171
>> m = mittelwerte([1, 2, 3], 'a')
m =
    2

```

Ein eigenes Funktionen-File wird wie eine eingebaute Funktion behandelt und kann somit über den Befehl `help myfunction` aufgerufen werden. Hierbei werden die in der Datei abgespeicherten Kommentarzeilen bis zur ersten Nichtkommentarzeile am Bildschirm ausgegeben.

Ein Kommentar kann durch das Zeichen % erzeugt werden. Sollen mehrere Zeilen auskommentiert werden kennzeichnet man den Beginn des Kommentars mit %{ (in einer eigenen Zeile) und das Ende des Kommentars mit %} (muss ebenfalls in einer eigenen Zeile stehen). Alternativ kann man den zu kommentierenden Teil markieren und den Tastenkürzel Strg-R benutzen. Analog kann mit Strg-T ein auskommentierter Abschnitt, welcher zuvor ausgewählt wurde, einkommentiert werden.

---

### 2.2.2 Schleifen

---

MATLAB hat verschiedene Konstrukte, um den Programmablauf steuern zu können. Diese Elemente von MATLAB können direkt im Befehlsfenster eingegeben oder in MATLAB-Funktionen verwendet werden.

---

## For-Schleife

---

Eine For-Schleife funktioniert in MATLAB wie in vielen Hochsprachen. Die Zählvariable durchläuft dabei alle Werte eines angegebenen Vektors. Sie ist eines der nützlichsten Konstrukte in MATLAB.

Die Form des For-Konstrukts ist

```
for Variable = Ausdruck
    Zuweisungen
end
```

Sehr oft ist der Ausdruck ein Vektor der Form `[i:j:k]`. Die Zuweisungen werden ausgeführt, wobei die Variable einmal gleich jedem Element des Ausdruckes gesetzt wird.

---

## While-Schleife

---

Die allgemeine Form der While-Schleife, die solange ausgeführt wird, bis die Relation falsch ist, lautet

```
while Relation
    Zuweisungen
end
```

In MATLAB ist eine Relation eine Matrix. Wenn diese Matrix mehr als ein Element hat, so werden die Statements im While-Konstrukt genau dann ausgeführt, wenn jede einzelne Komponente der Matrix den Wert `true` hat.

---

## If-else-Konstrukt

---

Die einfachste Form des If-Konstrukts ist

```
if logischer Ausdruck
    Zuweisungen
end
```

Die Zuweisungen werden ausgeführt, falls der logische Ausdruck wahr ist. Zuweisungen, die nur ausgeführt werden sollen, wenn der logische Ausdruck falsch ist, können nach einem `else` platziert werden.

```
if logischer Ausdruck
    Zuweisungen
else
    Zuweisungen
end
```

Es können auch mehrere logische Ausdrücke geprüft werden. Es werden dann die Zuweisungen der ersten Bedingung ausgeführt, die `true` ist:

```
if logischer Ausdruck
    Zuweisungen
elseif logischer Ausdruck
    Zuweisungen
else
    Zuweisungen
end
```



---

## Switch-case-Konstrukt

---

Wenn eine Variable eine feste Anzahl von Werten annehmen kann und für jeden von ihnen eine bestimmte verschiedene Befehlsfolge ausgeführt werden soll, bietet sich das Switch-case-Konstrukt zur Implementierung an. Dieses hat die Form

```
switch Variable
case Wert1
    Befehle Zuweisungen
case Wert2
    Befehle Zuweisungen
...
otherwise,
    Befehle
end
```

---

### 2.2.3 Ein- und Ausgabe von Text

---

Mit `help iofun` lassen sich alle von MATLAB verwendeten Funktionen für Ein- und Ausgabebefehle anzeigen. Hier wird nur auf ein paar wenige wichtige Befehle eingegangen, für weitere Informationen sei auf die entsprechende Hilfe verwiesen.

Mit der `disp`-Funktion erfolgt eine simple Textausgabe. Die Syntax hierfür ist `disp(' ')`. Mit der `input`-Funktion kann dagegen der Nutzer per Aufforderung einzelne Variablen eingeben. Mit einem zusätzlichen Argument `'s'` wird die Eingabe als Zeichenkette interpretiert.

```
>> x = input('Startwert: ')
Startwert: 15
x =
    15
>> titel = input('Ueberschrift: ', 's')
Ueberschrift: Meine erste Ueberschrift
titel =
Meine erste Ueberschrift
```

Die Standardausgabe von Rechenergebnissen bei MATLAB ist weder besonders schön noch übersichtlich. Zur besseren Steuerung der Textausgabe bietet MATLAB die Funktion `fprintf`. Wie in der Programmiersprache C gehorcht diese Funktion der Syntax `fprintf(Formatanweisung, Liste von Ausdrücken)`. Damit lassen sich Text und Werte von Variablen in beliebiger Form ausgeben. Dabei ist zu beachten, dass mit dem Ende der `fprintf` nicht automatisch in die nächste Zeile gewechselt wird. Ein Zeilenende muss explizit durch die Zeichenfolge `\n` erzeugt werden. Leerzeilen werden mit `fprintf('\n')` erzeugt.

Um Variablenwerte auszugeben, muss im Text ein Platzhalter eingegeben werden, der das Format der Ausgabe bestimmt. Für Gleitkommazahlen gibt es folgende Format-Platzhalter

- `'%e'` gibt die Zahl stets in Exponenten-Darstellung aus
- `'%f'` gibt die Zahl stets ohne Exponenten aus
- `'%g'` wechselt zwischen diesen Formaten, je nachdem, in welchem Zahlenbereich die auszugebende Zahl liegt. Außerdem werden unwichtige Nachkommastellen (Nullen) abgeschnitten.

---

Diese Platzhalter legen nur das Format und die Position der Ausgabe im laufenden Text fest. Die eigentlich auszugebenden Variablen müssen als weitere Argumente an die `fprintf` Anweisung übergeben werden. Zusätzlich zum Format kann auch eine Mindestanzahl von Stellen angegeben werden, die für die Zahl bereitgestellt werden sollen. Die Anzahl der Gesamtstellen und zusätzlich die Anzahl der Nachkommastellen kann angegeben werden, indem diese zwischen '%' und Formatzeichen geschrieben werden.

```
>> fprintf('%6.3f\n', pi)
3.142
```

Die Anweisung `%6.3f` erzeugt dabei eine 6-stellige Festkommazahl mit drei Nachkommastellen. Hierbei sei angemerkt, dass eine evtl. zu kleine Feldbreite von MATLAB automatisch angepasst wird und dass für negative Zahlen eine extra Stelle mitgerechnet wird. Bei der Anweisung `%12.3e` gibt die Ziffer nach dem Punkt die Anzahl Nachkommastellen in Exponentialnotation an. Um die Zahlen linksbündig auszurichten, muss ein Minus nach dem Prozentzeichen gesetzt werden.

`fprintf` kann nicht nur einfache Zahlen, sondern auch Vektoren und Matrizen ausgeben. Hierbei wird die entsprechende Formatanweisung für jedes Element der Matrix wiederholt. Hierbei wird spaltenweise vorgegangen, so dass die gewohnte Reihenfolge (zeilenweise) durch Transponieren der auszugebenden Matrix erzeugt werden muss. Das Skript

```
A = [1.5 2.7 3.456; 7.6 4.765 1.234; 3 4 5];
fprintf('Meine erste Matrix:\n');
fprintf('%f %f %f\n', A.');
```

liefert als Ausgabe

```
Meine erste Matrix:
1.500000 2.700000 3.456000
7.600000 4.765000 1.234000
3.000000 4.000000 5.000000
```

Unter Verwendung von `fprintf` kann mit `fopen` eine externe Datei (z. B. \*.txt) erzeugt und mit MATLAB Ausgaben geschrieben werden. Dabei wird das in Programmiersprachen übliche Konstrukt des *Streams* verwendet. Bei einem Schreib- bzw. Lesevorgang wird zuerst ein Stream zur entsprechenden Datei geöffnet. Erst danach kann der entsprechende Vorgang durchgeführt werden. Abschließend muss der Stream wieder geschlossen werden. Die Syntax zum Öffnen des Streams lautet

```
fid = fopen('dateiname.txt', 'w')
```

Das 'w' steht hierbei für Zugriffsrechte auf die Datei. Die Zugriffsrechte können anhand folgender Tabelle ausgewählt werden:

r	Lesen aus der Datei
w	Schreiben in die Datei (Erzeugen falls nötig)
a	Hinzufügen (Erzeugen falls nötig)
r+	Erst Lesen, dann Weiterschreiben (nicht erzeugen)
w+	Erst Schreiben, dann Weiterlesen
a+	Lesen und Hinzufügen (Erzeugen falls nötig)

Die Syntax, um Werte in die Datei `dateiname.txt` zu schreiben, lautet

---

```
fprintf(fid, '%6.4f', pi)
```

Mit dieser Befehlszeile kann `pi` in die Datei mit dem *Handle* `fid` geschrieben werden.

Um nun Werte oder Worte aus der Datei `dateiname.txt` auszulesen, kann der Befehl `fscanf` verwendet werden:

```
daten = fscanf(fid, format, anzahl)
```

Hierbei wird mit `format` ausgelesen und in `daten` der Inhalt gespeichert, wobei mit `anzahl` die Anzahl der zu lesenden Elemente angegeben werden kann. Der Zugriff auf die Datei wird mit dem Befehl `fclose` beendet:

```
fclose(fid)
```

---

## 2.3 Grafik in MATLAB

---

Grafiken werden in MATLAB in einem Extrafenster dargestellt. Das Fenster hat eine Nummer, die mit dem Befehl `gcf` (get current figure) herausgefunden werden kann. Diese Nummer wird *Handle* genannt und wird bei Referenzen auf die Grafik verwendet. Ein neues Fenster lässt sich mit dem Befehl `figure` erzeugen. Hat dieses die Nummer `n`, kann es jederzeit mit `figure(n)` zum aktuellen Fenster gemacht werden, in welches geplottet wird.

---

### 2.3.1 Der plot-Befehl – 2D

---

In MATLAB können mit wenigen Befehlen Daten sehr leicht grafisch dargestellt werden. Die Anweisung `plot(x, y)` erzeugt ein Grafikfenster und plottet `y` über `x`. `x` und `y` sind Vektoren, sie enthalten die `x`- und `y`-Koordinaten der zu zeichnenden Datenpunkte.

Als einführendes Beispiel wird ein Vektor `y` erzeugt, der Sinuswerte eines Vektors `x` enthält:

```
>> x = (-pi:.1:pi).';  
>> y = sin(x);  
>> plot(x, y);
```

Verschiedene Linienformen, Stärken und Farben können mit der Syntax `plot(x, y, ' ')` erzeugt werden. Innerhalb der Anführungszeichen muss dann anhand der folgenden Tabelle die jeweilige Eigenschaft vorgegeben werden:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

---

mit `plot(x, y, 'r--')` lässt sich beispielsweise ein gestrichelter roter Graph erzeugen.

Soll die Anzahl der gezeichneten Datenpunkte vorgegeben werden, so lässt sich der Vektor `x` am besten mit `linspace` erzeugen:

```
x = linspace(-pi, pi, 30);
```

Hier besteht der Vektor `x` aus 30 Werten im gleichen Abstand, wobei der erste Wert  $-\pi$  und der letzte Wert  $\pi$  ist. Fehlt die Angabe für die Anzahl der Werte, so wird standardmäßig 100 angenommen.

Alternativ kann ein Plot auch mit der Funktion `fplot(f, lims)` erzeugt werden. `f` ist hierbei entweder der Name einer als m-File abgelegten Funktion oder ein in Apostrophs eingeschlossener String, der die Funktion enthält. `lims` ist ein Vektor, der in der Form `[xmin xmax ymin ymax]` die Grenzen des Anzeigebereichs enthält.

Für das vorherige Beispiel lautet die Syntax

```
>> fplot('sin(x)', [-pi, pi]);
```

Bei diesem Befehl sucht sich MATLAB selbst die benötigten Datenpunkte (somit ist die Definition als `x`-Vektor nicht erforderlich), sondern nur die Grenze des `x`-Bereichs. Den `y`-Achsenabschnitt definiert MATLAB sich, falls nicht angegeben, selbst.

Mit den Befehlen `loglog()`, `semilogx()` und `semilogy()` können Plots mit doppelt bzw. einfach logarithmischer Achsenskalierung ausgegeben werden.

Um ein vorgegebenes Polygon durch die Eckpunkte

$$(x_i, y_i)_{i=1}^n$$

zu erzeugen, gibt es den `fill(x, y, 'c')` Befehl, wobei mit dem dritten Parameter die Farbe bestimmt wird.

Einen Überblick über die Befehle zur Beschriftung des Plots gibt folgende Tabelle:

<code>title('Ueberschrift')</code>	Titel
<code>xlabel('Zeit t')</code>	x-Achsenbeschriftung
<code>ylabel('Signal y')</code>	y-Achsenbeschriftung
<code>legend(...)</code>	Legende
<code>axis([2 5 -3 3])</code>	Achsenskalierung xmin xmax ymin ymax
<code>grid on, grid off</code>	Gitternetz an/aus

Neben diesen gibt es zur Nachbearbeitung der Grafik im Plotfenster unter *Menü* → *Tools* viele weitere Editierfunktionen.

Um mehrere Graphen zu plotten, gibt es prinzipiell drei Möglichkeiten:

- Mehrere Graphen in mehrere Plotfenster:  
Mithilfe von `figure` kann ein neues figure-Fenster erzeugt werden, in welches dann der nächste Plot eingefügt wird. So wird kein bestehender Plot überschrieben. Über `figure(1)`, `figure(2)`, `figure(3)` etc. können immer wieder bereits erzeugte figure-Fenster ausgewählt und ihr Inhalt mit dem nächsten Aufruf von `plot` überschrieben werden.
- Mehrere Graphen in ein Plotfenster:  
Mithilfe von `hold on` kann das aktuelle figure-Fenster festgehalten und nacheinander alle Graphen mit `plot()` reingeplottet werden, ohne dass diese sich überschreiben. Als Alternative können auch direkt in den `plot`-Befehl alle Datenvektoren eingegeben werden:

---

```
plot(x1, y1, x2, y2, x3, y3, ...)
```

MATLAB liefert denselben Plot, wenn die Vektoren  $x_1, x_2$ , usw. spaltenweise in einer Matrix  $X$  und die Vektoren  $y_1, y_2$ , usw. spaltenweise in einer Matrix  $Y$  stehen und `plot(X, Y)` aufgerufen wird.

- Jeden Graphen in jeweils einen Subplot:  
MATLAB bietet die Möglichkeit, innerhalb eines figure-Fensters mehrere Koordinatensysteme zu erzeugen und somit Teilfenster zu erhalten. Der Befehl hierfür lautet `subplot(m, n, p)`. Die erste Zahl  $m$  sagt, dass in vertikaler Richtung  $m$  Grafikteilfenster, die zweite Zahl  $n$ , dass in horizontaler Richtung  $n$  Grafikteilfenster vorgesehen sind. Die Zahl  $p$  gibt an, in welches der  $(m \times n)$ -Teilfenster die nächste Grafik gezeichnet werden soll. Gezählt wird *zeilenweise* von links oben nach rechts unten.

---

### 2.3.2 Der plot-Befehl – 3D

---

Zur Erzeugung von 3D-Plots gibt es u. a. folgende Befehle:

<code>plot3</code>	Analoger Befehl zum (2D-)plot\verb-Befehl
<code>mesh</code>	Gitterlinien- oder Netzplot
<code>surf</code>	(Ober-)flächenplot
<code>contour</code>	Kontur- oder Höhenlinienplot
<code>pcolor</code>	Pixelplot
<code>surfc</code>	Flächen- und Konturplot

Folgender Beispielcode erzeugt eine Spirale:

```
>> t = (-10:.1:10).';  
>> x = sin(t);  
>> y = cos(t);  
>> plot3(x, y, t);
```

Mit dem Befehl `view(az, el)` kann die Ansicht/der Blickwinkel des Plots geändert werden. Hierbei ist  $az$  die horizontale Rotation in Grad und  $el$  entsprechend die vertikale Rotation.

Mit `mesh()` und `surf()` können 3D-Funktionenplots in jeweils verschiedener Darstellungsart (Drahtmodell und Kachelmodell) erstellt werden. Wenn zusätzlich ein „c“ angehängt wird, also `meshc()`, können die Höhenlinien angezeigt werden. Werden nur die Höhenlinien benötigt, reicht der Befehl `contour()`.

Durch die Vorgabe von zwei Vektoren  $x = (x_i)_{i=1}^k$ ,  $y = (y_j)_{j=1}^n$  können mit

```
[X, Y] = meshgrid(x, y)
```

zwei Matrizen  $X$  und  $Y$  erzeugt werden, wobei jede Zeile von  $X$  der Vektor  $x$  ist und jede Spalte von  $Y$  den Vektor  $y$  enthält. Einander entsprechende Elemente dieser beiden Matrizen bilden gerade die beiden Koordinaten der Gitterpunkte.

Als Beispiel wird die Funktion

$$f(x,y) = e^{-x^2-y^2} \cdot \sin(\pi x y)$$

geplottet.

Dazu muss erst das entsprechende Gitter erzeugt werden:

---

```
>> x = linspace(-2, 2, 30);
>> y = linspace(-2, 2, 30);
>> [X, Y] = meshgrid(x, y);
```

Nach Eingabe der Funktion

```
>> Z = exp(-X.^2 - Y.^2) .* sin(pi * X .* Y);
```

kann die Funktion in den verschiedenen Darstellungsarten angezeigt werden, z. B.

```
>> subplot(2, 2, 1);
>> mesh(X, Y, Z); title('mesh');

>> subplot(2, 2, 2);
>> surf(X, Y, Z);
>> view(-26, 20); title('surf');

