Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CAD	computer-aided design (engl. für rechnergestützte Konstruktion)

1 Übersicht

Im einleitenden Kapitel werden die Motivation und die Aufgabenstellung der Bachelorarbeit besprochen. Gleichzeitig erfolgt ein kurzer Überblick zu den einzelnen Kapiteln dieses Berichts.

1.1 Motivation

Das Fahrwerk eines Rennwagens ist die Verbindung zwischen dem Reifen und der Karosserie und definiert die Bewegung des Reifens relativ zur Karosserie beim Federn bzw. Lenken. Oberstes Ziel bei der Konstruktion eines Rennwagens ist immer die Reduktion der Massen, speziell der der ungefederten Komponenten, um die wirkenden Kräfte in extremen Fahrsituationen zu verringern, was die Agilität des Fahrzeugs steigert. Die Fahrwerke der bisher an der Hochschule Mittweida gebauten Rennwagen wurden nur nach statischen Grundlagen dimensioniert. Nun liegt das Interesse darin, eine Grundlage für dynamische Betrachtung zu legen, um bei zukünftigen Konstruktionen eventuell noch mehr Masse reduzieren zu können. Des Weiteren besteht Interesse darin, herauszufinden, ob numerische Differentiationsverfahren für die dynamische Simulation von komplexeren, ungleichförmig übersetzten Getrieben, welche in hohem Grad nichtlinear sind, nutzbar sind.

1.2 Zielsetzung

Die vorliegende Arbeit befasst sich mit der Untersuchung zu dynamischen Analyseverfahren eines bereits realisierten Fahrwerks eines Rennwagens, welcher im Rahmen des Formula Student Wettbewerbs in der Saison 2010/2011 an der Hochschule Mittweida konstruiert und gebaut wurde.

Zur Untersuchung wird die Software MATLAB® [1] genutzt. In dieser Programmierumgebung werden alle Berechnungen vollzogen. Dabei sollen die Algorithmen im Mittelpunkt der Betrachtungen stehen. Die Ergebnisse werden am Ende anschaulich aufbereitet.

Hauptziel ist die Erstellung eines Analyseverfahrens für die Dynamik des Rennwagens, um damit später verschiedene Untersuchungen betreffend der dynamisch wirkenden Kräfte im Rennbetrieb vollziehen zu können.

---

[1] MATLAB® Eingetragenes Warenzeichen von ''The MathWorks, Inc''

1.3 Problembeschreibung und Formulierung der Aufgabenstellung

Problembeschreibung:

Ein konstruktiv bereits realisiertes Fahrwerk eines Rennwagens soll kinetisch untersucht werden. Dazu soll ein geeignetes Analyseverfahren ausgearbeitet werden. Die Ergebnisse sollen als Ansatz für dynamische Untersuchungen dienen.

Aufgabenstellung:

1. Vollständige Abbildung der Kinematik des Fahrzeugs in MATLAB®
2. Konzipierung der Theorie des Analyseverfahrens
3. Implementierung der Theorie in der Software (MATLAB®) mit Test an geeignetem Beispiel und Verallgemeinerung der Lösungsalgorithmen
4. Untersuchung des Komplexbeispiels “Rennwagen“
5. Auswertung der Ergebnisse

1.4 Kapitelübersicht

Die Bachelorarbeit besteht aus 7 Kapiteln.

Nach der Übersicht werden in **Kapitel 2** die Grundlagen für die Auseinandersetzung mit der Thematik dieser Bachelorarbeit geschaffen. Es wird gezeigt, warum die dynamischen Berechnungen von Bedeutung sind. Auch die Vorteile der Programmierumgebung MATLAB® als Entwicklungstool werden umrissen.

Anschließend folgt in **Kapitel 3** ausführlich die Herangehensweise an die Lösung der Kinematik. Auch die Implementierung in MATLAB® wird erklärt.

In **Kapitel 4** wird die grundlegende Theorie zur Untersuchung dynamisch belasteter Systeme besprochen.

Kapitel 5 behandelt die Umsetzung der Theorie aus Kapitel 4 in MATLAB®. Dabei wird Augenmerk auf die Gewährleistung, das Programm universell einsetzen zu können, gelegt.

Die Anwendung am Komplexbeispiel “Rennwagen“ des im Kapitel 5 entstandenen Grundprogramms wird in **Kapitel 6** gezeigt.

Die Auswertung der Ergebnisse und ein Ausblick werden im letzten Kapitel, dem **Kapitel 7**, behandelt.

2 Einführende Betrachtungen

In diesem Kapitel werden die Grundsteine für die dynamische Berechnung des Fahrwerks gelegt. Als erstes wird ein Einblick in die Bedeutung und die Systematik von dynamischen Berechnungen gegeben. Daran schließt sich eine Betrachtung der genutzten Berechnungssoftware MATLAB® an. Als Letztes folgt die ausführliche Veranschaulichung der Berechnung der zu Grunde liegenden Kinematik des Fahrwerks.

2.1 Dynamische Berechnungen

Um dynamische Festigkeitsnachweise für die Fahrwerkskomponenten erbringen zu können, müssen die Beanspruchungs-Zeit-Funktionen der belasteten Bauteile bekannt sein. *Dynamische Beanspruchungen lassen sich in stationär und instationär aufteilen. Instationäre Schwingungsvorgänge treten z. B. immer beim An- und Abfahren von Anlagen auf. Die stationären dynamischen Beanspruchungen lassen sich nochmals in gleichförmige und ungleichförmige Beanspruchungsfolgen untergliedern.*[2] Da beim Fahrzeug im Rennbetrieb die Beanspruchungen hauptsächlich von der Streckenführung, der Witterung und vom Fahrer selbst abhängen und keine instationären Beanspruchungen auftreten, ordnet sich die zeitliche Beanspruchung der Fahrwerkskomponenten in die dynamisch- stationäre- ungleichmäßige Beanspruchungsfolge ein. Um diese dynamischen Beanspruchungen schon in der Konstruktionsphase berücksichtigen zu können, dient das kinetische Modell des Rennwagens als Ausgangspunkt für dynamische Simulationen extremer Fahrsituationen.

Ziel ist es, die technische Struktur hinsichtlich ihres dynamischen Verhaltens zu beschreiben. Die beherrschende dynamische Gleichung wird gesucht.[3] Dabei hilft der Lagrange-Formalismus:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = 0 \qquad [1]$$

mit $$L = E_{kin} - E_{pot} \qquad [2]$$

q_i und $\dot{q}_i$ sind die generalisierten Koordinaten bzw. deren Ableitungen nach der Zeit.

---

[2] Schlecht, Berthold: Maschinenelemente 1, Pearson Studium, München, 1. Auflage, 2007, S. 29

[3] Zimmermann, Martin: Vorlesung Maschinendynamik an der Hochschule Mittweida, Mittweida, 2010, Vorlesung 1/ Folie 4

Die Differenz von kinetischer- und potentieller Energie ergibt die Lagrangefunktion L. Nach Durchführung der Ableitungen erhält man die Bewegungsgleichung/-en für sein System. *Mit diesen Differentialgleichungen lässt sich die Entwicklung des Systems, unter Einwirkung äußerer Einflüsse, vollständig räumlich und zeitlich beschreiben.*[4] Somit ist es möglich, die Kinetik des Fahrzeugs vollständig abzubilden.

2.2 Was ist MATLAB®?

Dieses Programmsystem ist ein Werkzeug zur numerischen Bearbeitung von einfachen bis hin zu komplexen technischen Systemen. Es ist zur Analyse und Synthese dynamischer Vorgänge geeignet [5].

Die Quelltexte in dieser Programmierhochsprache werden in Script- und Function-Files unterschieden. In Script-Files können Daten abgerufen oder abgelegt werden. *Diese Variablen sind nach Programmablauf verfügbar, d. h. die Daten sind global* [6]. In der Unterfunktionsstruktur der Function-Files werden abhängig von Eingangsvariablen und einer Funktion Ausgangsdaten berechnet, die dann der übergeordneten Funktion zur Verfügung stehen. *Der Kern dieser Hochsprache besteht aus einer Vielzahl von Built-In Functions, die für Berechnungen im Bereich der linearen Algebra, der Datenanalyse sowie der Lösung von Differentialgleichungen bestimmt sind. Die Built-In Functions sind für Vektor- und Matrizenoperationen bezüglich der Rechenzeit optimiert.*[7] Das ist das entscheidende Kriterium zur Wahl dieser Software zur Berechnung dynamischer Probleme.