>> subplot(2, 2, 3);
>> contour(X, Y, Z, 10); title('contour');
```

---

### 2.3.3 Weitere Darstellungen

---

Es gibt noch weitere Darstellungsweisen in MATLAB, sowohl für 2D als auch 3D. Dazu zählen z. B. `bar()` und `pie()`. Mit `bar` kann ein Balkendiagramm erzeugt werden und mit `pie` entsprechend ein Kuchendiagramm. Da die Syntax sich nicht wesentlich von der Beschriebenen unterscheidet, sei an dieser Stelle auf die entsprechenden help-files verwiesen.

---

## 2.4 Debugger

---

Um Fehler bei der Programmierung zu finden und zu lösen, bietet sich die Verwendung eines Debuggers an. Der Debugger kann den Programmablauf unterbrechen und ist in der Lage, Variablen zu verändern. Ein zentrales Element sind die *Breakpoints*. Erreicht der Programmablauf einen *Breakpoint*, so wird er dort unterbrochen. Es ist somit möglich, den Wert der Variablen im Speicher zu betrachten und auch zu ändern. Außerdem kann auch jeder Programmschritt einzeln durchlaufen werden, um die jeweiligen Auswirkungen zu untersuchen.

Ein *Breakpoint* wird in eine Funktion oder ein Skript durch einen Mausklick direkt rechts von der Zeilennummer gesetzt. Es erscheint ein roter Punkt. Durch erneutes Klicken verschwindet er wieder. Erreicht der Programmablauf diese Zeile, so erscheint ein grüner Pfeil rechts vom *Breakpoint*. Der Inhalt dieser Zeile wurde noch nicht ausgeführt. Die Variablen im Workspace können jetzt betrachtet und gegebenenfalls verändert werden.

Um den Programmablauf zu untersuchen, ist es sinnvoll, Schritt für Schritt mit dem Debugger hindurch zu laufen. Dafür stehen die Einträge im Editor-Ribbon unter dem Bereich „Debug“ zur Verfügung. (Dieser wird anstelle des Bereichs „Run“ angezeigt, wenn der Debugger an einem Haltepunkt gestoppt hat.) Durch *Step* (F10) wird die aktuelle Zeile ausgeführt, der grüne Pfeil wandert in die nächste Zeile. Durch *Step in* (F11) wird die erste Funktion in der aktuellen Zeile aufgerufen und dort in der ersten Zeile angehalten. *Step out* (Umschalt-F11) führt die aktuelle Funktion bis zum Ende aus und hält direkt nach dem Aufruf dieser Funktion an. *Continue* (F5) startet den Programmablauf wieder.

---

Sehr nützlich ist auch folgende Option: Im Editor-Ribbon unter *Breakpoints* kann der Punkt *Stop on Errors* ausgewählt werden. Tritt dann beim Durchlauf einer Funktion ein Fehler auf, so wird die Fehlermeldung ausgegeben, der Debugger hält allerdings direkt vor dem Auftreten des Fehlers an. Damit ist es möglich, die Ursache des Fehlers direkt zu finden.





# Versuch 3

## 3.1 Einführung in Differentialgleichungen

Differentialgleichungen (DGL) spielen eine wichtige Rolle bei der Beschreibung physikalischer, technischer Prozesse. Je nach Modellierung treten verschiedene Typen von Differentialgleichungen mit speziellen Eigenschaften auf. Diese Einführung soll dazu dienen, einen groben Überblick über die verschiedenen Typen von Differentialgleichungen zu geben. Allgemein wird zwischen

1. gewöhnlichen und partiellen,
2. linearen und nichtlinearen,
3. expliziten und impliziten bzw.
4. homogenen und inhomogenen

Differentialgleichungen mit konstanten oder variablen Koeffizienten unterschieden (falls die unabhängige Variable der Zeit  $t$  entspricht, wird auch von zeitinvariant bzw. zeitvariant gesprochen).

Der Unterschied zwischen gewöhnlichen und partiellen DGL besteht darin, dass bei gewöhnlichen DGL nur Ableitungen nach einer, bei partiellen DGL dagegen Ableitungen nach mehreren Variablen auftreten. Demnach lautet jeweils die allgemeine Form der gewöhnlichen bzw. partiellen DGL (mit zwei unabhängigen Variablen  $x$  und  $y$ )  $n$ -ter Ordnung in impliziter Form:

$$0 = F(x, y(x), y'(x), y''(x), \dots, y^{(n)}(x)) \quad (3.1)$$

$$0 = F\left(x, y, z(x, y), \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, \frac{\partial^2 z}{\partial x^2}, \frac{\partial^2 z}{\partial y \partial x}, \frac{\partial^2 z}{\partial y^2}, \dots, \frac{\partial^n z}{\partial y^n}\right) \quad (3.2)$$

Eine DGL  $n$ -ter Ordnung heißt linear, wenn sie das Superpositionsprinzip und das Verstärkungsprinzip erfüllt, anderenfalls nichtlinear. Für den Fall  $n = 2$  sind die Gl. (3.1) und Gl. (3.2) genau dann linear, wenn sie sich jeweils in die Form

$$f(x) = a_2(x) \cdot y'' + a_1(x) \cdot y' + a_0(x) \cdot y \quad (3.3)$$

$$f(x, y) = A(x, y) \cdot \frac{\partial^2 z(x, y)}{\partial x^2} + 2B(x, y) \cdot \frac{\partial^2 z(x, y)}{\partial y \partial x} + C(x, y) \cdot \frac{\partial^2 z(x, y)}{\partial y^2} + a(x, y) \cdot \frac{\partial z(x, y)}{\partial x} + b(x, y) \cdot \frac{\partial z(x, y)}{\partial y} + c(x, y) \cdot z(x, y) \quad (3.4)$$

überführen lassen. Sind  $f(x)$  bzw.  $f(x, y)$  gleich Null, dann sind die DGL homogen, anderenfalls inhomogen.

In diesem Praktikumsversuch werden ausschließlich gewöhnliche, lineare DGL mit konstanten Koeffizienten betrachtet, da diese in vielen Fällen physikalische, technische Prozesse ausreichend genau beschreiben. Die Behandlung partieller DGL wird hier nicht behandelt.

Die Gesamtheit aller Lösungen der gewöhnlichen, linearen DGL  $n$ -ter Ordnung mit konstanten Koeffizienten

$$a_n y^n(x) + a_{n-1} y^{n-1}(x) + \dots + a_0 y(x) = f(x)$$

---

wird die allgemeine Lösung genannt. Eine spezielle Lösung ist gegeben, falls neben der DGL außerdem die  $n$  Anfangsbedingungen der Form

$$\begin{aligned}y(x_0) &= y_0 \\y'(x_0) &= y_1 \\&\vdots \\y^{n-1}(x_0) &= y_{n-1}\end{aligned}$$

erfüllt sind.

Spezielle Lösungen können entweder analytisch oder numerisch bestimmt werden, wobei in diesem Praktikumsversuch die Berechnung der numerischen Lösung in MATLAB im Vordergrund steht. Auf die analytische Lösung mithilfe der *Symbolic Math Toolbox* wird lediglich kurz im folgenden Abschnitt eingegangen.

---

## 3.2 Lösen von Differentialgleichungen

---

---

### 3.2.1 Analytische Lösung

---

Mithilfe der *Symbolic Math Toolbox* können DGLs *analytisch* mit dem Befehl `dsolve` gelöst werden. Voreingestellt als unabhängige Variable ist  $t$ .

Als Beispiel wird die Differentialgleichung

$$\dot{y}(t) + 0,2y(t) = 1$$

betrachtet. Diese wird wie folgt eingegeben (Die erste Ableitung  $\dot{y}$  wird dabei mit `Dy` eingegeben):

```
>> dsolve('Dy + 1/5*y = 1')
ans =
C1*exp(-t/5) + 5
```

MATLAB bezeichnet die Integrationskonstante mit `C1`. Zusätzlich können auch Anfangsbedingungen angegeben werden:

```
>> dsolve('Dy + 1/5*y = 1', 'y(0) = 1')
ans =
5 - 4*exp(-t/5)
```

Bei Differentialgleichungen zweiter Ordnung wird die zweite Ableitung  $\ddot{y}$  mit `D2y` eingegeben:

```
>> dsolve('D2y + 4*Dy + 13*y = 120*cos(3*t)', 'Dy(0) = 0', 'y(0) = 0')
ans =
[...]
```

---

### 3.2.2 Approximation über Differenzengleichung

---

Eine Möglichkeit der numerischen, näherungsweisen Lösung von Differentialgleichungen ist es, die Differentiale durch Differenzen anzunähern.

---

So setzt man für hinreichend kleine Zeitschritte  $\Delta t = t_k - t_{k-1}$

$$\dot{x} = \frac{dx}{dt} \approx \frac{\Delta x}{\Delta t} = \frac{x_k - x_{k-1}}{t_k - t_{k-1}}.$$

Für die Differentialgleichung zweiter Ordnung

$$a_2 \ddot{x}(t) + a_1 \dot{x}(t) = a_0 u(t)$$

ergibt sich mit den Approximationen

$$\begin{aligned}\dot{x}|_k &\approx \frac{x_k - x_{k-1}}{\Delta t} \\ \ddot{x}|_k &\approx \frac{\dot{x}|_k - \dot{x}|_{k-1}}{\Delta t} \approx \frac{x_k - 2x_{k-1} + x_{k-2}}{\Delta t^2}\end{aligned}$$

die Differenzengleichung

$$\frac{a_2 + a_1 \Delta t}{\Delta t^2} \cdot x_k - \frac{2a_2 + a_1 \Delta t}{\Delta t^2} \cdot x_{k-1} + \frac{a_2}{\Delta t^2} \cdot x_{k-2} = a_0 u_k$$

bzw.

$$x_k = \frac{2a_2 + a_1 \Delta t}{a_2 + a_1 \Delta t} \cdot x_{k-1} - \frac{a_2}{a_2 + a_1 \Delta t} \cdot x_{k-2} + \frac{a_0 \Delta t^2}{a_2 + a_1 \Delta t} \cdot u_k.$$

Diese Differenzengleichung kann dann ausgehend von zwei Anfangswerten (z. B.  $x_{-2}$  und  $x_{-1}$ ) rekursiv gelöst werden.

---

### 3.2.3 Numerische Integration einer DGL

---

In diesem Versuch werden numerische Verfahren zur Lösung gewöhnlicher Differentialgleichungen erster Ordnung behandelt. Numerische Verfahren zur Lösung von gewöhnlichen Differentialgleichungen beliebiger Ordnung finden stets nur Lösungen für Differenzengleichungen erster Ordnung. Jedes numerische Verfahren besteht daher aus zwei Schritten:

1. Überführung der DGL  $n$ -ter Ordnung in ein System von  $n$  DGLs erster Ordnung (*Anmerkung:* In der Regelungstechnik entspricht dies dem Übergang in den Zustandsraum (siehe SDRT II, [1])),
2. Simultane Lösung der aus den DGLs erster Ordnung abgeleiteten Differenzengleichungen.

Die Numerik von gewöhnlichen Differentialgleichungen erster Ordnung bildet daher die Grundlage für numerische Verfahren zur Lösung von gewöhnlichen Differentialgleichungen beliebiger Ordnung. Numerische Verfahren zur Lösung von gewöhnlichen Differentialgleichungen können in jeder Programmiersprache realisiert werden. Ausgangspunkt ist immer eine Differentialgleichung in der Form

$$\dot{x} = f(t, x)$$

die im Intervall von  $t_1$  bis  $t_2$  zu integrieren ist.

---

## Unterteilung der numerischen Verfahren

---

Numerische Verfahren zur Integration von Differentialgleichungen können zum einen in explizite und implizite Verfahren, zum anderen in Ein- und Mehrschrittverfahren unterschieden werden.

Bei expliziten Verfahren wird der neue Wert  $x_k$  direkt aus dem zurückliegenden Wert (Einschrittverfahren) bzw. mehreren zurückliegenden Werten (Mehrschrittverfahren) berechnet. Formelmäßig wird ein explizites Einschrittverfahren durch

$$x_k = x_{k-1} + \Delta t_{k-1} \cdot \varphi(t_{k-1}, x_{k-1})$$

und ein explizites Mehrschrittverfahren durch

$$x_k = x_{k-1} + \Delta t_{k-1} \cdot \varphi(t_{k-1}, x_{k-1}, x_{k-2}, \dots)$$

ausgedrückt. Dabei ist  $\Delta t_{k-1} = t_k - t_{k-1}$  die Schrittweite und  $\varphi$  eine durch das Verfahren festgelegte Inkrementfunktion.

Für das einfache explizite Eulerverfahren

$$x_k = x_{k-1} + \Delta t_{k-1} \cdot f(t_{k-1}, x_{k-1})$$

gilt also

$$\varphi(t_k, x_k) = f(t_k, x_k).$$

Die Inkrementfunktion kann jedoch auch komplizierter sein. Für das Runge-Kutta-Verfahren vierter Ordnung lautet diese beispielsweise

$$\begin{aligned}\varphi(t_k, x_k) &= \frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4 \\ k_1 &= f(t_k, x_k) \\ k_2 &= f\left(t_k + \frac{\Delta t_k}{2}, x_k + \frac{\Delta t_k}{2} \cdot k_1\right) \\ k_3 &= f\left(t_k + \frac{\Delta t_k}{2}, x_k + \frac{\Delta t_k}{2} \cdot k_2\right) \\ k_4 &= f(t_k + \Delta t_k, x_k + \Delta t_k \cdot k_3).\end{aligned}$$

Die rechte Seite  $f(t, x)$  der DGL wird hierbei häufiger ausgewertet, um die Ordnung (d. h. die Genauigkeit) des Verfahrens zu erhöhen.

Bei Mehrschrittverfahren erfolgt die Erhöhung der Ordnung dagegen durch die Berücksichtigung weiter zurückliegender Punkte. So lautet beispielsweise das Adams-Bashforth-Verfahren dritter Ordnung

$$x_k = x_{k-1} + \Delta t \cdot \left( \frac{23}{12}f(t_{k-1}, x_{k-1}) - \frac{16}{12}f(t_{k-2}, x_{k-2}) + \frac{5}{12}f(t_{k-3}, x_{k-3}) \right),$$

wobei hier der Einfachheit halber eine feste Rechenschrittweite  $\Delta t$  angenommen ist. Es muss dabei für jeden neuen Schritt nur einmal die rechte Seite der DGL neu ausgewertet werden, da  $f(t_{k-2}, x_{k-2})$  und  $f(t_{k-3}, x_{k-3})$  schon in den beiden vorangegangenen Schritten berechnet wurden.

Wesentliches Merkmal der expliziten Verfahren ist, dass ausgehend von  $x_{k-1}$  (bzw.  $x_{k-1}, x_{k-2}, \dots$ ) der neue Wert  $x_k$  über eine feste Anzahl an Rechenschritten ausgerechnet werden kann.

---

Implizite Verfahren dagegen sind von der Form

$$x_k = x_{k-1} + \Delta t_{k-1} \cdot \varphi(t_{k-1}, x_k, x_{k-1})$$

(Einschrittverfahren) bzw.

$$x_k = x_{k-1} + \Delta t_{k-1} \cdot \varphi(t_k, x_k, x_{k-1}, x_{k-2}, \dots)$$

(Mehrschrittverfahren). Hierbei taucht  $x_k$  auch auf der rechten Seite in der Inkrementfunktion auf, so dass eine einfache Umformung nach  $x_k$  im Allgemeinen nicht möglich ist. Die Gleichung muss daher in jedem Schritt numerisch (z. B. mit dem Newton-Verfahren) gelöst werden.

Ein Beispiel für ein einfaches implizites Einschrittverfahren ist das implizite Eulerverfahren

$$x_k = x_{k-1} + \Delta t_{k-1} \cdot f(t_k, x_k).$$

---

### Stabilität numerischer Verfahren

---

Ein wesentlicher Aspekt bei der numerischen Lösung von Differentialgleichungen ist die Frage nach der Stabilität der *Simulation*. Diese ist nicht gleichbedeutend mit der Stabilität des Systems, welches simuliert wird. So kann es sein, dass ein stabiles System instabil simuliert wird! D. h. dass die berechnete Lösung aufschwingt oder wegläuft, obwohl das simulierte System stabil ist.

Die Stabilität der Simulation hängt, neben den Eigenschaften des simulierten Systems, von dem gewählten Verfahren und der gewählten Rechenschrittweite ab. Dabei sind die prinzipiellen Abhängigkeiten wie folgt: Eine Verkleinerung der Rechenschrittweite wirkt stabilisierend. Implizite Verfahren besitzen bessere Stabilitätseigenschaften als explizite Verfahren, Einschrittverfahren sind aus Sicht der Stabilität günstiger als Mehrschrittverfahren.

---

### Numerische Integration in MATLAB

---

Gewöhnliche Differentialgleichungen können in MATLAB mit verschiedenen, bereits implementierten Verfahren gelöst werden. Die Lösungsverfahren heißen alle ode, gefolgt von einer Ziffernkombination. Darin steht ode für Ordinary Differential Equation (= gewöhnliche Differentialgleichung). Die Syntax zum Aufruf einer der Funktionen ist:

$$[t, x] = \text{solver}(\text{odefun}, \text{tspan}, y_0, \text{options})$$

wobei solver der Name des Verfahrens ist, tspan ein Vektor, der Anfang  $t_1$  und Ende  $t_2$  des Integrationsintervalls angibt und  $y_0$  der Anfangswert. Die von MATLAB zur Verfügung gestellten Verfahren sind in Tabelle 3.1 aufgelistet. Die einzelnen Verfahren unterscheiden sich in Hinblick auf ihre Genauigkeit und die benötigte Rechenzeit. Außerdem ist nicht jedes Lösungsverfahren für jede Art von DGL gleich gut geeignet. Dadurch muss je nach Problemstellung ein eigenes, evtl. angepasstes Verfahren gefunden werden.

Die ersten drei Verfahren (*explizite* Verfahren) in der Tabelle sind für *nicht-steife*, letztere (*implizite* Verfahren) für *steife* Differentialgleichungen geeignet. Eine Differentialgleichung wird als steif bezeichnet, wenn die Eigenwerte des beschriebenen dynamischen Systems sehr weit auseinanderliegen.

Diese implementierten Verfahren besitzen alle eine automatische Schrittweitensteuerung. Damit muss keine Schrittweite vorgegeben werden, sondern diese wird anhand von gegebenen Genauigkeitsmaßen automatisch bestimmt und im Laufe der Simulation auch stetig angepasst. Damit wird gegenüber

**Tabelle 3.1.: Übersicht der verschiedenen Solver-Typen**

Solver	Beschreibung	Ordnung	Verfahren
ode45	Runge-Kutta-Verfahren	4 und 5	explizit
ode23	Runge-Kutta-Verfahren	2 und 3	
ode113	Adams-Bashford-Moulton-Verfahren	1 bis 13	
ode15s	Gear's Verfahren (rückwärtige Differentiation)	1 bis 5	implizit
ode23s	modifiziertes Rosenbrock-Verfahren	2 und 3	
ode23tb	implizites Runge-Kutta-Verfahren	2 und 3	

von Verfahren mit fester Schrittweite bei vergleichbarer Genauigkeit eine (z. T. wesentlich) schnellere Simulation erreicht. Auch hilft die Schrittweitensteuerung dabei, das System stabil zu simulieren.

Allerdings kann auch die automatische Schrittweitensteuerung eine stabile Simulation nicht garantieren! Sollte vermutet werden, dass ein instabiles Verhalten des simulierten Systems (Dauerschwingen, Aufschwingen, Weglaufen) durch die Simulation und nicht das System selber zu erklären ist, dann kann testweise die geforderte Genauigkeit erhöht (und damit eine geringere Schrittweite erzwungen) werden oder es können andere (implizite) Solver verwendet werden.

Für weitere Details zur numerischen Lösung von gewöhnlichen DGLs sei auf die entsprechende Literatur ([4]) verwiesen.

Mithilfe von `options` können weitere Parameter übergeben werden, die z. B. die Genauigkeit und die Art der Ausgabe der Lösung bestimmen. Die Übergabe wird über `options` gesteuert und mit der Anweisung `odeset` gesetzt. Einige der wichtigen Optionen sind:

- `RelTol`: Gewünschte relative Genauigkeit, Voreinstellung  $1e-3$  (0,1 Prozent)
- `AbsTol`: Gewünschte absolute Genauigkeit, Voreinstellung  $1e-6$
- `Refine`: Feinheit der Ausgabe, steuert die Anzahl der Punkte, die ausgegeben wird. Voreinstellung: 4 für `ode45`, 1 für alle anderen
- `Stats`: Ausgabe des numerischen Aufwands eines Verfahrens, `on` oder `off` (Voreinstellung)
- `Vectorized`: Angabe, ob das m-File der rechten Seite der DGL Vektoreingaben korrekt verarbeiten kann, `on` oder `off` (Voreinstellung)
- `MaxStep`: Maximal erlaubte Schrittweite, Voreinstellung:  $1/10$  der Rechenintervallbreite
- `InitialStep`: Vorschlag für Anfangsschrittweite, Voreinstellung: Automatische Auswahl
- `NormControl`:
  - `on`: Die Genauigkeit der Lösung wird über die 2-Norm gemessen
  - `off`: Die Genauigkeit wird komponentenweise gemessen (Voreinstellung)

Eine beispielhafte Einstellung wäre

```
options = odeset('Stats', 'on', 'AbsTol', 1e-10);
```

## Explizite Verfahren

Explizite Verfahren werden hauptsächlich für nicht-steife Differentialgleichungen verwendet. Die wichtigsten expliziten MATLAB Verfahren sind `ode45` und `ode23`. `ode45` ist eine Kombination aus einem Runge-Kutta-Verfahren vierter Ordnung mit einem fünfter Ordnung, entsprechendes gilt für `ode23`. Die Kombination von zwei Ordnungen erlaubt die Überprüfung von Genauigkeit durch einen direkten Vergleich der beiden. `ode45` ist sowohl in der Schnelligkeit als auch in der Genauigkeit als gut

---

einzustufen. Mit ode23 wird zwar schneller ein Ergebnis erreicht, jedoch mit geringerer Genauigkeit. Dadurch kann es bei einigen DGL sinnvoll zur schnellen Abschätzung genutzt werden.

ode113 ist im Gegensatz zu den beiden anderen kein Einschritt- sondern ein Mehrschrittverfahren. Da explizite Mehrschrittverfahren die rechte Seite  $f(t, x)$  der DGL unabhängig von der Verfahrensordnung nur einmal auswerten, sind diese besonders für Systeme geeignet, bei denen die rechte Seite der DGL sehr aufwendig auszuwerten ist. Nachteilig bei Mehrschrittverfahren sind die schlechteren Stabilitätseigenschaften.

### Implizite Verfahren

Implizite Verfahren werden hauptsächlich zum Lösen steifer Differentialgleichungen verwendet. MATLAB stellt mit ode15s, ode23tb und ode23s drei implizite Verfahren zur Verfügung. Hierbei handelt es sich bei ode23tb und ode23s um Einschrittverfahren, während ode15s ein Mehrschrittverfahren darstellt.

### Beispiele

Als einführendes Beispiel wird die Differentialgleichung

$$\dot{x}(t) = x(t), \quad x(t_0) = x_0$$

mit ode23 gelöst. ode benötigt eine Funktion mit den Parametern  $t$  und  $x$ , die die DGL erzeugt. Diese ist in Listing 3.2 gezeigt.

**Listing 3.2:** Funktion expwachs.

---

```
function dx = expwachs(t, x)
    dx = x;
end
```

---

Zusätzlich muss das Lösungsintervall und der Anfangswert übergeben werden. Um die Differentialgleichung zu lösen, muss somit folgender Befehl ausgeführt werden:

```
>> [t1, x1] = ode23('expwachs', [0, 5], 0.1);
>> plot(t1, x1)
```

Entsprechend geht man bei Differentialgleichungen höherer Ordnung vor, nur muss zuerst das System in zwei Differentialgleichungen erster Ordnung umgeschrieben werden:

$$\ddot{x}(t) + d\dot{x}(t) + kx(t) = 0 \tag{3.5}$$

Daraus folgt mit  $x_1 = x(t)$  und  $x_2 = \dot{x}(t)$

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -dx_2 - kx_1 \end{aligned}$$

bzw. in vektorieller Form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & -d \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Die sich damit ergebende Definition der Funktion ist in Listing 3.3 gegeben.

---

**Listing 3.3:** Funktion lin.

---