[4] http://de.wikipedia.org/wiki/Bewegungsgleichung, verfügbar am 05.08.2011, 15:00 Uhr

[5] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, B.G. Teubner Verlag, Wiesbaden, 2.Auflage, 2006, S.1

[6] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, B.G. Teubner Verlag, Wiesbaden, 2.Auflage, 2006, S.29

[7] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, B.G. Teubner Verlag, Wiesbaden, 2.Auflage, 2006, S.1

3 Kinematik

Die Kinematik der Doppelquerlenkeraufhängung legt den Grundstein für die dynamischen Berechnungen am Rennwagen. Sie wird ebenfalls vollständig in MATLAB® gelöst.

3.1 Definition

Die Kinematik ist die Lehre von der geometrischen und analytischen Beschreibung der Bewegungszustände von Punkten und Körpern. Sie berücksichtigt nicht die Kräfte und Momente als Ursachen der Bewegung.[8]

3.2 Die Doppelquerlenkeraufhängung

Zum Grundverständnis der Doppelquerlenkeraufhängung werden im Folgenden nun die einzelnen Komponenten definiert und auf deren Wechselwirkungen eingegangen. Außerdem werden die Fachbegriffe erklärt, die zum Verständnis der Bachelorarbeit beitragen.

3.2.1 Komponenten der Doppelquerlenkeraufhängung

Ein Querlenker (Nr. 1 und 3 Abbildung 1) ist eine V-förmige Komponente, bei der die offenen Enden gelenkig mit der Karosserie verbunden sind. Das Ende an der spitzen Seite ist gelenkig mit dem Radträger verbunden. Bei der Doppelquerlenkeraufhängung werden pro Radaufhängung zwei Querlenker verbaut. Der eine (Nr. 1 Abbildung 1) oberhalb der Rotationsachse des Rades und der andere (Nr. 3 Abbildung 1) unterhalb derer. Im Idealfall werden in den Stäben nur Zug und Druckkräfte übertragen.

Der Radträger (Nr. 7 Abbildung 1) verbindet die spitzen Enden der beiden Querlenker. Des Weiteren ist am Radträger die Radnabe angebracht, die der Befestigung des Rades (Nr. 8 Abbildung 1) dient und dafür sorgt, dass das Rad um seine Rotationsachse drehbar gelagert ist. Außerdem ist der Radträger die Aufnahme des Bremssattels und der Spurstange.

Die Spurstange (Nr. 4 Abbildung 1) ist ein Stab mit gelenkigen Enden. Ein Ende ist am Radträger befestigt und das andere an der Karosserie bzw. am Lenkgetriebe (Vorderach

[8] W. Beitz und K.-H. Grote: DUBBEL – Taschenbuch für den Maschinenbau, Springer- Verlag, Berlin, 19. Auflage, 1997, S.B 17

se). Diese sorgt dafür, dass der Freiheitsgrad der Rotation um die Hochachse des Radträgers gebunden wird.

Die im Beispiel aus drei Komponenten bestehende Federung setzt sich aus der Feder-/Dämpfereinheit (Nr. 2 Abbildung 1), dem Umlenkhebel (Nr. 5 Abbildung 1) und der Zugstange (Nr. 6 Abbildung 1) zusammen. Die Feder-/Dämpfereinheit ist an einer Seite mit der Karosserie, und an der anderen mit dem Umlenkhebel drehbar verbunden. Der Umlenkhebel bildet ein Dreieck, dessen spitze Ecke drehbar an der Karosserie befestigt ist. Die Zugstange bildet die gelenkige Verbindung zwischen der Spitze des oberen Querlenkers und der freien Spitze des Umlenkhebels. Der Umlenkhebel sorgt in dieser Konstellation für eine definierte Federungskennlinie.