```
function dx = lin(t, x)
2     d = 0.1;
        k = 1;
        dx = [0, 1; -k, -d] * x;
end
```

---

Zum Lösen von Gl. (3.5) mit den Anfangsbedingungen  $x(0) = 1$  und  $\dot{x}(0) = 0$  im Zeitintervall  $[0, 50]$  lautet der Befehl:

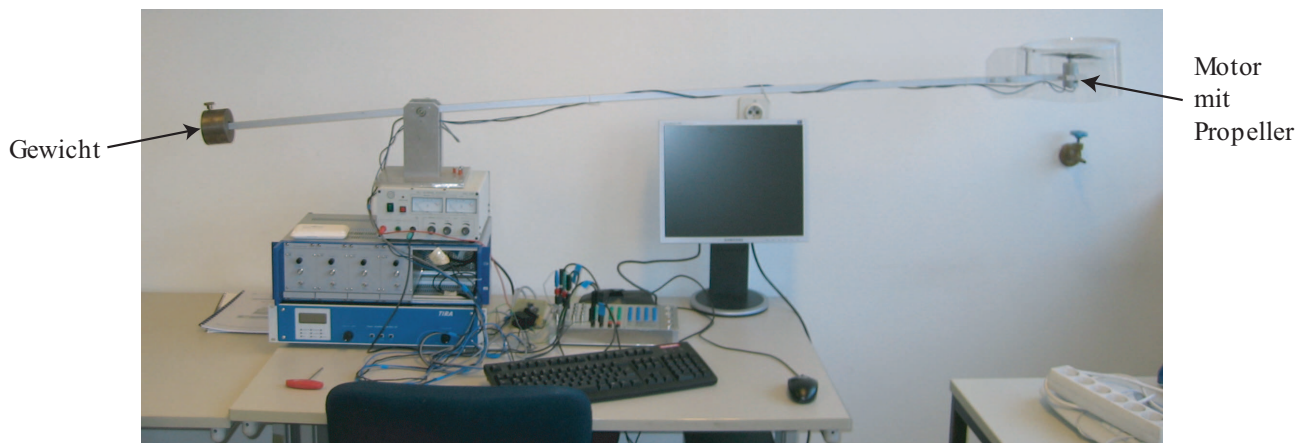
```
>> [t, x] = ode23('lin', [0, 50], [1; 0]);
>> plot(t, x)
```



# A Einführung Versuch 4 – 6

## A.1 Der Pendelschrauber

In den folgenden Versuchen wird als Regelstrecke ein Hubschraubermodell mit einem Freiheitsgrad (Pendelschrauber) betrachtet. Dieses Hubschraubermodell wurde am IAT als Versuchsaufbau realisiert (Abbildung A.1). Der Pendelschrauber besteht aus einem Hebel, einem Motor mit Propeller und



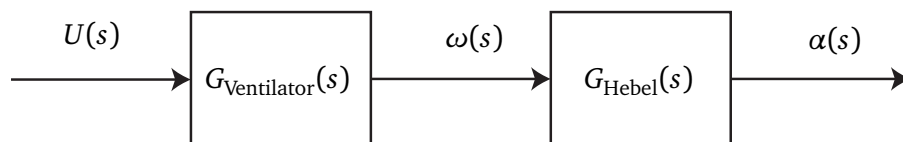
**Abbildung A.1.:** Versuchsstand Pendelschrauber

einem Gegengewicht. Der Hebel ist nach einem Viertel seiner Länge drehbar gelagert. Auf der einen Seite des Hebels ist der Elektromotor mit dem Propeller angebracht, auf der anderen Seite befindet sich das Gegengewicht. Die Motorspannung ist variabel und der Winkel des Hebels wird über einen Sensor gemessen. Als Eingangsgröße des Systems wird im Folgenden daher die Eingangsspannung des Motors  $U$  verwendet, als Ausgangsgröße der Winkel des Hebelarms  $\alpha$ .

Für die weitere Betrachtung wird das System in zwei Teilsysteme unterteilt (Abbildung A.2):

- Ventilator (Motor mit Propeller)
- Hebel

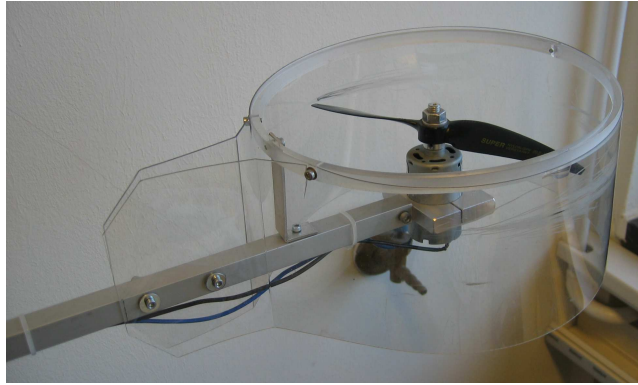
Als Zwischengröße wird die Drehzahl des Motors bzw. des Propellers  $\omega$  verwendet.



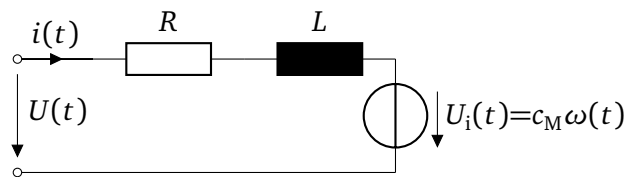
**Abbildung A.2.:** Teilsysteme Pendelschrauber

### A.1.1 Das Teilsystem Ventilator

Der Ventilator besteht aus einem Gleichstrommotor mit Propeller (Abbildung A.3). Durch die Rotation des Propellers wird eine Schubkraft  $S$  erzeugt.



**Abbildung A.3.: Ventilator**



**Abbildung A.4.: Ersatzschaltbild des Gleichstrommotors**

Der Gleichstrommotor lässt sich durch ein Ersatzschaltbild darstellen (Abbildung A.4). Es gilt

$$U(t) = Ri(t) + L \frac{di(t)}{dt} + U_i(t) \quad (\text{A.1})$$

$$U_i(t) = c_M \omega(t) \quad (\text{A.2})$$

Der Drallsatz für den Motor mit Propeller lautet mit dem Trägheitsmoment  $J_P$  des Propellers

$$J_P \dot{\omega}(t) = M_E(t) - M_L(t) \quad (\text{A.3})$$

Das Motormoment  $M_E(t)$  ergibt sich aus

$$M_E(t) = c_M i(t) \quad (\text{A.4})$$

Das Lastmoment  $M_L(t)$  durch den Propeller kann näherungsweise durch einen Faktor  $\beta$  in Abhängigkeit des Quadrates der Winkelgeschwindigkeit  $\omega$  beschrieben werden,

$$M_L(t) = \beta \omega^2(t). \quad (\text{A.5})$$

$\beta$  wurde dabei aus Messdaten identifiziert.

Zusammengefasst ergeben sich die Differentialgleichungen des Ventilators

$$U(t) = Ri(t) + L \frac{di(t)}{dt} + c_M \omega(t) \quad (\text{A.6})$$

$$J_P \dot{\omega}(t) = c_M i(t) - \beta \omega^2(t) \quad (\text{A.7})$$

Innerhalb eines kleinen Bereichs um einen Arbeitspunkt können die Gleichungen linearisiert werden. Es gilt dann für das linearisierte System

$$\Delta U(t) = R \Delta i(t) + L \frac{d\Delta i(t)}{dt} + c_M \Delta \omega(t) \quad (\text{A.8})$$

$$J_P \Delta \dot{\omega}(t) = c_M \Delta i(t) - 2\beta \omega_s \Delta \omega(t) \quad (\text{A.9})$$



Für das Reibmoment wird mit dem Reibkoeffizient  $c_W$  vereinfachend

$$M_R(t) = c_W \dot{\alpha}(t) \quad (\text{A.13})$$

angenommen.

Die Schubkraft  $S(t)$  kann mit dem Faktor  $\gamma$  in Abhängigkeit der Propellerdrehzahl  $\omega$  durch

$$S(t) = \gamma \omega^2(t) \quad (\text{A.14})$$

angenähert werden. Der Parameter  $\gamma$  wurde aus Messdaten identifiziert. Daraus ergibt sich mit dem Hebelarm  $H_R$  das Schubmoment

$$M_S(t) = H_R S(t) = H_R \gamma \omega^2(t). \quad (\text{A.15})$$

Das Schubmoment ist unabhängig vom Winkel des Hebels. Die Schubkraft wirkt wegen der festen Einspannung des Ventilators immer senkrecht zur Stange. Die Gewichtsmomente der Komponenten ergeben sich aus den Gewichtskräften und den jeweiligen Hebelarmen der einzelnen Komponenten. Die Länge der Hebelarme ist hier vom Winkel des Hebels abhängig:

$$M_G(t) = k \cdot \cos \alpha(t). \quad (\text{A.16})$$

Die Konstante  $k$  fasst Massen und Hebelarme der einzelnen Komponenten zusammen,

$$k = -\frac{1}{2} \cdot m_{SL} \cdot g \cdot S_L - m_L \cdot g \cdot H_L + \frac{1}{2} \cdot m_{SR} \cdot g \cdot S_R + m_P \cdot g \cdot H_R, \quad (\text{A.17})$$

wobei  $m_{SL}$  und  $m_{SR}$  die Masse der Teile des Stabes (ohne Motor/Propeller und Gegengewicht) sind, die sich links bzw. rechts des Drehpunktes befinden.

Die nichtlineare Differentialgleichung des Hebels lautet damit

$$J_W \ddot{\alpha}(t) = H_R \gamma \omega^2(t) - c_W \dot{\alpha}(t) - k \cdot \cos \alpha(t). \quad (\text{A.18})$$

Diese Differentialgleichung lässt sich analog zu den Gleichungen des Ventilators in ein Blockschaltbild umwandeln (Abbildung A.7a).

Auch dieses Teilsystem wird um einen Arbeitspunkt linearisiert. Es ergibt sich

$$J_W \Delta \ddot{\alpha}(t) = 2H_R \gamma \omega_s \Delta \omega(t) - c_W \Delta \dot{\alpha}(t) + k \cdot \sin \alpha_s \Delta \alpha(t). \quad (\text{A.19})$$

Abbildung A.7b zeigt das Blockschaltbild des linearisierten Systems. Zusätzlich lässt sich eine lineare Übertragungsfunktion für das System angeben

$$G_{\text{Hebel}}(s) = \frac{\Delta \alpha(s)}{\Delta \omega(s)} = \frac{2\gamma \omega_s H_R}{J_W s^2 + c_W s + (-k \sin \alpha_s)}. \quad (\text{A.20})$$

---

### A.1.3 Übertragungsfunktion des (linearisierten) Gesamtsystems

---

Mit den linearisierten Übertragungsfunktionen der Teilsysteme,

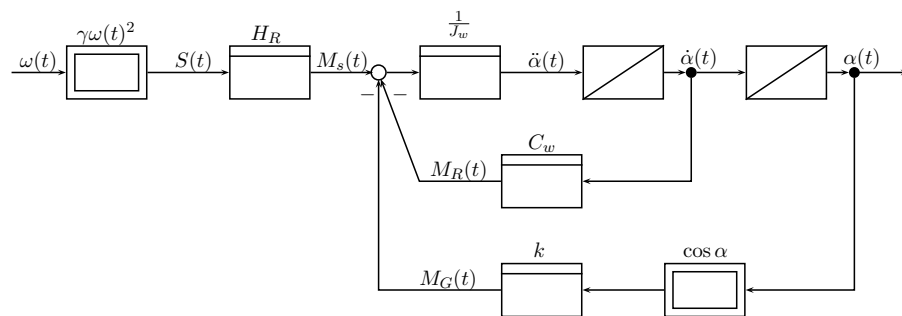
$$G_{\text{Ventilator}}(s) = \frac{\Delta \omega(s)}{\Delta U(s)} = \frac{c_M}{J_P L \cdot s^2 + (KL + RJ_P) \cdot s + (RK + c_M^2)}, \quad (\text{A.21})$$

$$G_{\text{Hebel}}(s) = \frac{\Delta \alpha(s)}{\Delta \omega(s)} = \frac{2\gamma \omega_s H_R}{J_W s^2 + c_W s + (-k \sin \alpha_s)}, \quad (\text{A.22})$$

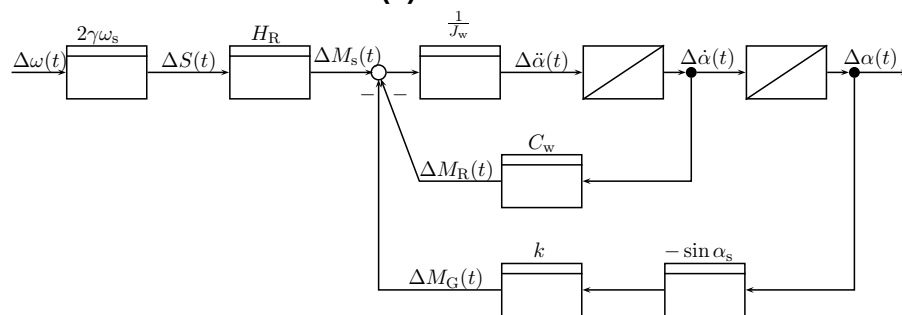
ergibt sich

$$\begin{aligned} G(s) &= G_{\text{Ventilator}}(s) \cdot G_{\text{Hebel}}(s) \\ &= \frac{2c_M \gamma \omega_s H_R}{(J_P L \cdot s^2 + (KL + RJ_P) \cdot s + (RK + c_M^2)) \cdot (J_W s^2 + c_W s + (-k \sin \alpha_s))}. \end{aligned} \quad (\text{A.23})$$

als Übertragungsfunktion des linearisierten Gesamtsystems.



**(a) nichtlinear**



**(b) linearisiert**

**Abbildung A.7.: Blockschaltbilder Wippe**

## A.2 Formelzeichen und Zahlenwerte

**Tabelle A.1.: Parameter Pendelschrauber**

Parameter	Bedeutung	Wert	Einheit
<b>Ventilator</b>			
$c_M$	Motorkonstante	$6,2133 \cdot 10^{-3}$	$\frac{\text{Nm}}{\text{A}}$
$J_P$	Massenträgheitsmoment des Propellers	$1,6833 \cdot 10^{-4}$	$\text{kgm}^2$
$L$	Induktivität des Elektromotors	$5,34 \cdot 10^{-6}$	H
$R$	Innenwiderstand des Elektromotors	0,1655	$\Omega$
$\beta$	Proportionalitätsfaktor des Lastmoments	$\frac{1}{12500000}$	$\text{Ns}^2$
$\omega_s$	Arbeitspunkt der Winkelgeschwindigkeit	320,8745	$\frac{1}{\text{s}}$
$K$	Linearisierung des Lastmomentes	$\Rightarrow (\text{A.10})$	Ns
<b>Pendel</b>			
$c_W$	Reibkoeffizient Wippe	0,75	Nms
$g$	Erdbeschleunigung	9,81	$\frac{\text{m}}{\text{s}^2}$
$H_L$	Hebellänge (links)	0,48	m
$H_R$	Hebellänge (rechts)	1,52	m
$J_W$	Massenträgheitsmoment Wippe	1,9141	$\text{kgm}^2$
$m_L$	Masse Gewicht (links)	2,01134	kg
$m_P$	Masse Propeller	0,48	kg
$m_{SL}$	Masse Stangenabschnitt (links)	0,1404	kg
$m_{SR}$	Masse Stangenabschnitt (rechts)	0,44	kg
$S_L$	Länge der Stange (links)	0,5	m
$S_R$	Länge der Stange (rechts)	1,5	m
$\alpha_s$	Arbeitspunkt des Ausgangswinkels	0	
$\gamma$	Proportionalitätsfaktor des Schubs	$3,7019 \cdot 10^{-6}$	$\text{Ns}^2$
$k$	Gesamtmoment der Gewichtskräfte bei $\alpha = 0$	$\Rightarrow (\text{A.17})$	Nm

### A.3 Wichtige Befehle

Tabelle A.2 gibt eine Zusammenfassung und kurze Erklärung der wichtigsten in den nächsten Versuchen benutzten Befehle. Für eine genauere Erklärung, insbesondere bzgl. der genauen Benutzung der Befehle, sollte man die MATLAB-Hilfe zum jeweiligen Befehl über `help Befehl` im „Command Window“ aufrufen.

**Tabelle A.2.: Tabelle wichtiger Befehle**

Befehl	Erklärung
<code>bode</code>	Zeichnet das Bode-Diagramm (Amplituden- und Phasenverlauf) eines Systems
<code>dcgain</code>	Berechnet die stationäre Verstärkung
<code>feedback</code>	Berechnet die Übertragungsfunktion des geschlossenen Regelkreises
<code>impulse</code>	Zeichnet die Reaktion des Systems auf einen Impuls
<code>lsim</code>	Zeichnet die Reaktion eines Systems auf ein beliebiges vorgegebenes Signal
<code>margin</code>	Ähnlich dem Bode Befehl, mit zusätzlichen Möglichkeiten
<code>nyquist</code>	Zeichnet die Ortskurve eines Systems
<code>pole</code>	Berechnet die Polstellen eines Systems
<code>pzmap</code>	Zeichnet die Lage der Nullstellen und Pole in die komplexe Ebene ein
<code>solve</code>	Löst Gleichungen in MATLAB und gibt die Lösung als Vektor aus
<code>step</code>	Zeichnet die Reaktion des Systems auf einen Sprung
<code>tf</code>	Erstellt anhand eines Zähler- und eines Nennerpolynoms eine Übertragungsfunktion
<code>zero</code>	Gibt die Nullstellen eines Systems aus
<code>zpk</code>	Übertragungsfunktion in der Form, dass Nullstellen, Pole und die Verstärkung ablesbar sind Hinweis: Es wird das System in folgender Form beschrieben: $G(s) = k \cdot \frac{(s-z_1)(s-z_2)\dots}{(s-p_1)(s-p_2)\dots}$ Die Verstärkung $k$ entspricht <i>nicht</i> der stationären Verstärkung $G(0)$ !





# Versuch 4

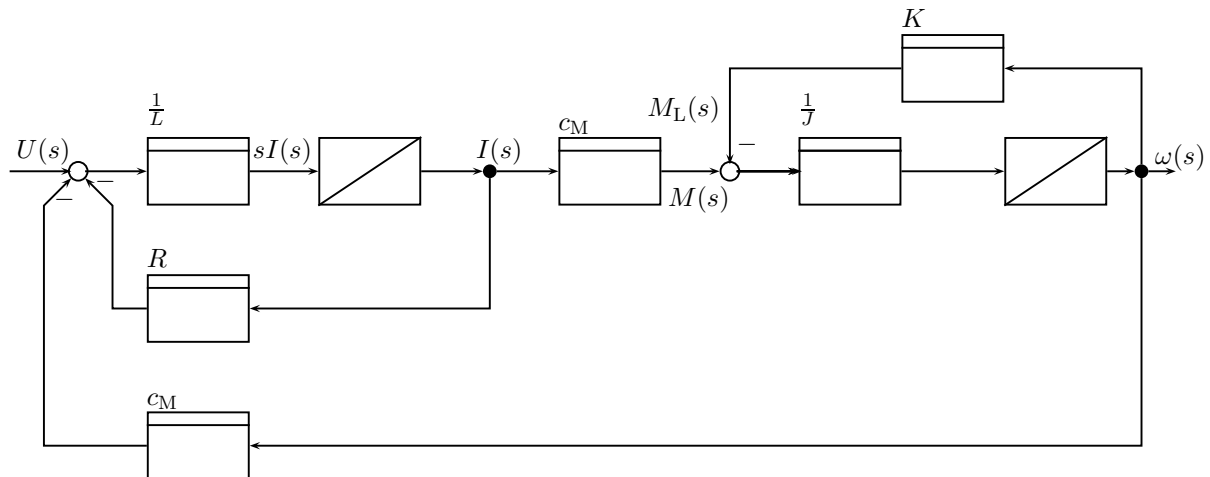
Für die folgenden drei Versuche wird die Einführung in Anhang A benötigt. Diese sollte vor dem vierten Versuch als Vorbereitung gelesen werden.

## 4.1 Die Regelstrecke

In diesem Versuch soll als Regelstrecke ein vereinfachtes Modell eines Ventilators dienen (siehe Einleitung). Im Prinzip handelt es sich hierbei um einen Motor, auf den ein drehzahlproportionales Lastmoment wirkt.

### 4.1.1 Blockschaltbild und Übertragungsfunktion der Regelstrecke

Der Ventilator kann durch ein Blockschaltbild dargestellt werden (Abbildung 4.1).



**Abbildung 4.1.:** Blockschaltbild des Ventilators (linearisiert)

Wenn man das Blockschaltbild mittels bekannter Regeln vereinfacht, ergibt sich schließlich für den Ventilator eine Übertragungsfunktion der Form (Herleitung und Parameterwerte siehe Einführung A.1.1 und A.2).

$$G_{\text{Ventilator}}(s) = \frac{c_M}{LJ_p \cdot s^2 + (KL + RJ_p) \cdot s + (KR + c_M^2)} \quad (4.1)$$

$$= \frac{0,006213}{8,989 \cdot 10^{-10}s^2 + 2,786 \cdot 10^{-5}s + 4,71 \cdot 10^{-5}} \quad (4.2)$$

Dies ist die Übertragungsfunktion eines  $PT_2$ -Gliedes.

### 4.1.2 Eigenschaften und Parameter von $PT_2$ -Gliedern

Allgemein wird die Übertragungsfunktion eines schwingungsfähigen  $PT_2$ -Gliedes durch

$$G^*(s) = \frac{K}{T^2s^2 + 2dT s + 1} \quad \text{mit} \quad T = \frac{1}{\omega_0} \quad (4.3)$$

beschrieben.

Die wichtigsten Eigenschaften dieses Übertragungsgliedes lassen sich aus folgenden Kennwerten ablesen:

- Die *stationäre Verstärkung*  $K$  gibt den Wert an, um den ein konstantes Eingangssignal für  $t \rightarrow \infty$  verstärkt wird.
- Die *Kennkreisfrequenz*  $\omega_0 = \frac{1}{T}$  ist die *Eigenkreisfrequenz des als dämpfungslos gedachten Systems*.
- Der *Dämpfungsgrad*  $d$  ist eine Maßzahl für das Abklingen der Schwingung. Abbildung 4.3 zeigt die Sprungantworten eines Systems zweiter Ordnung mit verschiedenen Dämpfungsgraden. Abbildung 4.4 zeigt das Bode-Diagramm eines  $PT_2$ -Gliedes bei einer Verstärkung von  $K = 1$  und verschiedenen Dämpfungsgraden  $d$ .
- Die *Eigenkreisfrequenz der gedämpften Schwingung*  $\omega_e$  berechnet sich über

$$\omega_e = \omega_0 \cdot \sqrt{1 - d^2}. \quad (4.4)$$

- Die Amplitudenkennlinie hat ihr Maximum bei der *Resonanzfrequenz*  $\omega_r$ ,

$$\omega_r = \omega_0 \cdot \sqrt{1 - 2 \cdot d^2}, \quad \text{für } 0 < d < \frac{1}{\sqrt{2}}. \quad (4.5)$$

Bei technischen Konstruktionen wird eine Anregung mit dieser Frequenz vermieden, d. h. die Erregerfrequenz sollte sich nicht in der Nähe der Resonanzfrequenz befinden.

### Beispiel

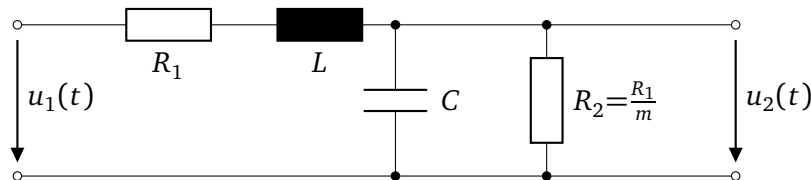


Abbildung 4.2.: RLC-Netzwerk

Gegeben sei das auf Abbildung 4.2 dargestellte RLC-Netzwerk. Aus diesem Netzwerk lässt sich eine Übertragungsfunktion zwischen Eingang  $u_1(t)$  und Ausgang  $u_2(t)$  ermitteln, welche  $PT_2$ -Verhalten aufweist:

$$F(s) = \frac{\frac{1}{1+m}}{1 + \frac{1}{1+m} \cdot \left( m \cdot \frac{L}{R_1} + R_1 \cdot C \right) \cdot s + \frac{1}{1+m} \cdot C \cdot L \cdot s^2}.$$

Vergleicht man  $F(s)$  mit der allgemeinen Form  $G^*(s)$  eines  $PT_2$ -Gliedes nach Gl. (4.3), so können daraus in Abhängigkeit der Parameter des RLC-Netzwerkes die Kennwerte für das Beispielnetzwerk bestimmt werden:

- Kreisfrequenz des Netzwerkes

$$\omega_0 = \frac{1}{T} = \frac{1}{\sqrt{\frac{1}{m+1} \cdot C \cdot L}}$$

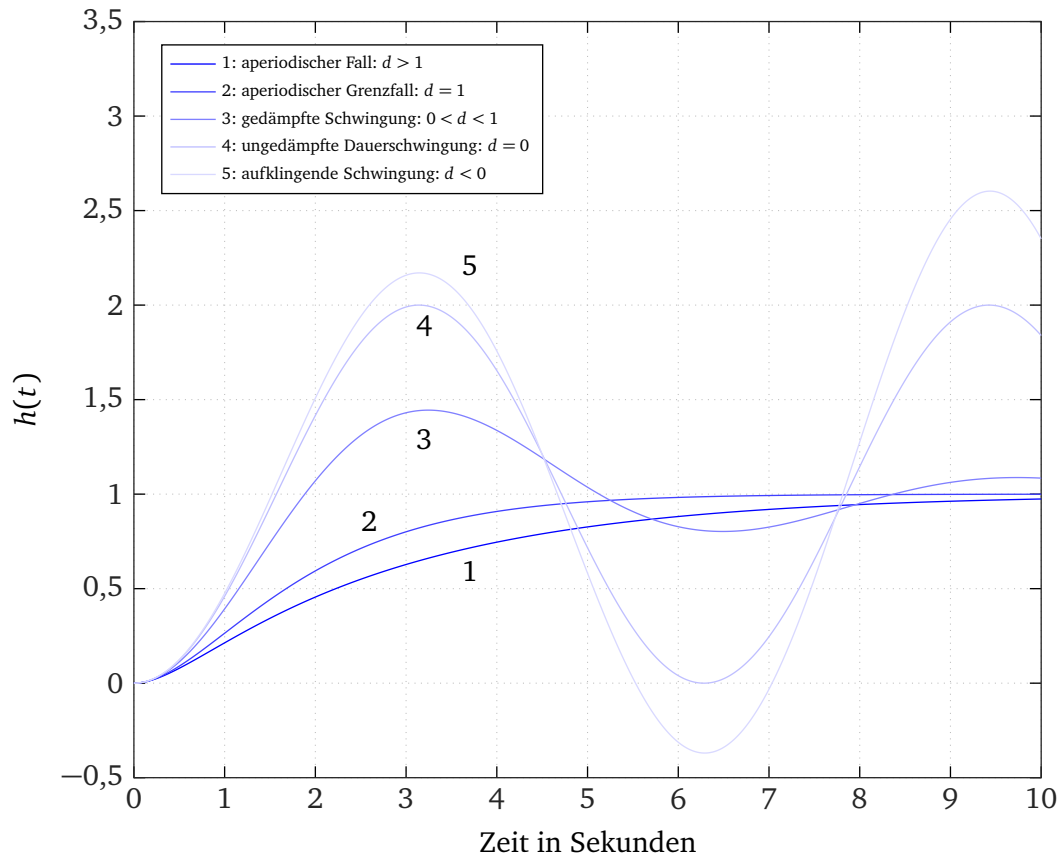
- Stationäre Verstärkung

$$K = \frac{1}{1+m}$$

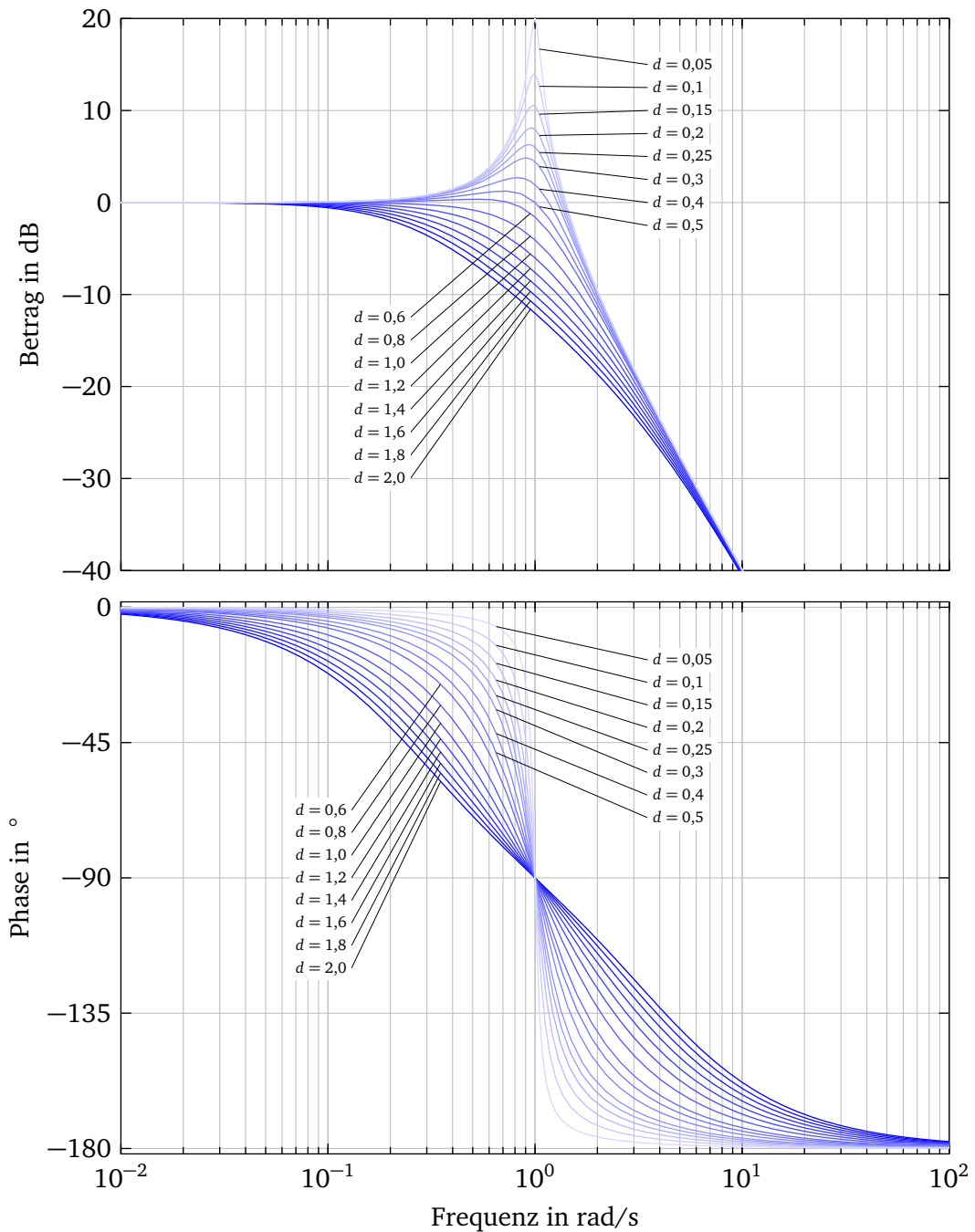
- Dämpfung des Netzwerks

$$d = \frac{1}{2 \cdot \sqrt{C \cdot L \cdot (m+1)}} \cdot \left( m \cdot \frac{L}{R_1} + R_1 \cdot C \right)$$

In Abhängigkeit der Dämpfung  $d$  ergeben sich verschiedene Fälle für den Verlauf der Sprungantwort und des Bode-Diagramms, die in den Abbildungen 4.3 und 4.4 gezeigt werden.



**Abbildung 4.3.:** Sprungantwort eines Verzögerungssystems zweiter Ordnung bei verschiedenen Dämpfungsgraden  $d$  ( $\omega_0 = 1, K = 1$ )



**Abbildung 4.4.:** Bodediagramm eines Verzögerungssystems zweiter Ordnung für verschiedene Dämpfungsgrade  $d$  ( $\omega_0 = 1, K = 1$ )

---

## 4.2 Analyse des Systemverhaltens der Strecke

---

Das System Ventilator soll jetzt mit Hilfe von MATLAB analysiert werden. Eine Übertragungsfunktion wird in MATLAB durch ihr Zähler- und Nennerpolynom beschrieben. Diese Polynome werden in Form von Zeilenvektoren eingegeben, wobei die Vektoren ihre Polynomkoeffizienten in absteigender Potenzfolge enthalten.

Zur besseren Veranschaulichung wird die Analyse für das Beispielnetzwerk mit seiner Übertragungsfunktion  $F(s)$  durchgeführt. Dabei sind die Parameter des RLC-Netzwerks so gewählt, dass es im Weiteren eine Eigenkreisfrequenz von  $\omega_0 = 1 \frac{\text{rad}}{\text{s}}$ , eine Dämpfung von  $d = 0,25$  und eine Verstärkung von  $K = 1$  besitzt, d. h.

$$F(s) = \frac{1}{s^2 + 0,5 \cdot s + 1}.$$

Die Übertragungsfunktion der Strecke  $F(s)$  wird in MATLAB wie folgt eingegeben:

```
>> zaehler = [1];  
>> nenner = [1 0.5 1];
```

Die Übertragungsfunktion ergibt sich aus dem Zähler- und Nennerpolynom durch:

```
>> s1 = tf(zaehler, nenner)  
s1 =  
      1  
-----  
s^2 + 0.5 s + 1  
Continuous-time transfer function.
```

Weiterhin kann man sich die Übertragungsfunktion auch in der Pol/Nullstellen-Form angeben lassen, so dass man direkt Nullstellen, Pole und Verstärkung<sup>1</sup> ablesen kann:

```
>> zpk(s1)  
[...]
```

(Da hier nur ein konjugiert-komplexes Polpaar vorliegt, und zpk diese ausmultipliziert darstellt, ändert sich die Darstellung in diesem Fall nicht.) Mit diesem Befehl kann auch direkt über die Eingabe der Nullstellen, Polstellen und Verstärkung die Übertragungsfunktion eingegeben werden. Für Details siehe `help zpk`.

Für diese Strecke lässt sich die Lage ihrer Pole mit Hilfe des Befehles `roots` ermitteln.

```
>> roots(nenner)  
ans =  
-0.2500 + 0.9682i  
-0.2500 - 0.9682i
```

Alternativ ist es möglich, sich die Pole durch den Befehl `pole` und die Nullstellen durch den Befehl `zero` angeben zu lassen:

---

<sup>1</sup> Hinweis zum Befehl `zpk` in Tabelle A.2 auf Seite 39 beachten.

```
>> pole(s1)
ans =
    -0.2500 + 0.9682i
    -0.2500 - 0.9682i
>> zero(s1)
ans =
Empty matrix: 0-by-1
```

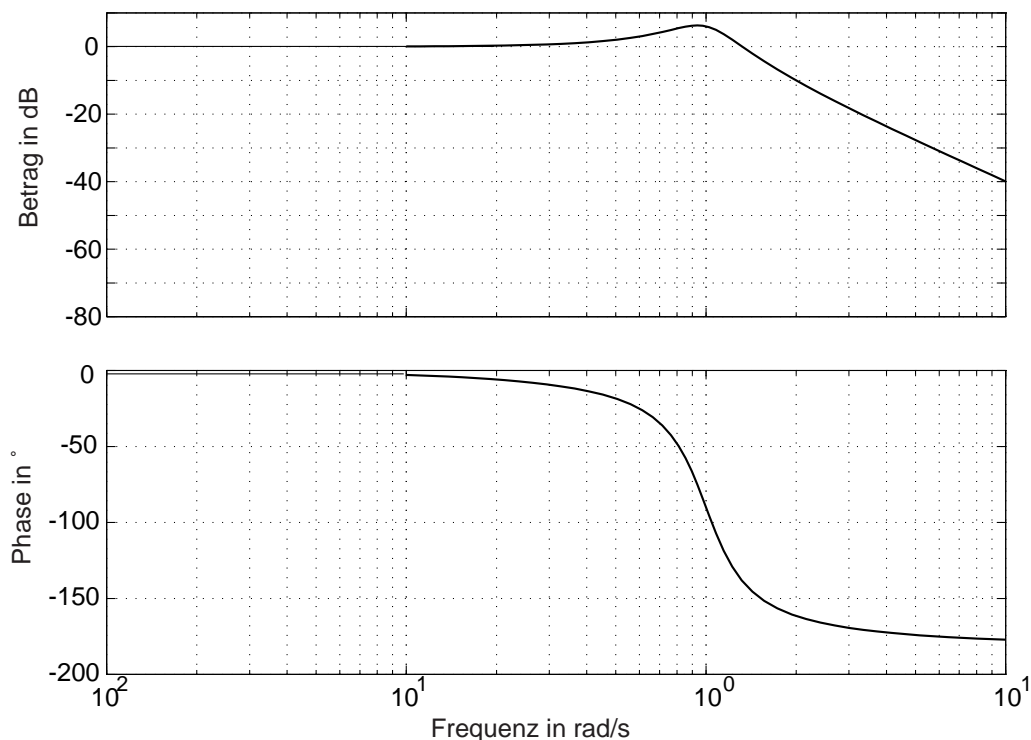
Die graphische Darstellung der Pole und Nullstellen kann man in MATLAB anhand des Befehls pzmap erreichen:

```
>> pzmap(s1)
```

Das Bode-Diagramm der Strecke erhält man durch den Befehl bode:

```
>> bode(s1, {0.01, 10})
```

Der obige Befehl bode berechnet und zeichnet das Bode-Diagramm in Abbildung 4.5 im Bereich der Kreisfrequenz  $0,01 \leq \omega \leq 10 \frac{\text{rad}}{\text{s}}$ . Ohne die Angabe des Frequenzbereiches in geschweiften Klammern wird das Bode-Diagramm in einem von MATLAB automatisch bestimmten Bereich angegeben.



**Abbildung 4.5.:** Bode-Diagramm der  $PT_2$ -Strecke (RLC-Netzwerk)

Alternativ kann auch der Befehl margin verwendet werden, um sich das Bode-Diagramm anzeigen zu lassen.

Um die Dynamik dieser Strecke zu untersuchen, wird ihre Sprungantwort im Bereich  $0 < t < 30 \text{ s}$  mit Hilfe des Befehls step berechnet:

```
>> t = [0:0.1:30].';
>> y = step(s1, t);
```

oder alternativ

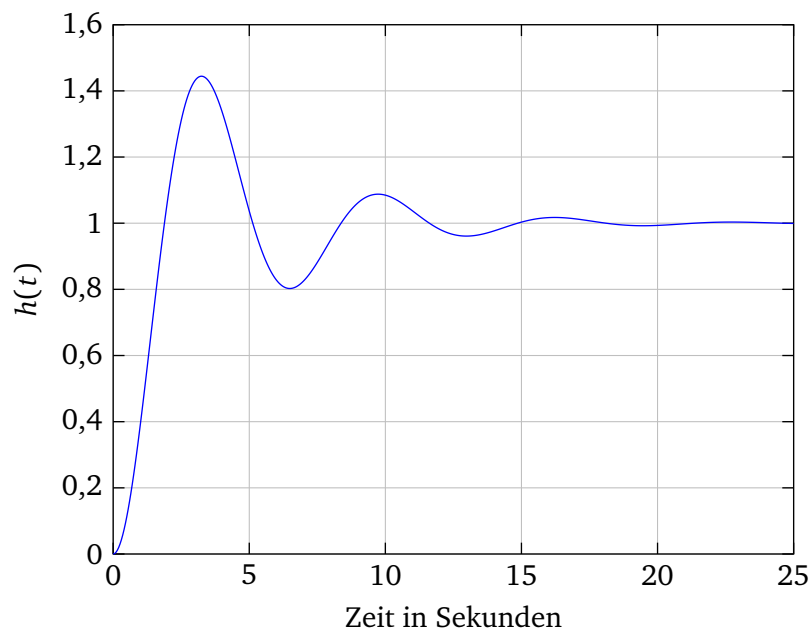
```
>> [y, t] = step(s1, t);
```

Der Spaltenvektor  $y$  enthält die Antwort der Strecke zu den in  $t$  definierten Zeitpunkten.

Der Verlauf der Sprungantwort kann mit Hilfe des Befehles `plot` gezeichnet werden. `grid on` zeichnet Gitternetzlinien ein:

```
>> plot(t, y);  
>> grid on
```

Die Sprungantwort des Beispielsystems ist in Abbildung 4.6 gezeigt.



**Abbildung 4.6.:** Sprungantwort der  $PT_2$ -Strecke (RLC-Netzwerk)

Will man nur den geplotteten Verlauf der Sprungantwort sehen, dann kann `step` auch ohne Rückgabewerte aufgerufen werden. Der Befehl