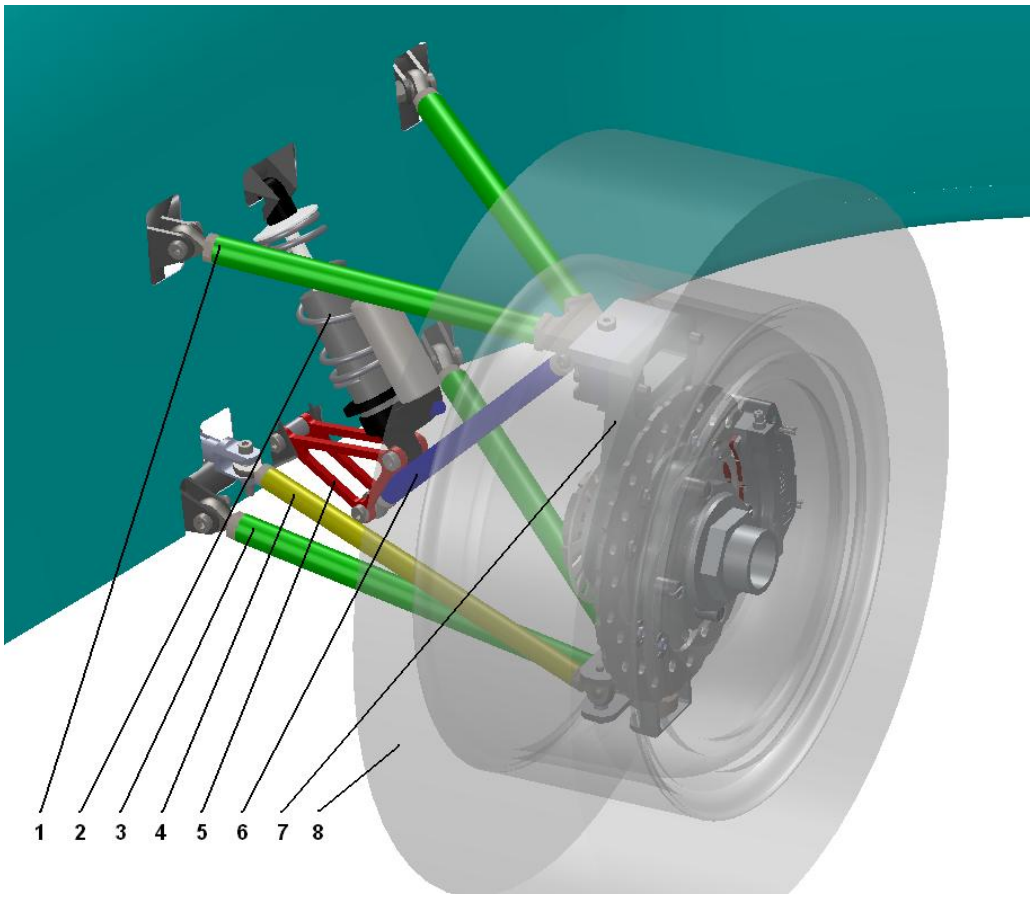


Abbildung 1: Darstellung des Fahrwerks vorn links

3.2.2 Funktionsprinzip

Um die Funktionsweise darzustellen, wird als erstes die Aufhängung vorn links in Abbildung 2 als Getriebe versinnbildlicht. Das Getriebe wird zunächst eben betrachtet. Es handelt sich um ein sechsgliedriges Getriebe. Die Festlager am Getriebeglied 1 (Abbildung 2) stellen die Karosserie dar. Getriebeglied 2 steht für den oberen, Glied 4 für den unteren Querlenker. Das Getriebeglied 3 stellt den Radträger mit Rad und Nabe dar. Die Gelenke

A bis H sind Drehgelenke. In folgender Berechnung wird mit der Zwanglaufgleichung nachgewiesen, dass die beweglichen Gelenke B, C, F und G in ihrer Lage nur von der Länge der Feder abhängig sind.

F... Anzahl der Freiheitsgrade

$z = 6$... Anzahl der Getriebeglieder

$g = 7$... Anzahl der Gelenke (Gelenk E wird außen vorgelassen, da Feder flexibel ist.)

$F = 3*(z-1) - 2*g$ $\quad F = 3*(6-1) - 2*7 \quad F = 1$

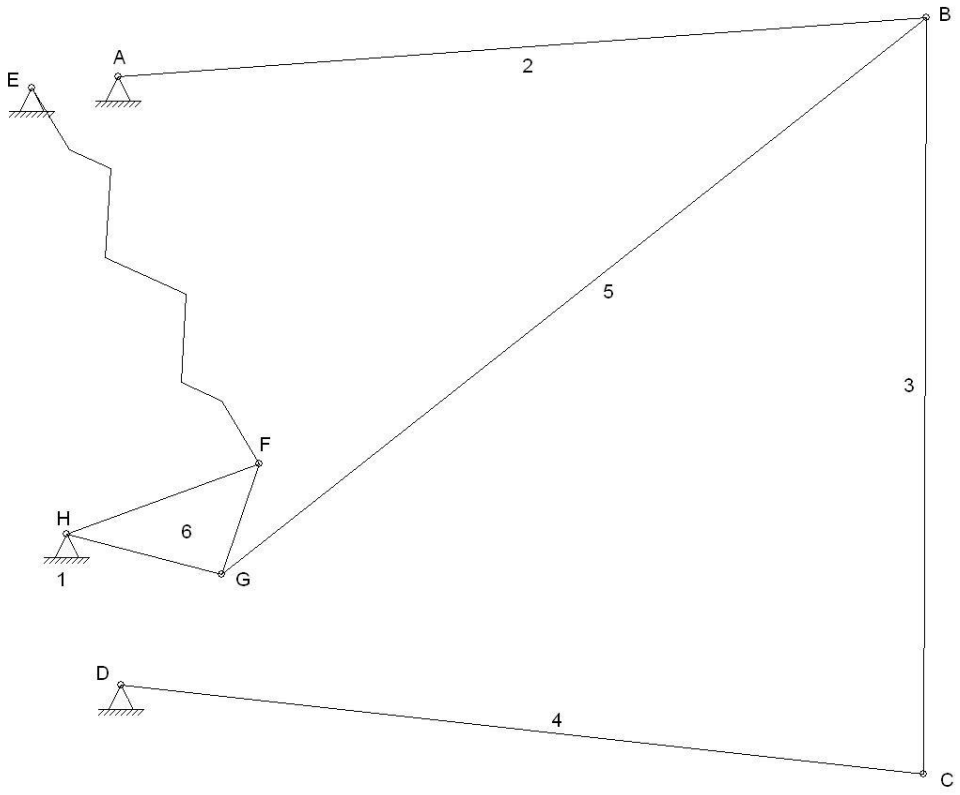


Abbildung 2: Getriebedarstellung der Aufhängung

Die Zwangläufigkeit gilt ebenso für das räumliche Getriebe jener Komponenten. Die Anzahl der Getriebeglieder und Gelenke sind gleich der im ebenen Problem. Lediglich entsteht an jedem Querlenker ein neuer Punkt an der Karosserie. Da die Spitzen der Querlenker steif sind, entsteht hier kein neues Getriebeglied. Die Achse, die durch die beiden offenen Enden der Querlenker verläuft, bildet die neue Drehachse, um die sich der Punkt in der Spitze drehen kann. Obwohl konstruktiv zwei Gelenke mehr als im ebenen Problem vorhanden sind, kann sich der Querlenker nur um diese neue Drehachse bewegen. Es ist nur eine Rotation möglich (Scharniergelenk).

Um die Möglichkeit der Rotation des Radträgers (Getriebeglied 3) um die Achse B-C zu binden, existiert im räumlichen Getriebe zusätzlich die Spurstange, welche den Radträger zum Lenkgetriebe (Vorderachse) bzw. zur Karosserie (Hinterachse) abstützt.

Zusammenfassend gilt, dass die beweglichen Punkte der Aufhängung nur von der Länge der Feder (bzw. Federweg des Rades) und der Lenkradstellung abhängig sind (Hinterachsaufhängung ist nur von Federweg abhängig).

3.2.3 Vereinbarungen

Zur kinematischen Beschreibung des Fahrwerks werden zunächst folgende Vereinbarungen getroffen:

Da hier von einem real existierenden Fahrwerk ausgegangen wird, werden auch dessen CAD[9] Daten als Grundlage verwendet. Somit liegt der Ursprung des dreidimensionalen kartesischen Koordinatensystems zwischen den Radaufstandspunkten der Vorderachse auf Fahrbahnniveau.

Die Y-Achse steht somit normal auf der Fahrbahnebene, die X-Achse verläuft parallel zur Längsachse des Fahrzeugs auf der Fahrbahnebene mit positiver Richtung zum hinteren Teil des Fahrzeugs. Die Z-Achse liegt ebenfalls auf der Fahrbahnebene mit positiver Richtung zur linken Seite des Fahrzeugs. X, Y, Z bilden ein Rechtssystem (siehe Abbildung 3).