```
>> step(s1, 30)
```

plottet direkt die Sprungantwort bis 30 s. Lässt man die Angabe der Endzeit weg, so wird diese anhand der Systemdynamik von MATLAB selber bestimmt:

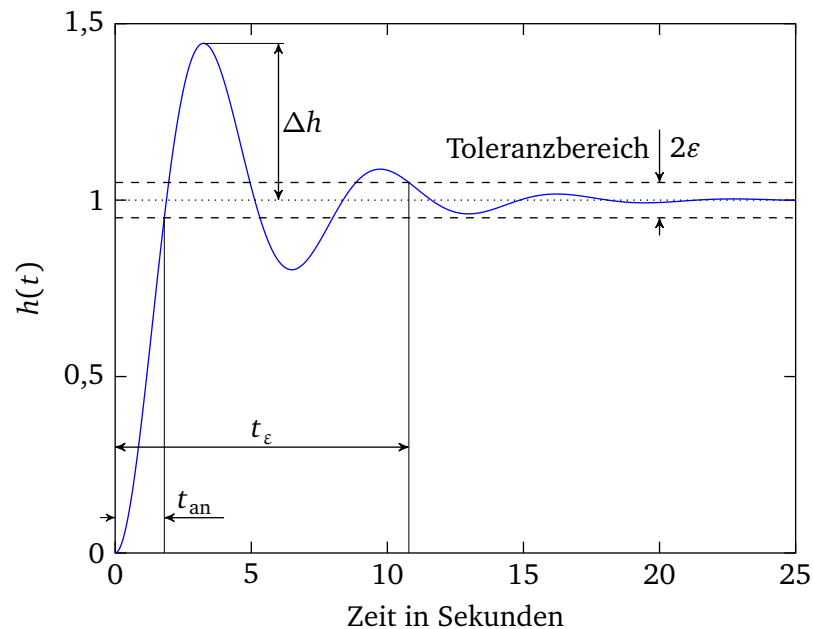
```
>> step(s1)
```

Wie man in Abbildung 4.6 sieht und auch an der Lage der Pole erkennt, weist die Strecke ein starkes Überspringen auf. Den stationären Endwert der Antwort eines Systems auf einen Einheitssprung kann man sich über den Befehl

```
>> dcgain(s1)  
ans =  
1
```

berechnen lassen.

Das Verhalten eines Regelkreises im Zeitbereich wird nach DIN 19226 durch den Verlauf der Regelgröße nach einem Führungsgrößensprung charakterisiert. Dabei werden unter anderem folgende Kenngrößen benutzt (siehe Abbildung 4.7):



**Abbildung 4.7.:** Verlauf der Regelgröße bei einem Führungssprung

- $\Delta h = \frac{h_{\max} - h_{\infty}}{h_{\infty}}$ : Überschwingweite
- $t_{\varepsilon}$ : Ausregelzeit
- $t_{\text{an}}$ : Anregelzeit

#### Hinweis:

Über die Befehle `doc` und `help` bekommen Sie Hinweise über die Benutzung und Informationen zu weiteren verwandten Befehlen. In Tabelle A.2 auf Seite 39 sind die wichtigsten Befehle aufgelistet.

---

### 4.3 Reglerentwurf

---

Die analysierte Strecke soll nun so geregelt werden, dass sich ein gewünschtes Verhalten einstellt.

Hierfür soll einerseits das *Frequenzkennlinienverfahren*, andererseits die *Synthese nach dem Betragsoptimum* genutzt werden. Anschließend sollen die Resultate miteinander verglichen werden.

---

#### 4.3.1 Das Frequenzkennlinienverfahren

---

Beim Frequenzkennlinienverfahren muss man sich, je nach Anforderungen an das System, zunächst für einen Reglertyp entscheiden. Hierbei müssen Forderungen hinsichtlich der Robustheit, der Stabilität, der gewünschten Genauigkeit der Regelung und Geschwindigkeit getroffen werden. Diese werden u. a. durch die Überschwingweite  $\Delta h$ , die Phasenreserve  $\varphi_R$ , die Durchtrittsfrequenz  $\omega_D$  und die Dämpfung  $d$  beschrieben.

Die am häufigsten eingesetzten Reglertypen sind der P-, PI- und PID-Regler. Ihre Übertragungsfunktionen lauten wie folgt:



- **P-Regler**

$$G_R(s) = K_R \quad (4.6)$$

- **PI-Regler**

$$G_R(s) = K_R \cdot \frac{1 + T_R s}{s} \quad (4.7)$$

- **PID-Regler**

$$G_R(s) = K_R \cdot \frac{(1 + T_{R1}s) \cdot (1 + T_{R2}s)}{s} \quad (4.8)$$

Wie man sieht, hat man abhängig vom gewählten Reglertyp unterschiedlich viele Parameter zu bestimmen. Dies sind einerseits die *Reglerzeitkonstanten*  $T_{Ri}$ , andererseits der *Verstärkungsfaktor*  $K_R$ . Es gilt diese nun so zu bestimmen, dass das geregelte System ein gewünschtes Verhalten aufweist.

---

### Wahl der Reglerzeitkonstanten

---

Die Reglerzeitkonstanten (falls vorhanden) werden entsprechend den Zeitkonstanten der Strecke gewählt.

Falls die Nennerzeitkonstanten der Strecke alle unterschiedlich sind, wählt man im Allgemeinen die Reglerzeitkonstanten beim PID-Regler  $T_{R1}$  und  $T_{R2}$  so, dass sie den beiden größten Streckenzeitkonstanten entsprechen. Entsprechend wird bei einem PI-Regler  $T_{R1}$  gleich der größten Streckenzeitkonstanten gewählt.

Falls alle Nennerzeitkonstanten der Strecke  $T_{Ni}$  in etwa gleich groß sind und zusätzlich  $T_{N1} \geq T_{N2} \geq \dots \geq T_{Nn}$  gilt, wählt man die Reglerzeitkonstanten wie folgt:

$$\begin{aligned} \text{PI-Regler:} \quad T_{R1} &= \sum_{i=1}^n T_{Ni} \\ \text{PID-Regler:} \quad T_{R1} &= T_{N1} \\ T_{R2} &= \sum_{i=2}^n T_{Ni} \end{aligned}$$

Bei konjugiert-komplexen Polen  $s_{Ni}$  wird entweder

$$T_{Ri} = \frac{1}{|\operatorname{Re}(s_{Ni})|} \quad \text{oder} \quad T_{Ri} = \frac{1}{|s_{Ni}|}$$

gewählt.

---

### Wahl der Reglerverstärkung

---

Anschließend wird die Reglerverstärkung  $K_R$  so gewählt, dass sich eine ausreichende Phasenreserve  $\varphi_R$  ergibt.

Ausgehend von einer gewünschten Überschwingweite  $\Delta h$ , welche kleiner sein sollte als die maximal zulässige Überschwingweite  $\Delta h_M$ , bestimmt man die nötige Dämpfung  $d \geq d_{\min}$  und daraus die nötige Phasenreserve  $\varphi_R$ .

## Benötigte Formeln

- Zusammenhang zwischen Überschwingweite  $\Delta h$  und Dämpfung  $d$ :

$$\Delta h = \frac{h_{\max} - h_{\infty}}{h_{\infty}} = e^{-\frac{d \cdot \pi}{\sqrt{1-d^2}}} \leq \Delta h_M. \quad (4.9)$$

- Nach der Dämpfung umgestellt erhält man

$$d \geq d_{\min} = \frac{-\ln(\Delta h_M)}{\sqrt{\pi^2 + \ln^2(\Delta h_M)}}. \quad (4.10)$$

- Näherungsweise gilt hier der folgende Zusammenhang zwischen der Dämpfung  $d$  des geschlossenen und der Phasenreserve  $\varphi_R$  des offenen Regelkreises:

$$\varphi_R \approx \frac{120^\circ}{\pi} \cdot d \cdot (3 - d^2). \quad (4.11)$$

Damit diese Phasenreserve erhalten wird, muss für die Phase des offenen Regelkreises bei der Durchtrittsfrequenz  $\omega_D$

$$\angle G(j\omega_D) \stackrel{!}{=} \varphi_R - 180^\circ \quad (4.12)$$

gelten.

## Vorgehensweise

Aus der Frequenzkennlinie des offenen Regelkreises wird die zur gewählten Phasenreserve  $\varphi_R$  gehörige *Durchtrittsfrequenz*  $\omega_D$  ermittelt. Da an dieser Stelle  $|G(j\omega_D)|_{\text{dB}} = 0$  gelten muss, wird durch Variation von  $K_R$  die Betragskennlinie entsprechend angehoben oder abgesenkt.

## Beispiel

Als Beispiel für die Vorgehensweise beim Frequenzkennlinienverfahren soll nun für die oben angegebene Beispielstrecke RLC-Netzwerk ein PI-Regler entworfen werden.

Dieser soll eine maximale Überschwingweite von  $\Delta h_M = 0,5$  nicht überschreiten. Die Übertragungsfunktion der Beispielstrecke ist

$$F(s) = \frac{1}{s^2 + 0,5 \cdot s + 1} = \frac{1}{(s + 0,25 + 0,9682j) \cdot (s + 0,25 - 0,9682j)}.$$

Damit wird also, der Regel entsprechend, eine Zeitkonstante  $T_R = \frac{1}{|\text{Re}(s_N)|} = 4$  gewählt, so dass für den Regler

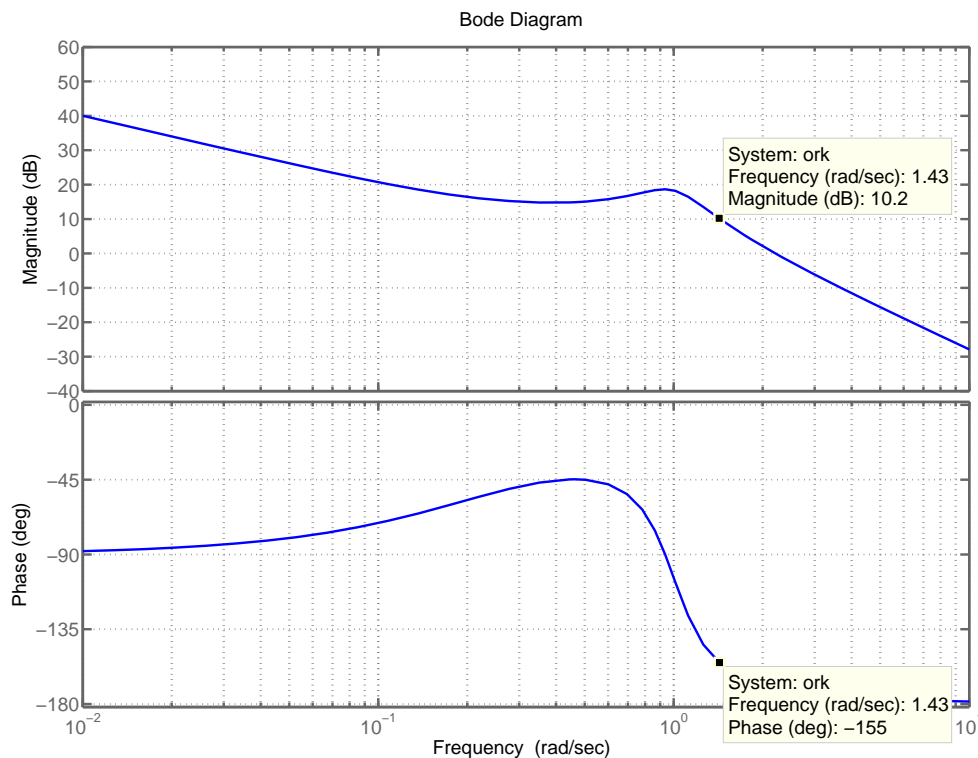
$$G_R(s) = K_R \frac{1 + 4s}{s}$$

gilt.

Als benötigte Dämpfung ergibt sich  $d \geq 0,2154$  bzw. eine benötigte Phasenreserve von  $\varphi_R \geq 24,30^\circ$ , wodurch man die Phase von  $\angle G(j\omega_D) = -155,7^\circ$  betrachten muss. Bei dieser Phase hat die Strecke eine Durchtrittsfrequenz von  $\omega_D = 1,43 \frac{\text{rad}}{\text{s}}$ . Damit  $|G(j \cdot 1,43)|_{\text{dB}} = 0$  gilt, muss die Betragskennlinie, wie in Abbildung 4.8 zu sehen ist, um ca. 10,2 dB abgesenkt werden, wodurch ein  $K_R \approx 0,309$  eingestellt werden muss. Insgesamt ergibt sich schließlich ein PI-Regler mit der Übertragungsfunktion

$$G_R(s) = 0,309 \cdot \frac{1 + 4 \cdot s}{s}.$$

In Abbildung 4.9 sind die Sprungantworten der unregulierten Strecke und des über den PI-Regler geschlossenen Regelkreises zu sehen (Der Befehl `feedback` erzeugt die Übertragungsfunktion des geschlossenen Regelkreises, wobei unbedingt die Form des Regelkreises in der Dokumentation der Funktion zu beachten ist).



**Abbildung 4.8.:** Bode-Diagramm des offenen Regelkreises des RLC-Netzwerks

### 4.3.2 Die Synthese nach dem Betragsoptimum

#### Voraussetzungen für die Anwendung des Betragsoptimums

Die Synthese nach dem Betragsoptimum geht von einem Standardregelkreis nach Abbildung 4.10 aus. Damit die Synthese nach dem Betragsoptimum angewendet werden kann, muss gelten, dass die Strecke  $G_S(s)$  ein reines Verzögerungsglied ist. Die Strecke muss also durch eine Übertragungsfunktion der Art

$$G_S(s) = \frac{1}{a_0 + a_1 \cdot s + a_2 \cdot s^2 + \dots} = \frac{1}{A(s)}$$

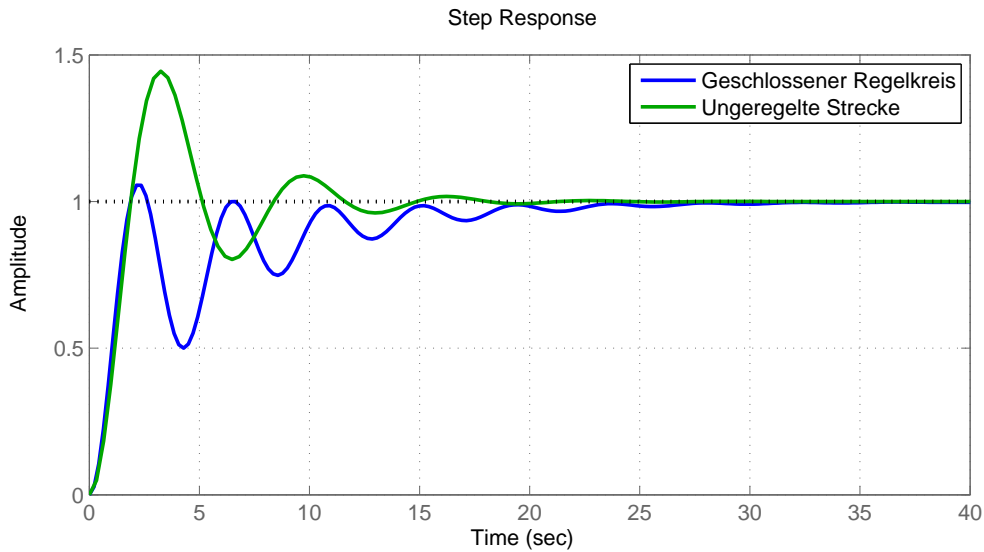
beschreibbar sein.

Im Fall eines PID-Reglers wird der Regler  $G_R(s)$  so angesetzt, dass seine Übertragungsfunktion in der Form

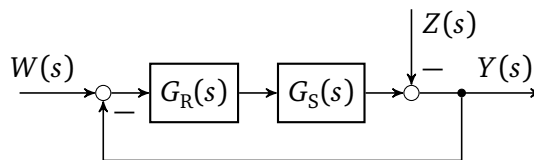
$$G_R(s) = k_R \cdot \left( 1 + \frac{1}{T_N \cdot s} + T_V \cdot s \right)$$

vorliegt. Diese wird zunächst in eine für die weitere Rechnung bessere Form

$$G_R(s) = \frac{2k_R \cdot s + \frac{2k_R}{T_N} + 2k_R T_V s^2}{2 \cdot s} = \frac{r_0 + r_1 s + r_2 s^2}{2 \cdot s} = \frac{R(s)}{2 \cdot s} \quad (4.13)$$



**Abbildung 4.9.:** Sprungantwort des geschlossenen RK und der Strecke



**Abbildung 4.10.:** Standardregelkreis

mit den Parametern

$$r_0 = \frac{2k_R}{T_N}, \quad r_1 = 2k_R \quad \text{und} \quad r_2 = 2k_R T_V. \quad (4.14)$$

gebracht.

Analog wird ein PI-Regler durch die neue Übertragungsfunktion

$$G_R(s) = \frac{r_0 + r_1 \cdot s}{2 \cdot s} \quad (4.15)$$

beschrieben. (Der PI-Regler ergibt sich aus dem PID-Regler der obigen Darstellung dadurch, dass  $T_V = 0$  gesetzt wird.)

---

### Berechnung der Reglerparameter nach dem Betragsoptimum

---

Das Betragsoptimum zielt nun darauf ab, dass der Betrag der Führungsübertragungsfunktion  $|F_w(s)|$  zu 1 gemacht wird, sich also wie ein P-Glied mit Verstärkungsfaktor  $K = 1$  verhält. Die Führungsübertragungsfunktion  $F_w(s)$  wird durch

$$F_w(s) = \frac{G_R(s) \cdot G_S(s)}{1 + G_R(s) \cdot G_S(s)}$$

berechnet.

---

Zur Bestimmung der Parameter des Reglers müssen also die Faktoren  $r_0$ ,  $r_1$  und gegebenenfalls  $r_2$  so bestimmt werden, dass  $F_w(s)$  diese Bedingungen möglichst gut erfüllt. Es gelten die Bedingungen

$$r_0 = \frac{(a_0 a_2 - a_1^2) \cdot a_0}{a_0 a_3 - a_1 a_2}, \quad r_1 = -\frac{a_0^2 a_3 - 2a_0 a_1 a_2 + a_1^3}{a_0 a_3 - a_1 a_2},$$

wobei nur die für den hier durchzuführenden Versuch benötigten angegeben sind.

Die Variablen  $a_0 \dots a_3$  werden anhand der Streckenübertragungsfunktion bestimmt. Sie entsprechen den, wie oben angegeben, einzelnen Polynomwerten des Nennerpolynoms, wenn der Zähler zu 1 normiert ist.

Die Gleichungen zur Bestimmung von  $r_0$ ,  $r_1$  und  $r_2$  für einen PID-Regler sind in [2] zu finden.



# Versuch 5

## 5.1 Kurzeinführung in Simulink

Simulink ist eine Programmiererweiterung zu MATLAB und dient der graphisch unterstützten Simulation von dynamischen Systemen. Zur Darstellung von dynamischen Systemen werden Blockdiagramme verwendet, die aus verschiedenen Bibliotheken kopiert, verändert oder auch selbst erstellt werden können.

Nach dem Start von MATLAB kann Simulink entweder durch einen Mausklick auf das Simulink-Symbol (siehe I) oder alternativ über die Eingabe von `simulink` im **MATLAB Command Window** (siehe II) gestartet werden (Abbildung 5.1).

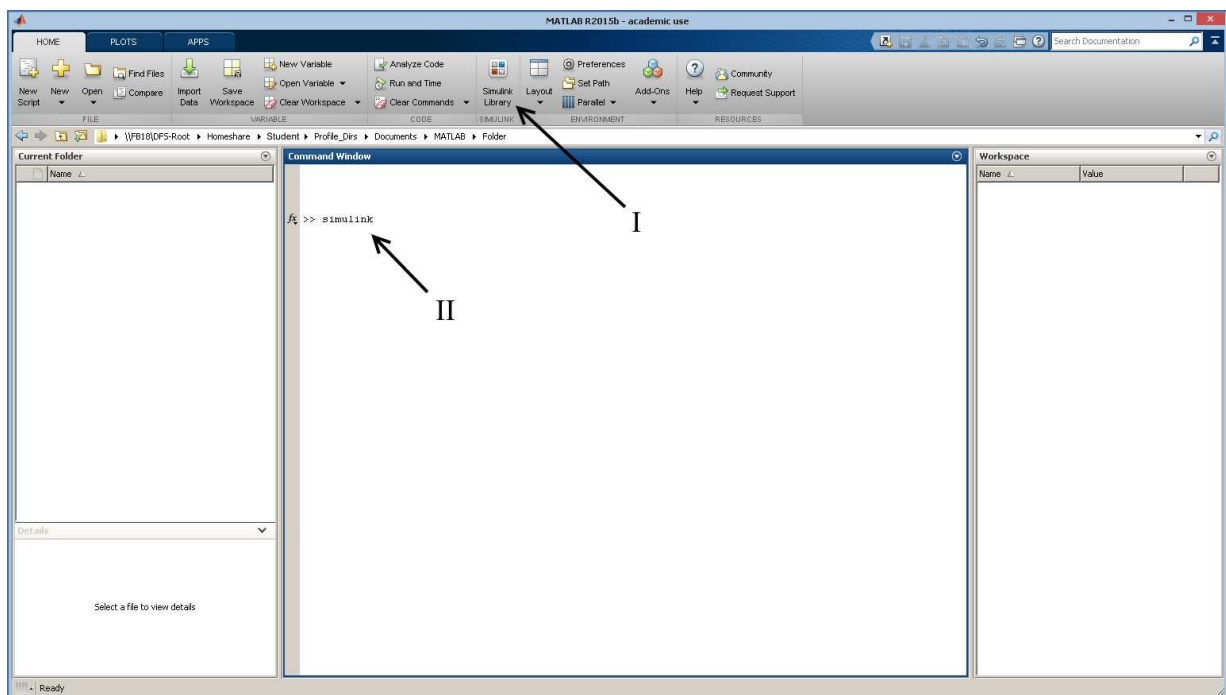


Abbildung 5.1.: Aufruf von MATLAB/Simulink

Es erscheint der **Simulink Library Browser** (Abbildung 5.2). Dieser beinhaltet verschiedene Bibliotheken, geordnet nach unterschiedlichen Oberbegriffen wie Sinks, Sources, Continuous, etc. Diese enthalten die zum Aufbau eines Modells benötigten Funktionsblöcke. Tabelle 5.1 gibt eine kleine Auflistung der wichtigsten Elemente und ihren Verzeichnisnamen an. Es sei darauf verwiesen, dass diese Tabelle nur einen kleinen Auszug des Anwendungsspektrums von Simulink wiedergibt. Eine Vertiefung wird es im MATLAB-Praktikum II geben.

Mittels **File** → **New** kann ein neues Modell erzeugt werden. Durch einfaches Drag-and-Drop können die verschiedenen Blöcke in das eigene Modell übertragen werden. Die übertragenen Blöcke müssen nur noch verbunden werden. Hierzu wird der Block ausgewählt, von dem eine Verbindung ausgehen soll und anschließend mit gedrückter **Strg**-Taste der Zielblock ausgewählt. Alternativ kann auch der Ausgang des einen Blockes mit dem Eingang des anderen Blockes durch selbstständiges „Verlegen der Leitung“ verbunden werden.

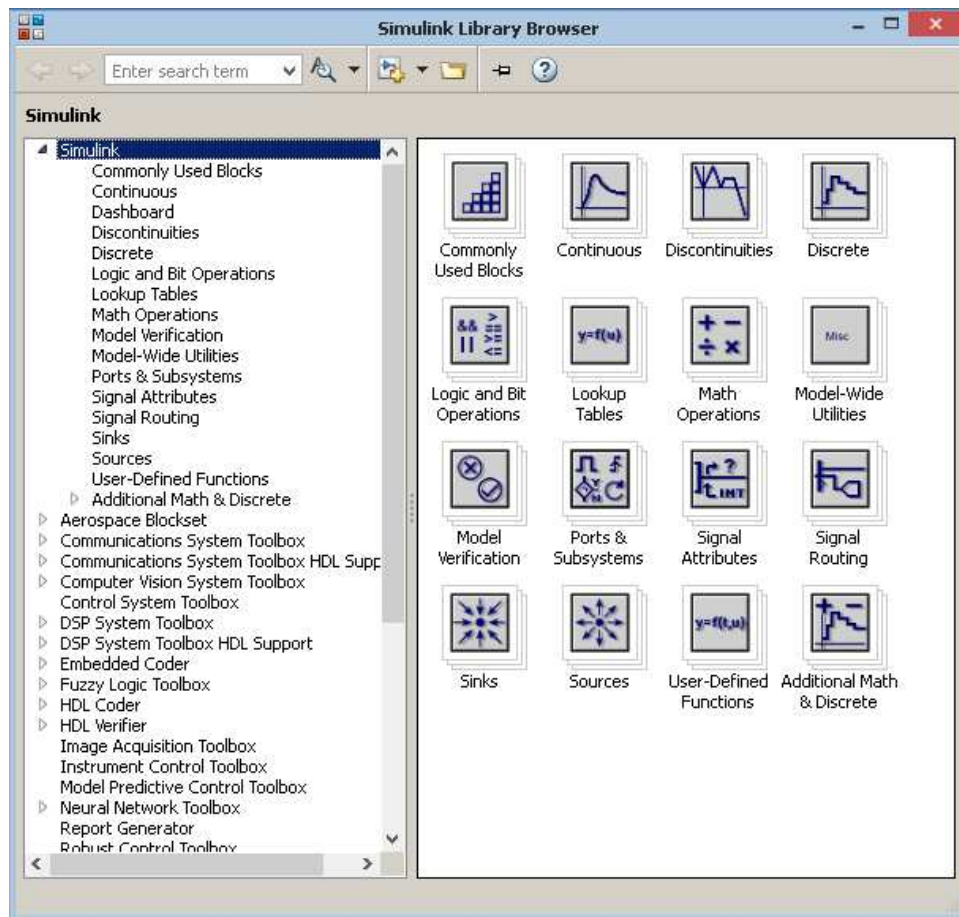


Abbildung 5.2.: Simulink Library Browser

Tabelle 5.1.: Strukturübersicht der Bibliothek in Simulink

Verzeichnisnamen	Inhalt
Commonly Used Blocks	<ul style="list-style-type: none"> <li>- Multiplexoren (MUX)</li> <li>- Summationsstellen (SUM)</li> <li>- Teil- / Subsysteme (Subsystem)</li> <li>- Verstärker (Gain)</li> </ul>
Continuous	<ul style="list-style-type: none"> <li>- Integratoren (Integrator)</li> <li>- Übertragungsfunktionen (Transfer Fcn, Zero-Pole)</li> </ul>
Sink	<ul style="list-style-type: none"> <li>- Oszilloskope (Scope)</li> </ul>
Source	<ul style="list-style-type: none"> <li>- Sprünge (Step)</li> <li>- Rampensignale (Ramp)</li> <li>- Rechteckschwingungen (Discrete Pulse Generator)</li> </ul>

Die einzelnen Blöcke haben zunächst festgelegte Parameter (Verstärkung, Verzögerung, Anfangswerte, Schrittweite, ...), welche durch einen Doppelklick auf den Block verändert werden können.

Als Beispiel sei hier der Block **Continuous** → **Transfer Fcn** beschrieben. Mit ihm können beliebige Übertragungsfunktionen dargestellt werden. Als einzige Einschränkung darf der Zählergrad nicht größer als der Nennergrad sein („Kausalität“). Zähler (engl. Numerator) und Nenner (engl. Denomi-



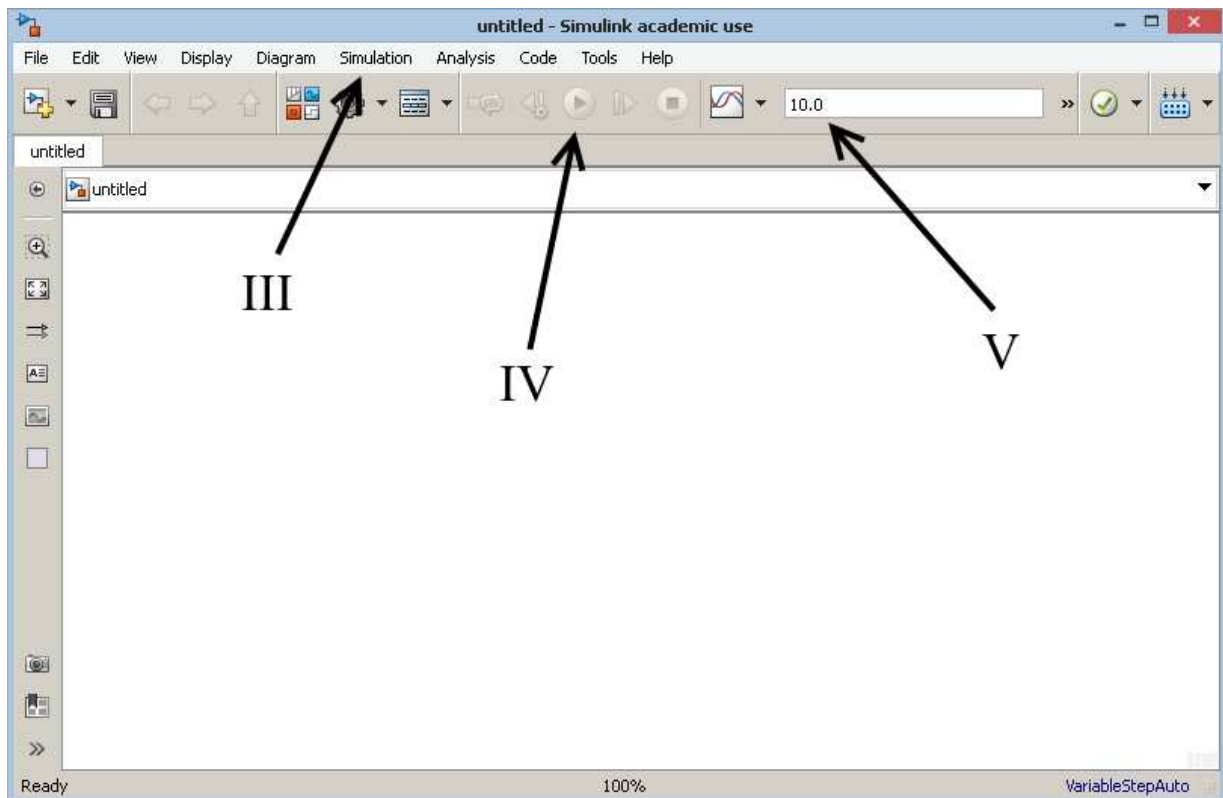


Abbildung 5.3.: Neues Modell

nator) werden, wie in MATLAB üblich, durch Vektoren dargestellt. Deren Inhalte sind die Koeffizienten  $b_i$  und  $a_i$  der Zähler- und Nennerpolynome  $b_ms^m + b_{m-1}s^{m-1} + \dots + b_0s^0$  und  $a_ns^n + a_{n-1}s^{n-1} + \dots + a_0s^0$ .

### Beispiel

Eine Eingabe von

```
Numerator coefficient = [1]
Denominator coefficient = [2, 1]
```

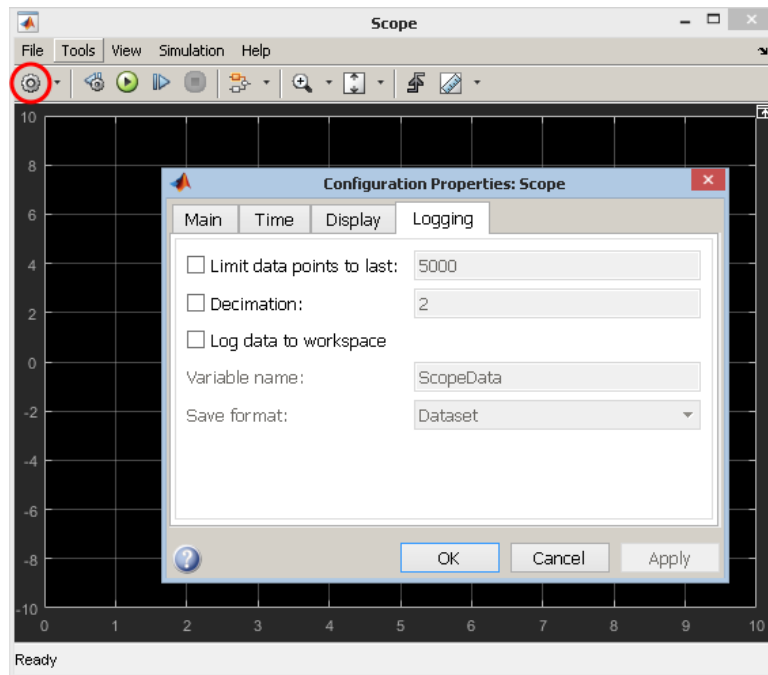
ergibt die Übertragungsfunktion  $G(s) = \frac{1}{2s+1}$ .

Wird ein Modell zu komplex, ist eine Unterteilung in mehrere Teilsysteme sinnvoll, um die Übersichtlichkeit zu wahren. Dafür steht im Verzeichnis **Commonly Used Blocks** der Block **Subsystem** zur Verfügung. Durch einen Doppelklick auf diesen Block wird ein Teilsystem mit beliebig vielen Ein- und Ausgängen erstellt. Alternativ kann auch zunächst ein Modell erzeugt werden, dann mehrere Blöcke umrahmt und durch einen Rechtsklick die Funktion **Create Subsystem** ausgewählt werden. Die markierten Elemente werden dann zu einem Subsystem zusammengefasst.

Ist das Modell vollständig, kann die Simulation über **Simulation** → **Start** (siehe III), über den **Play-Button** (siehe IV) oder über MATLAB mit dem Befehl `sim Modelname.slx` bzw. `sim Modelname.mdl` gestartet werden (Abbildung 5.3). Die Simulationszeit kann direkt im Modell eingestellt werden (siehe V).

Das Ergebnis der Simulation kann dann durch einen Doppelklick auf den Block **Scope** eingesehen werden. Es ist möglich, die Ausgabe der Simulation über die Scopeeinstellungen an MATLAB zu übergeben. Hierzu muss das Scope mittels Doppelklick geöffnet werden. Dort kann in den Parametereinstellungen (vgl. Abbildung 5.4) im Register „Logging“ die Option „Log data to workspace“

aktiviert werden. Nach der Simulation werden die Daten unter der gewählten Variablenbezeichnung im Workspace abgelegt. Der Zugriff auf die Wert kann über die Variablenbezeichnung z.B. `ScopeData.signals.values` in MATLAB erfolgen. Die Ausgabe kann auch über den Funktionsblock **To Workspace** unter **Sinks** direkt, ohne Anzeige in einem Scope, an den Workspace erfolgen.



**Abbildung 5.4.:** Parametereinstellungen im Scope

Die Standardeinstellung des Scopes steht auf 5000 Datenpunkten. Das heißt, dass nur die letzten 5000 Datenwerte angezeigt werden. Für Simulationen mit mehr als 5000 Ausgabenwerten muss die Zahl der Datenwerte erhöht oder die Begrenzung ausgeschaltet werden. Dies ist ebenfalls über die Parametereinstellung „Limit data points to last“ im Scope möglich.

Soll dem Modell noch Text (erklärend, Bezeichnung, Dokumentation usw.) hinzugefügt werden, so kann dieser an eine durch einen Doppelklick in den freien Bereich des Modells ausgewählten Stelle geschrieben werden.

## 5.2 Regelung des Ventilators unter Simulink

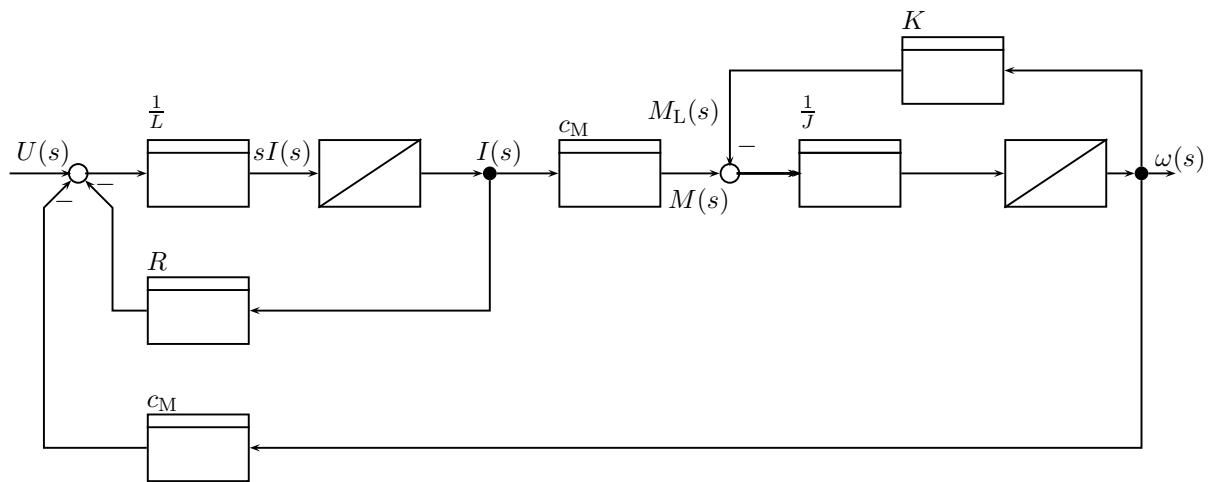
### 5.2.1 Der Ventilator als Regelstrecke

In diesem Versuch wird als Regelstrecke das vereinfachte Modell des Ventilators verwendet, welches in Abbildung 5.5 als Blockschaltbild dargestellt ist. (Siehe auch Abschnitt A.1.1 ab Seite 33.)

Die Modellparameter können der Tabelle A.1 auf Seite 38 entnommen werden.

Die Konstante  $K$  steht für die Linearisierung  $2\beta\omega_s$  des Lastmomentes. Im späteren Verlauf soll ebenfalls das nichtlineare Modell erprobt werden. Hierfür müssen Sie diese Konstante nur durch  $\beta\omega^2(s)$  ersetzen.

Wenn sich im Modell Parameter häufig wiederholen oder ändern, empfiehlt es sich, die Parameter nicht für jede Simulation erneut einzugeben, sondern sie als Variablen in den Workspace zu speichern. Dazu wird statt eines Zahlenwertes eine Variable als Parameter in den Block eingegeben. Zu Beginn der Simulation lädt Simulink die Variablen aus dem Workspace. Daher können die Werte der Variablen



**Abbildung 5.5.:** Blockdiagramm des Ventilators (linearisiert)

auch in einem selbsterstellten m-File gesetzt werden. Durch Ausführung des m-Files werden dann alle Variablen im Workspace angelegt und dann als Parameter in das Modell übernommen. Sollen dann Parameter geändert werden, muss nur noch der Wert der zugehörigen Variablen im m-File geändert werden. Das m-File muss dann allerdings neu ausgeführt werden, weil es nicht automatisch von Simulink gestartet wird. Parameteränderungen direkt am Modell sind dadurch nicht notwendig, was erheblichen Mehraufwand einspart.

### 5.2.2 Reglerentwurf für den Ventilator

Als Regler sollen zunächst die in Versuch 4 sowohl anhand des Frequenzkennlinienverfahrens erstellten P- und PI-Regler als auch der mittels Betragsoptimums erhaltene PI-Regler verwendet werden.

$$P_{\text{Freq.}} : G_R(s) = 263,0268 \quad (5.1)$$

$$PI_{\text{Freq.}} : G_R(s) = 436,516 \cdot \frac{0,5917 \cdot s + 1}{s} \quad (5.2)$$

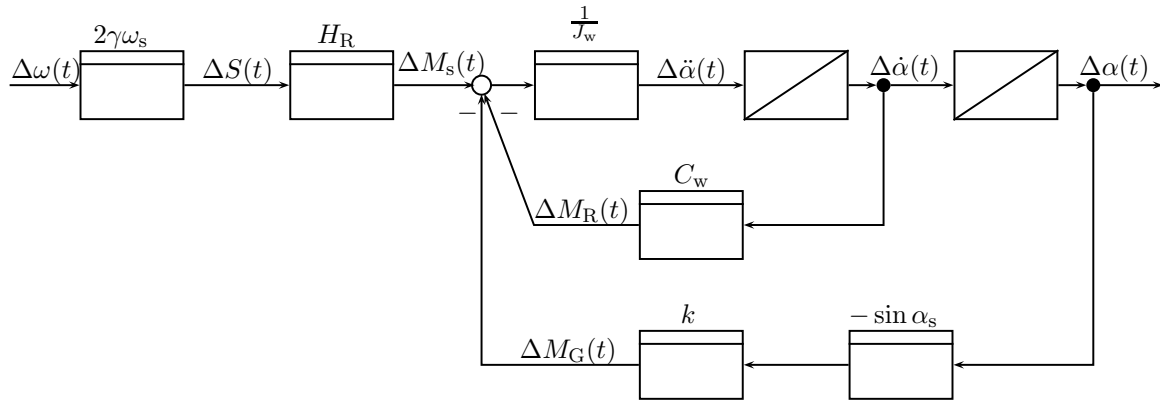
$$PI_{\text{Betragsopt.}} : G_R(s) = \frac{138,9s + 234,9}{2s} \quad (5.3)$$

Da sich, wie aus Versuch 4 noch bekannt, der Regelkreis mit den obigen Reglern sehr schnell einschwingt, sollte bei der Simulation die Simulationszeit (siehe V in Abbildung 5.3) auf einen Wert von 0,001 Sekunden eingestellt werden. Bei Beibehaltung der Standardeinstellung von 10 Sekunden wäre der Einschwingvorgang nicht zu sehen.

## 5.3 Regelung des Pendelschraubers unter Simulink

### 5.3.1 Der vollständige Pendelschrauber als Regelstrecke

Im Folgenden soll der vollständige Pendelschrauber in Simulink simuliert werden. Erweitern Sie dazu das *lineare* Teilsystem des Ventilators um das Teilsystem Hebel in Abbildung 5.6. Dieser Teil entspricht dem *linearisierten* Hebel, siehe auch Abschnitt A.1.2.



**Abbildung 5.6.:** Blockdiagramm des zweiten Teils des Pendelschraubers (linearisiert)

### Übertragungsfunktion des Gesamtsystems

Die Parameter des Systems können Abschnitt A.1.2 ab Seite 35 und der Tabelle A.1 auf Seite 38 entnommen werden.

Für  $k$  gilt

$$k = -0,5 \cdot m_{SL} \cdot g \cdot S_L - m_L \cdot g \cdot H_L + 0,5 \cdot m_{SR} \cdot g \cdot S_R + m_P \cdot g \cdot H_R .$$

Es empfiehlt sich, das bereits vorhandene m-File mit den obigen Parametern zu erweitern, und den Wert für  $k$  aus diesen Parametern zu berechnen.

Die Übertragungsfunktion des linearisierten Pendelschraubers lässt sich durch die beiden Teilübertragungsfunktionen

$$G_{\text{Ventilator}}(s) = \frac{c_M}{L \cdot J_P \cdot s^2 + (R \cdot J_P + 2 \cdot \beta \cdot \omega_s \cdot L) \cdot s + (2 \cdot \beta \cdot \omega_s \cdot R + c_M^2)} \quad (5.4)$$

und

$$G_{\text{Hebel}}(s) = \frac{2 \cdot \gamma \cdot \omega_s \cdot H_R}{J_W s^2 + c_W s + (-k \cdot \sin(\alpha_s))} \quad (5.5)$$

als

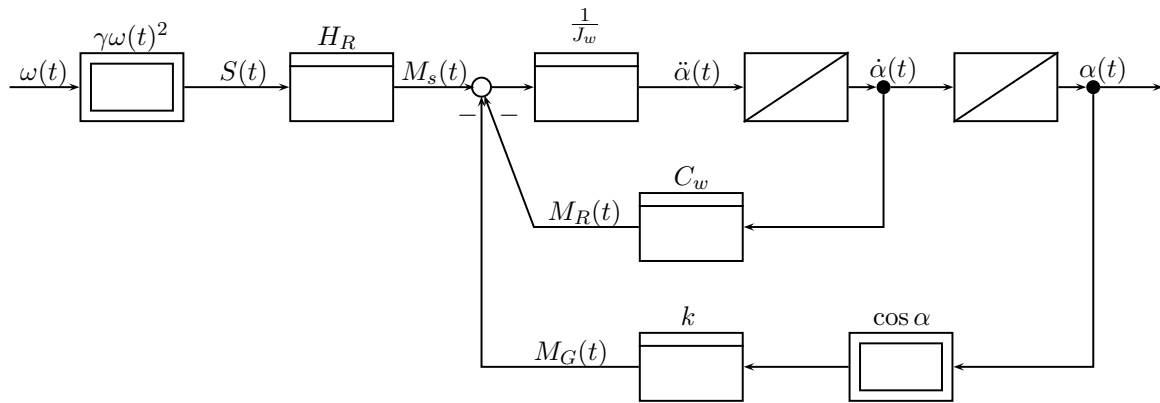
$$G(s) = G_{\text{Ventilator}}(s) \cdot G_{\text{Hebel}}(s) \quad (5.6)$$

zusammensetzen.

Auch hier empfiehlt es sich, die Übertragungsfunktionen mittels des Befehls `tf` im bereits angelegten m-File zu realisieren.

### Das nichtlineare Modell des Hebels

Abbildung 5.6 stellt das linearisierte Modell des zweiten Teils des Pendelschraubers dar. Der für das linearisierte Modell ermittelte PID-Regler soll auch am nichtlinearen Modell getestet werden. In Abbildung 5.7 ist das Blockschaltbild für den nicht linearisierten Fall angegeben.



**Abbildung 5.7.:** Blockdiagramm des zweiten Teils des Pendelschraubers (nicht linearisiert)

### 5.3.2 Reglerentwurf für den vollständigen Pendelschrauber

Im Folgenden soll für den gesamten Pendelschrauber ein PID-Regler mittels Frequenzkennlinienverfahren entworfen werden. Eine mögliche Darstellungsform der Übertragungsfunktion eines PID-Reglers lautet

$$G_R(s) = K_R \cdot \frac{(1 + T_{R1}s) \cdot (1 + T_{R2}s)}{s}. \quad (5.7)$$

Die genaue Vorgehensweise des Frequenzkennlinienverfahrens ist Abschnitt 4.3.1 von Versuch 4 zu entnehmen. Die wichtigsten Formeln des Frequenzkennlinienverfahrens sind auf Seite 50 gegeben.

#### Realisierung der Regler

Die aus der Vorlesung SDRT I bekannten Standardregler P-, PI-, ...-Regler können in verschiedenen Formen dargestellt werden. Jede dieser Formen hat Vor- und Nachteile in Theorie und Praxis. Da in der praktischen Umsetzung häufig die Struktur des Reglers fest vorgegeben ist, müssen die für die Theorie „angenehmeren“ zunächst in die vorliegende Formen umgerechnet werden.

#### Beispiel PI-Regler

- Pol-/Nullstellenform (Befehl `zpk`):

$$G_{R1}(s) = K \cdot \frac{s - s_1}{s}$$

- Pol-/Nullstellenform, über Zeitkonstanten parametrisiert:

$$G_{R1}(s) = K' \cdot \frac{1 + T \cdot s}{s}$$

- Koeffizientenform (Befehl `tf`):

$$G_{R2}(s) = \frac{a_1s + a_2}{s}$$

- Parallelschaltung aus P- und I-Glied:

$$G_{R3}(s) = K_p + K_I \cdot \frac{1}{s}$$

- 
- Angabe von Verstärkung und Nachstellzeit:

$$G_{R4}(s) = K_R \cdot \left( 1 + \frac{1}{T_N \cdot s} \right)$$

---

# Versuch 6

---

## 6.1 Die Wurzelortskurve

---

---

### 6.1.1 Grundlagen

---

Bis jetzt wurde bei der Analyse eines Systems nur die aktuelle Lage der Pole im offenen Regelkreis  $F_o(s)$  betrachtet. Das Wurzelortskurvenverfahren betrachtet die Lage der Nullstellen des charakteristischen Polynoms des geschlossenen Regelkreises in Abhängigkeit eines Parameters, meistens eines Reglerparameters. Das Führungsübertragungsverhalten wird durch die Übertragungsfunktion

$$F_w(s) = \frac{F_o(s)}{1 + F_o(s)}$$

charakterisiert, wobei

$$F_o(s) = \frac{k \cdot Z_o(s)}{N_o(s)}$$

die Übertragungsfunktion des offenen Kreises ist. In diesem Fall ist  $k$  der zu variierende Verstärkungsparameter des Reglers. Eingesetzt ergibt sich damit

$$F_w(s) = \frac{k \cdot Z_o(s)}{k \cdot Z_o(s) + N_o(s)} .$$

Die Pole des geschlossenen Regelkreises entsprechen den Nullstellen des Nennerpolynoms von  $F_w(s)$ , d. h. für die Pole gilt

$$1 + F_o(s) = 0$$

bzw.

$$N_o(s) + k \cdot Z_o(s) = 0 . \tag{6.1}$$

Die Gleichung (6.1) wird auch als *charakteristische Gleichung* bezeichnet.

In vielen Fällen interessiert, wie sich diese charakteristische Gleichung (und damit die Pole des geschlossenen Regelkreises) in Abhängigkeit von einem Reglerparameter verändert.

Dies wird im Folgenden anhand eines Beispiels analysiert. Als Beispielstrecke dient ein  $IT_1$ -Glied mit der Übertragungsfunktion

$$G_s(s) = \frac{1}{s \cdot (s + 1)} .$$

Die Strecke hat einen Pol in  $s = 0$  und einen Pol in  $s = -1$ . Als Regler wird der Einfachheit halber einen P-Regler mit

$$G_r(s) = k$$

verwendet.

Die Frage ist nun, für welche  $k$  das geregelte System stabil ist.

Dazu wird zunächst die Übertragungsfunktion

$$F_o(s) = G_s(s) \cdot G_r(s) = \frac{k}{s \cdot (s + 1)} \quad (6.2)$$

des offenen Regelkreises betrachtet.

Verwendet man nun Gleichung (6.1), um die charakteristische Gleichung des Systems und daraus die Pole zu bestimmen, erhält man

$$k + s \cdot (s + 1) = 0$$

bzw.

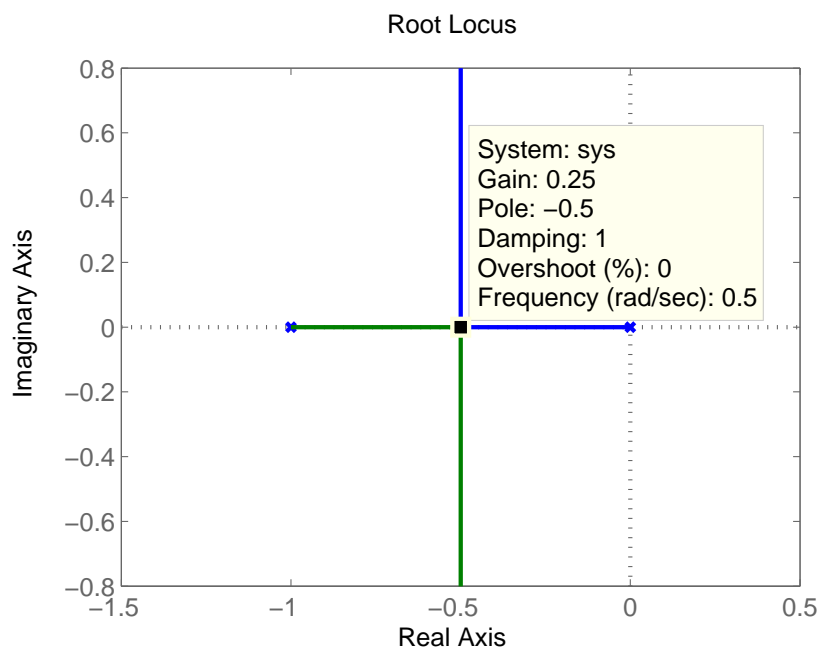
$$s^2 + s + k = 0 .$$

Damit ergeben sich die Pole

$$s_{1,2} = -0,5 \pm \sqrt{0,25 - k} .$$

Man erkennt, dass für Werte von  $k \geq 0,25$  die Pole komplex werden. Das System ist aber für alle  $k \geq 0$  stabil.

Dies ist in Abbildung 6.1 anhand der Wurzelortskurve (WOK) veranschaulicht.



**Abbildung 6.1.:** WOK des Beispielsystems IT<sub>1</sub> mit P-Regler



---

## 6.1.2 Konstruktion der WOK

---

Für die Konstruktion der WOK gibt es mehrere Regeln. Ausführlich werden diese Regeln in den Vorlesungen „Systemdynamik und Regelungstechnik 1“ und „Systemdynamik und Regelungstechnik 2“ betrachtet. Für eine detaillierte Behandlung sei deshalb auf [2] und [1] verwiesen.

Für dieses Praktikum werden nur einige grundlegende Regeln benötigt:

1. Die WOK (für  $k \geq 0$ ) beginnt in den Polen und endet in den Nullstellen des offenen Regelkreises.  $s = \infty$  ist dabei als  $(n - m)$ -fache Nullstelle mitzuzählen.
2. Ein Punkt auf der reellen Achse gehört genau dann zur WOK (für  $k \geq 0$ ), wenn rechts von ihm eine ungerade Anzahl von kritischen Stellen (Pole bzw. Nullstellen) liegt.
3. Die Verzweigungspunkte  $\sigma$  der WOK folgen aus

$$\sum_{\mu=1}^m \frac{1}{\sigma - n_{\mu}} - \sum_{\nu=1}^n \frac{1}{\sigma - p_{\nu}} = 0. \quad (6.3)$$

Voraussetzung: Alle Pole  $p_{\nu}$  und alle Nullstellen  $n_{\mu}$  sind reell.

4. Die Ordinate  $\omega$  und die Verstärkung  $k_0$  an der Stabilitätsgrenze folgen aus

$$k_0 \cdot Z_o(j\omega) + N_o(j\omega) = 0. \quad (6.4)$$

5. Der Wurzelschwerpunkt der nach Unendlich laufenden WOK-Äste ergibt sich durch

$$\sigma_s = \frac{\sum_{j=1}^m \operatorname{Re}(n_j) - \sum_{i=1}^n \operatorname{Re}(p_i)}{m - n}, \quad (6.5)$$

wobei  $n$  die Anzahl der Pole und  $m$  die Anzahl der Nullstellen angibt.

6. Für den Winkel der Asymptoten im Wurzelschwerpunkt gilt

$$\varphi_i = \frac{2i + 1}{n - m} \cdot \pi, \quad i = 0, 1, \dots, n - m - 1. \quad (6.6)$$

Besonders die Regeln 3 und 4 sind hier wichtig. Regel 4 gibt Auskunft über den Bereich, in dem  $k$  liegen darf, damit der geschlossene Regelkreis stabil ist. Regel 3 gibt die Verzweigungspunkte auf der reellen Achse der WOK an. Diese gehören zum asymptotischem Grenzfall, der als Regelverhalten oft gewünscht wird, da dort das geregelte System möglichst schnell aber noch nicht schwingungsfähig ist.

Damit aus der Lage des Verzweigungspunktes auch die dafür benötigte Reglerverstärkung berechnet werden kann, wird noch eine weitere Formel benötigt. Diese ergibt sich direkt aus Gleichung (6.1)

$$N_o(s) + k \cdot Z_o(s) = 0 \Rightarrow k \cdot \frac{Z_o(s)}{N_o(s)} = -1.$$

Über eine kurze Umformung erhält man

$$|k| = \left| \frac{N_o(s)}{Z_o(s)} \right|. \quad (6.7)$$

Mit Hilfe von Gleichung (6.7) kann nun auch die benötigte Regelverstärkung eingestellt werden.

MATLAB bietet die Möglichkeit, die Lage der Pole in Abhängigkeit vom Reglerparameter  $k$  anzeigen zu lassen. Diese Funktion lautet `rlocus(G)`, wobei  $G$  die Übertragungsfunktion des *offenen Regelkreises* ohne den Reglerparameter ist.

Für obiges Beispiel würde der MATLAB-Code wie folgt aussehen.

---