Die Koordinaten der Punkte A bis N (Tabelle 1) sind identisch mit denen des real existierenden Fahrwerks. Auch die konstanten Beträge zwischen den Punkten sind damit definiert. Die Abbildung 3 zeigt nun die Skizze der Doppelquerlenkeraufhängung vorn links, welche mit der CAD-Software SolidWorks[10] erstellt wurde. Abbildung 3 zeigt die Situation, in der das Fahrzeug mit einem 75 kg schweren Fahrer belastet ist und 0° Lenkeinschlagwinkel hat. Diese Situation wird im Folgenden als Ausgangslage bezeichnet.

Nun wird nur auf diese Aufhängung eingegangen. Die darin definierten Punkte im Kartesischen Koordinatensystem werden in der Tabelle 1aufgeführt und erklärt.

[9] CAD: computer-aided design (rechnergestützte Konstruktion)

[10] SolidWorks®: Eingetragenes Warenzeichen von ''Dassault Systèms, DS''

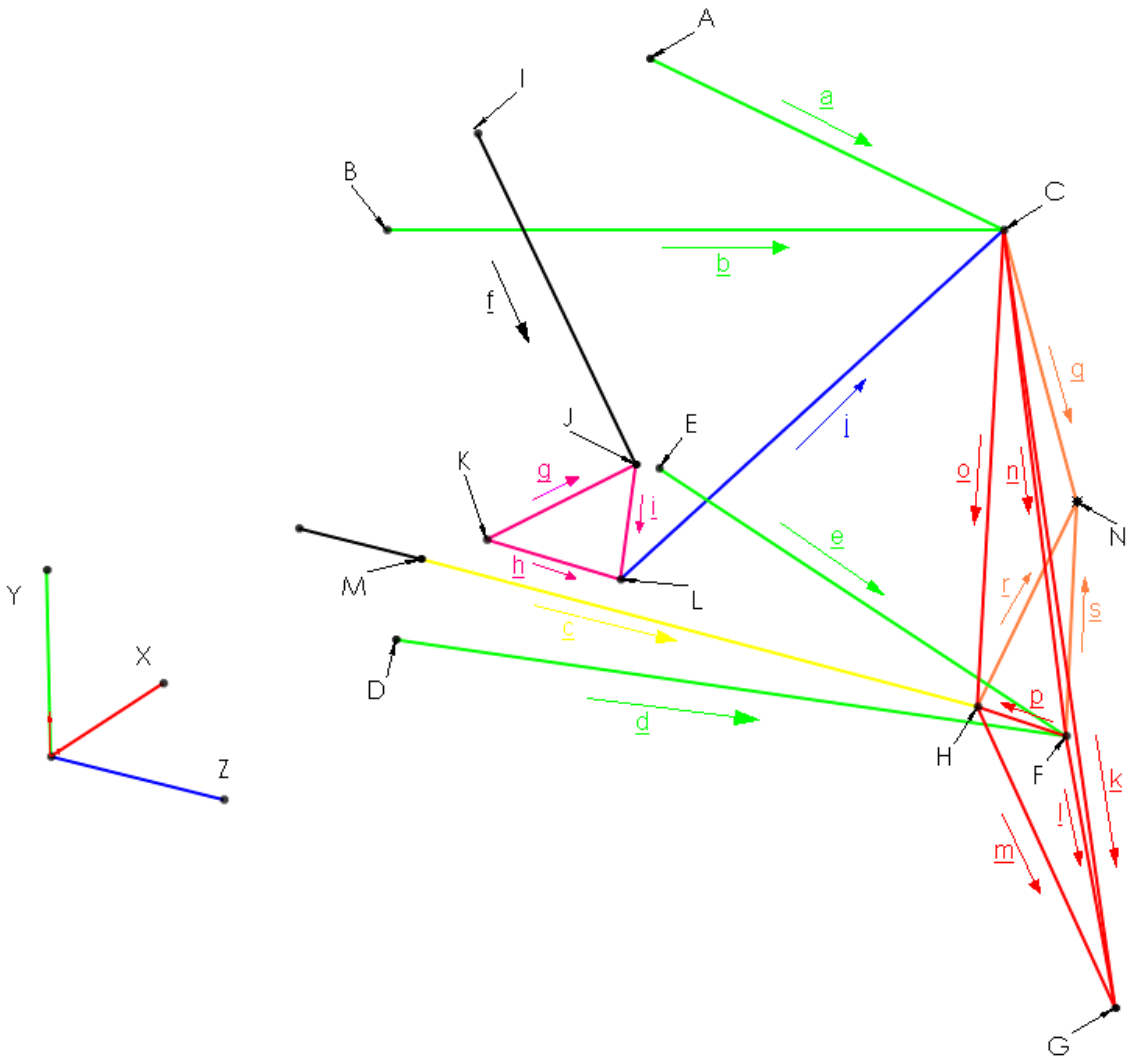


Abbildung 3: Skizze der Doppelquerlenkeraufhängung vorn links

Punkt	Lagerart	Funktion	Freiheitsgrade
A	Festlager	Verbindung Querlenker-Karosserie	3 Rotationen
B	Festleger	Verbindung Querlenker-Karosserie	3 Rotationen
C	Gelenklager	Verbindung Querlenker-Radträger-Zugstange	3 Rotationen
D	Festlager	Verbindung Querlenker-Karosserie	3 Rotationen
E	Festlager	Verbindung Querlenker-Karosserie	3 Rotationen
F	Gelenklager	Verbindung Querlenker-Radträger	3 Rotationen
G	-	Radaufstandspunkt	-
H	Gelenklager	Aufnahme Spurstange	3 Rotationen
I	Festlager	Verbindung Feder/Dämpfer-Karosserie	1 Rotation
J	Gelenklager	Verbindung Feder/Dämpfer-Umlenkhebel	1 Rotation
K	Festlager	Verbindung Umlenkblech-Karosserie	1 Rotation
L	Gelenklager	Verbindung Umlenkhebel-Zugstange	1 Rotation
M	Loslager	Verbindung Spurstange-Lenkgetriebe	3 Rot ., 1 Trans. (in Z)
N	-	Radmittelpunkt	-

Tabelle 1: Vereinbarte Punkte des Fahrwerks