```
>> G = tf([1], [1 1 0]);  
>> rlocus(G)
```

Um die Verstärkung in einem gewünschten Punkt zu betrachten, kann man im *Figure-Fenster* mit Hilfe des *Data Cursors* den gewünschten Punkt anklicken.

---

### 6.1.3 Weiterführende Details

---

In diesem Abschnitt wird besonders auf das Verhalten der WOK in Bezug auf Null- und Polstellen eingegangen.

Dies soll wieder an der Beispielstrecke

$$G_s(s) = \frac{1}{s \cdot (s + 1)}$$

verdeutlicht werden. Diese Strecke besitzt zwei Pole. Mit einem P-Regler ergibt sich die WOK in Abbildung 6.2a (die auch schon in Abbildung 6.1 dargestellt ist.)

Es werden nun die Verläufe der Wurzelortskurven betrachtet, wenn zusätzliche kritische Stellen hinzugefügt werden. Wird ein Pol bei  $s = -2$  hinzugefügt erhält man die WOK in Abbildung 6.2b. Fügt man dem ursprünglichen System eine Nullstelle bei  $s = -2$  hinzu, ergibt sich die WOK in Abbildung 6.2c.

Wie man anhand der Bilder sehen kann, verformt die zusätzliche kritische Stelle den Verlauf der WOK. Ein Pol (Abbildung 6.2b) stößt die Wurzeläste ab. Eine Nullstelle (Abbildung 6.2c) zieht die Wurzeläste an.

Diese Tatsache kann man sich beim Reglerentwurf zunutze machen. Speziell instabile Strecken können durch geschickte Wahl von Reglernullstellen und mit einer entsprechenden Reglerverstärkung  $k$  stabilisiert werden. Für den praktischen Entwurf bietet MATLAB das *SISO Design Tool* an (SISO = Single-Input-Single-Output). Dieses wird im Folgenden näher betrachtet.

---

## 6.2 Das SISO-Tool in MATLAB

---

Die Beschreibung des SISO-Tools (*SISO Design Tool*) bezieht sich auf die Versionen bis MATLAB 2015b.

---

### 6.2.1 Grundlagen des SISO-Tools

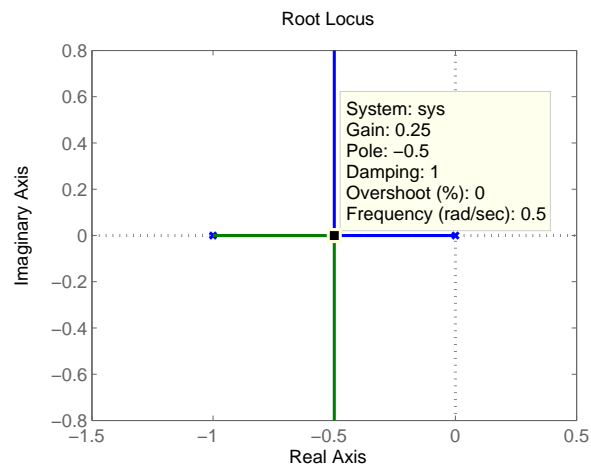
---

Der Befehl `sisotool` öffnet das SISO-Tool in MATLAB. Dabei öffnen sich zwei Fenster (Abbildung 6.3):

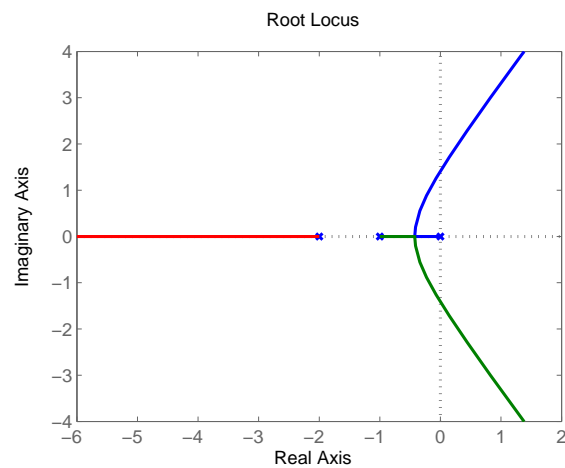
- Control and Estimation Tools Manager
- SISO Design for SISO Design Task

Im *Control and Estimation Tools Manager* Fenster können im Reiter *Architecture* über die Schaltfläche *System Data* Übertragungsfunktionen eingelesen werden, wobei die Bezeichnungen

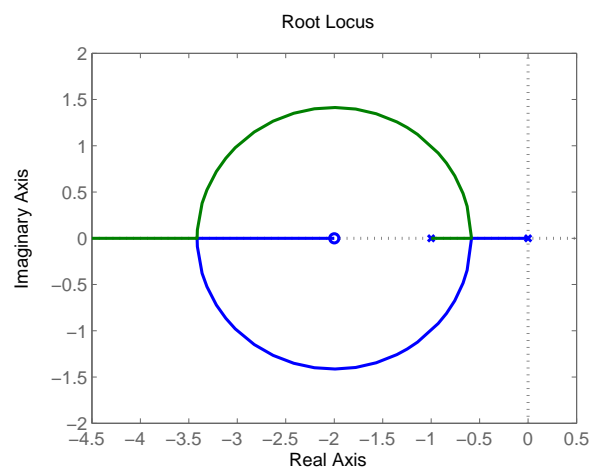
- G (Plant): Strecke
- H (Sensor): Messglied
- F (Prefilter): Vorfilter
- C (Compensator): Regler



(a) WOK von  $F_o = \frac{1}{s \cdot (s+1)}$



(b) WOK von  $F_o(s) = \frac{1}{s^3 + 3s^2 + 2s}$



(c) WOK von  $F_o(s) = \frac{s+2}{s^2 + s}$

Abbildung 6.2.: Einfluss von Polen und Nullstellen auf die WOK

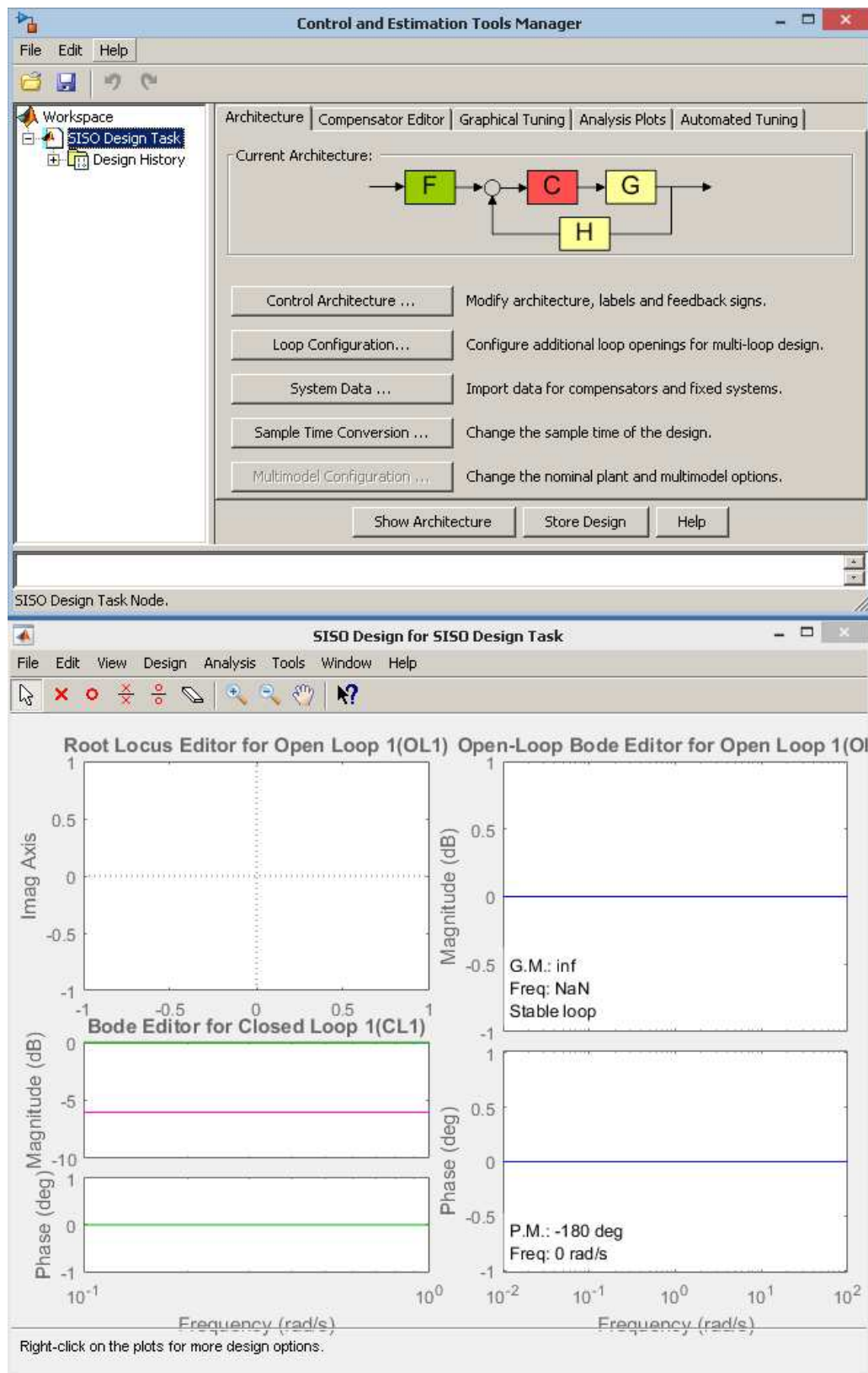


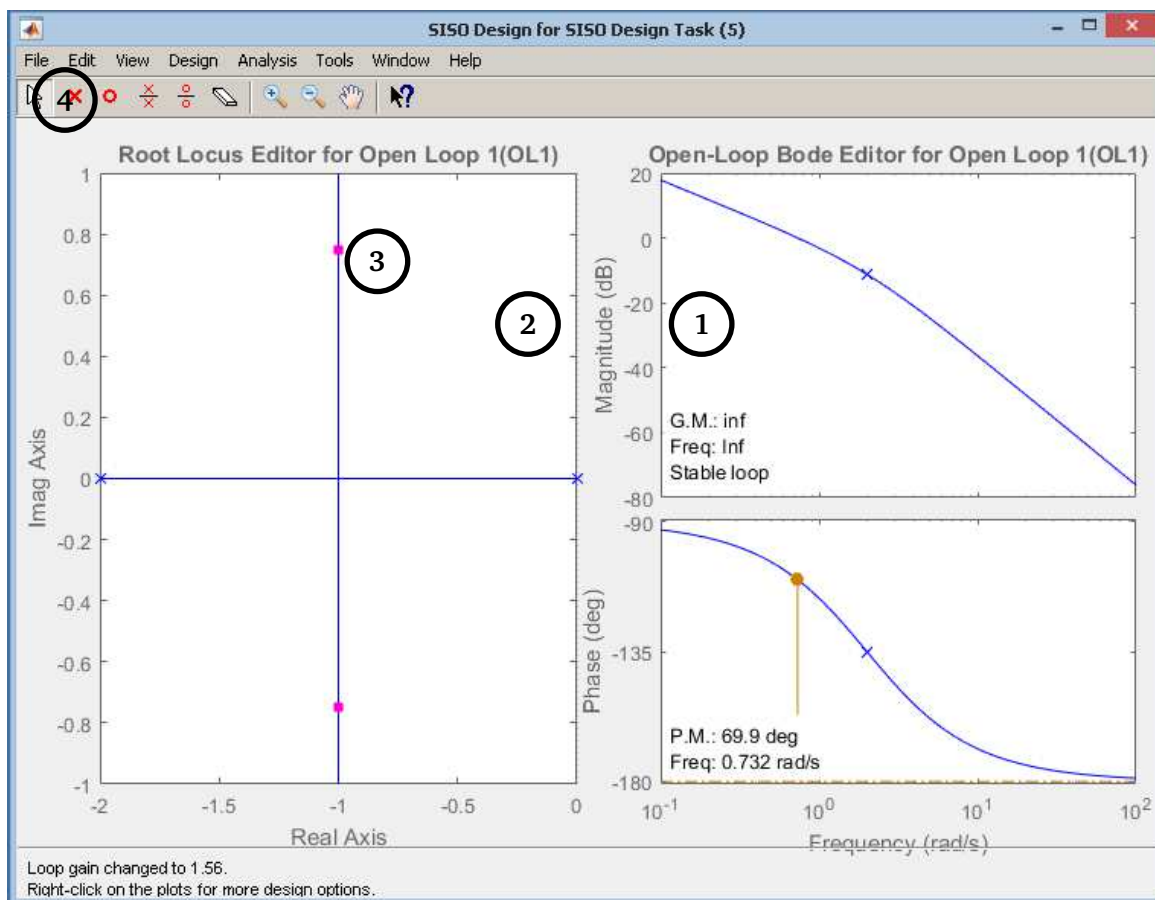
Abbildung 6.3.: Startbildschirm SISO-Tool

den Modellblöcken der gezeigten *Current Architecture* entsprechen. Wenn im Workspace die Variable

```
>> G = tf([1], [1 1 0]);
```

mit der Übertragungsfunktion des Beispielsystems angelegt ist, kann diese jetzt hier direkt angegeben werden. Über *Browse* können die Übertragungsfunktionen auch aus einer mat-File ausgewählt werden.

Wenn die Übertragungsfunktion G ausgewählt wurde, sollte sich die Anzeige wie in Abbildung 6.4 ergeben, wobei hier noch das Bodediagramm des geschlossenen Kreises (in Abbildung 6.3 links unten) ausgeblendet wurde. (Zum Ausblenden im Reiter *Graphical Tuning* den *Plot Type* des dritten Plots auf „None“ setzen.) Es wird sowohl die WOK (2) als auch das Bodediagramm des offenen Regelkreises (1) dargestellt.



**Abbildung 6.4.:** SISO-Tool mit Standardbeispiel

Auf der WOK lassen sich zwei rosafarbene Vierecke (3) erkennen. Diese zeigen die Lage der Pole des geschlossenen Regelkreises. Durch Anklicken und Verschieben der Vierecke kann man den Verstärkungsfaktor des Reglers verändern. Dies gibt dem Benutzer die Möglichkeit, den Verstärkungsfaktor in wünschenswerten Polkonfigurationen zu ermitteln, ohne aufwendige Rechnungen zu machen.

Um die Achsen anders zu skalieren, macht man einen Rechtsklick im WOK-Fenster und verändert unter *Limits* die entsprechenden Einstellungen.

Unterhalb der Menüleiste (4) befinden sich verschiedene Schaltflächen, mit deren Hilfe man dem Regler zusätzliche Pole und Nullstellen hinzufügen kann. Die Symbole der Schaltflächen bedeuten:

- Einfacher Pol : ×

- Einfache Nullstelle :  $\circ$
- Konjugiert komplexer Pol :  $\frac{\times}{\times}$
- Konjugiert komplexe Nullstellen :  $\frac{\circ}{\circ}$
- Löschen von Polen/Nullstellen

Durch Anklicken und Platzieren im WOK Fenster bestimmt man die Lage der zusätzlichen kritischen Stellen. Hinzugefügte kritische Stellen können mit der Maus verschoben werden.

---

### 6.2.2 Besondere Funktionen des SISO-Tools

---

In diesem Abschnitt wird noch auf einige besondere Möglichkeiten des SISO-Tools eingegangen, die die Systemanalyse und den Reglerentwurf erleichtern.

Das SISO-Tool bietet die Möglichkeit, für den eingestellten Regelkreis verschiedene Analysen durchzuführen. Auf der rechten Seite in Abbildung 6.4, im Bereich (1), wird das Bodediagramm des offenen Regelkreises angezeigt. In dem Diagramm wird die Phasenreserve ockerfarben angezeigt. Weitere Analysen können über das Menü *Analyse* geöffnet werden, z. B.

- die Sprungantwort des geschlossenen Regelkreises von Führungsgröße auf Regelgröße (*Closed Loop r to y*) und
- die Sprungantwort des geschlossenen Regelkreises von Führungsgröße auf Stellgröße (*Closes Loop r to u*).

Für das Beispielsystem aus Abschnitt 6.2.1 kann man sich so beispielsweise die Sprungantwort des geschlossenen Regelkreises anzeigen lassen. Über einen Rechtsklick im *LTI Viewer*-Fenster kann man sich dann im Kontextmenü verschiedene charakteristische Werte anzeigen lassen, z. B. die Anstiegszeit oder die Ausregelzeit, wie in Abbildung 6.5 zu sehen. (Zum Anzeigen dieser Werte darf mit der rechten Maustaste nicht auf die Kurve selber, sondern es muss in den weißen Plotbereich geklickt werden. Dann können die Werte über *Characteristics* ausgewählt werden.)

Eine Regelung hat immer das Ziel, dass der geschlossene Regelkreis verschiedene Anforderungen an das System – zumindest annähernd – erfüllt. Damit der geschlossene Regelkreis diese Anforderungen erfüllt, müssen die Pole des geschlossenen Regelkreises in einem bestimmten Gebiet in der *s*-Ebene liegen. Das SISO-Tool bietet die Möglichkeit, verschiedene Anforderungen einzustellen, damit der gewünschte Bereich in der *s*-Ebene angezeigt wird. Um die Anforderungen einzustellen, macht man einen Rechtsklick im WOK-Fenster und wählt das Menü *Design Constraints* bzw. *Design Requirements* aus. Dort kann dann die

- die Ausregelzeit (Settletime) und
- das prozentuale Überschwingen (Percent overshoot)

eingestellt werden. Es gibt noch weitere mögliche Einstellungen, die an dieser Stelle aber nicht relevant sind.

---

## 6.3 Analyse von Parameteränderungen mit Hilfe der WOK

---

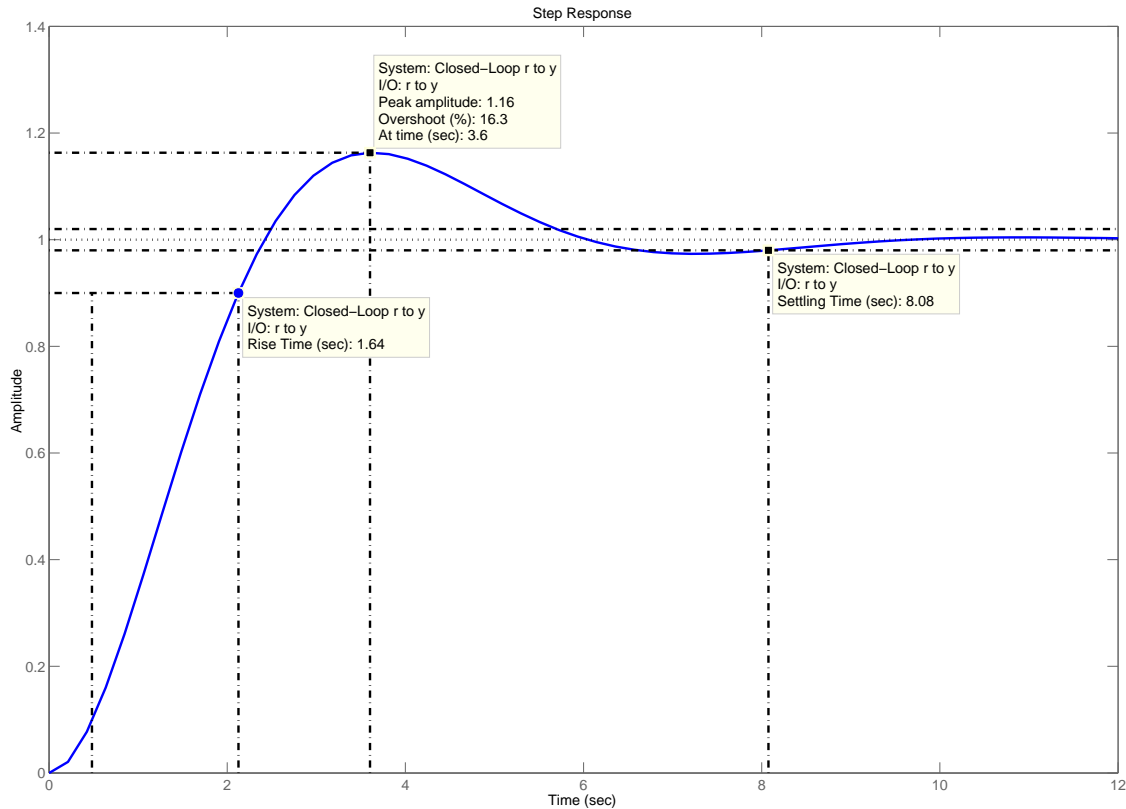


---

### 6.3.1 Allgemeine Betrachtung von Parameterschwankungen

---

Mit Hilfe der WOK kann man bei einem vorgegebenen Regler das Verhalten des geschlossenen Systems in Abhängigkeit von einem Streckenparameter (statt der Reglerverstärkung) analysieren. Es kann so die Robustheit eines Systems analysiert werden. Dazu muss man den Nenner des geschlossenen



**Abbildung 6.5.:** Sprungantwort des Beispielsystems mit eingetragenen charakteristischen Werten

Kreises nur auf die Form der charakteristische Gleichung (6.1) bringen. Dies wird im Weiteren an einem kleinen Beispiel erläutert.

Als Strecke sei das  $PT_2$ -System

$$G_s(s) = \frac{1}{(T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1)} \quad (6.8)$$

gegeben. Die Pole der Strecke liegen bei  $s_1 = \frac{1}{-T_1}$  und  $s_2 = \frac{1}{-T_2}$ .

Als Regler soll ein P-Regler

$$G_r(s) = k_p \quad (6.9)$$

verwendet werden, wobei vorausgesetzt wird, dass  $k_p$  fest gewählt ist und *nicht* mehr verändert werden kann.

Damit ergibt sich für den offenen Kreis die Übertragungsfunktion

$$F_o(s) = G_s(s) \cdot G_r(s) = \frac{k_p}{(T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1)} \quad (6.10)$$

Der geschlossene Regelkreis hat damit

$$F_w(s) = \frac{F_o(s)}{1 + F_o(s)} = \frac{k_p}{(T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1) + k_p} \quad (6.11)$$

als Übertragungsfunktion mit dem charakteristischen Polynom

$$(T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1) + k_p . \quad (6.12)$$

Es wird nun angenommen, dass sich der Streckenparameter  $T_1$ , z. B. aufgrund von Temperaturschwankungen, verändert. Ziel ist es jetzt, die Lage der Pole des geschlossenen Regelkreises in Abhängigkeit des Parameters  $T_1$  anzugeben.

Dazu wird das charakteristische Polynom so umgeformt, dass die Form der charakteristischen Gleichung (6.1) in Abhängigkeit von  $T_1$  erhalten wird. D. h.  $T_1$  soll die Rolle des  $k$  in Gl. (6.1) einnehmen. Man erhält also

$$T_1 \cdot (T_2 \cdot s^2 + s) + (k_p + 1) + T_2 \cdot s . \quad (6.13)$$

Somit gilt

$$N_o(s) = (k_p + 1) + T_2 s ,$$

$$Z_o(s) = T_2 \cdot s^2 + s .$$

Formal kann man also die WOK-Betrachtungen an der Strecke

$$G_o(s) = \frac{Z_o(s)}{N_o(s)}$$

mit dem P-Regler, dessen Verstärkung dem Streckenparameter  $T_1$  entspricht,

$$k = T_1 ,$$

durchführen. Mit Hilfe von  $N_o(s)$  und  $Z_o(s)$  kann damit die Übertragungsfunktion des offenen Kreises in MATLAB eingegeben und im SISO-Tool in Abhängigkeit von  $T_1$  analysiert werden.

Mit den festen Regler- und Streckenparametern

$$k_p = 1 ,$$

$$T_2 = 1$$

erhält man die WOK in Abbildung 6.6.

Aus dieser lässt sich in diesem Fall ablesen, dass der Regelkreis mit den gegebenen festen Parametern  $k_p$  und  $T_2$  für beliebige positiven Werte für  $T_1$  stabil ist.

---

### 6.3.2 Betrachtung des Arbeitspunkt winkels als Parameter

---

Die allgemeine Übertragungsfunktion des linearisierten Pendelschraubers hat für feste  $\alpha_s$  die Form

$$G(s) = G_{\text{Ventilator}}(s) \cdot G_{\text{Hebel}}(s)$$

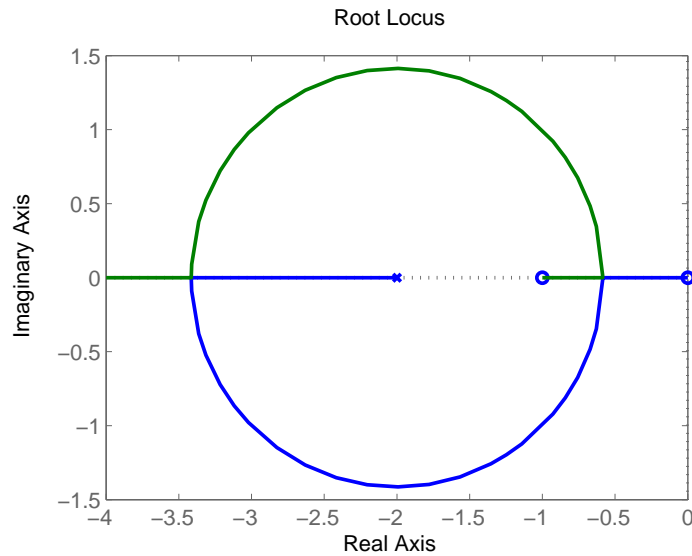
mit

$$G_{\text{Hebel}}(s) = \frac{2\gamma \omega_s H_R}{J_W s^2 + c_W s - k \sin \alpha_s} , \quad (6.14)$$

$$G_{\text{Ventilator}}(s) = \frac{c_M}{L J_P s^2 + (R J_P + 2\beta \omega_s L) \cdot s + (2\beta \omega_s R + c_M^2)} \quad \text{und} \quad (6.15)$$

$$\omega_s = \sqrt{\frac{k}{H_R \gamma} \cdot \cos \alpha_s} . \quad (6.16)$$





**Abbildung 6.6.:** WOK des Beispielsystems in Abhängigkeit von  $T_1$

Die Werte der Konstanten können Tabelle A.1 auf Seite 38 entnommen werden.

In diesem Versuch soll die WOK in Abhängigkeit des Parameters  $\alpha_s$  untersucht werden. Für den Parameter  $\alpha_s$  sind die Gleichungen (6.14) und (6.16) auf Grund des Sinus- bzw. Cosinus-Terms nichtlinear. Die WOK kann nur auf lineare Systeme angewandt werden, deshalb werden die nichtlinearen Terme durch

$$\cos \alpha_s \approx 1$$

$$\sin \alpha_s \approx \alpha_s$$

approximiert. Diese Approximation ist für kleine Winkel  $\alpha_s$  ausreichend.



---

# Literaturverzeichnis

- [1] **Prof. Dr.-Ing. J. Adamy**  
Systemdynamik und Regelungstechnik II  
TU-Darmstadt: Institut für Automatisierungstechnik und Mechatronik, 2012
- [2] **Prof. Dr.-Ing. U. Konigorski**  
Systemdynamik und Regelungstechnik I  
TU-Darmstadt: Institut für Automatisierungstechnik und Mechatronik, WS 2012/2013
- [3] **W. D. Pietruszka**  
MATLAB und Simulink in der Ingenieurspraxis; Modellbildung, Berechnung und Simulation  
Wiesbaden: Teubner, 2. Aufl. 2006
- [4] **Stoer, J. und Burlisch, R.**  
Numerische Mathematik 2  
Springer, 2005