3.2.4 Kernproblem

Folgendes bezieht sich auf Abbildung 3 und die Tabelle 1:

Um die Kinematik komplett beschreiben zu können, müssen die Positionen aller beweglichen Punkte (J, L, C, F, H, G, N) bei jeder sinnvollen Vorgabe der Freiheitsgrade eindeutig sein. Daraus ergeben sich folgende Funktionen, welche für jede Aufhängung zu formulieren sind.

An der Vorderachse:

$$f(x, y) = \underline{\underline{X}}$$

$x =$ Vorgabe der Federlänge

$y =$ Vorgabe des Lenkeinschlagwinkels

$\underline{\underline{X}} =$ Ergebnismatrix mit den Koordinaten aller beweglichen Punkte der Vorderachse

An der Hinterachse:

$$f(x) = \underline{\underline{X}}$$

$x =$ Vorgabe der Federlänge

$\underline{\underline{X}} =$ Ergebnismatrix mit den Koordinaten aller beweglichen Punkte der Hinterachse

Begonnen wird mit der Berechnung von Punkt J, da er der erste Punkt ist, der direkt von der Länge der Feder abhängt. J befindet sich in der gleichen Ebene wie die Punkte I, K, L und C in Ausgangslage. Die Punkte I und K sind an der Karosserie fixiert, wobei sich J abhängig von der Länge der Feder auf einer Kreisbahn bewegt, bei welcher der Radius die Länge der Feder ist und der Mittelpunkt dem Punkt I entspricht. Der zweite Kreis um den Punkt K ist mit dem Radius |g| immer konstant. Bei sinnvollen Federlängen (im Beispiel 150mm ≤ |f| ≤ 200mm) existieren immer zwei Schnittpunkte der beiden Kreise, da gilt: (|f| + |g|) > |IK|.

Für die Berechnung des Punktes L gilt dieses Prinzip nun analog. Der erste Kreis hat den Radius |h| um den Mittelpunkt K und der zweite Kreis hat den Radius |i| um den von |f| abhängigen Punkt J.

Ab der Berechnung des Punktes C wird das Problem räumlich. Punkt C entsteht als Schnittpunkt von drei Kugeln (Abbildung 4).

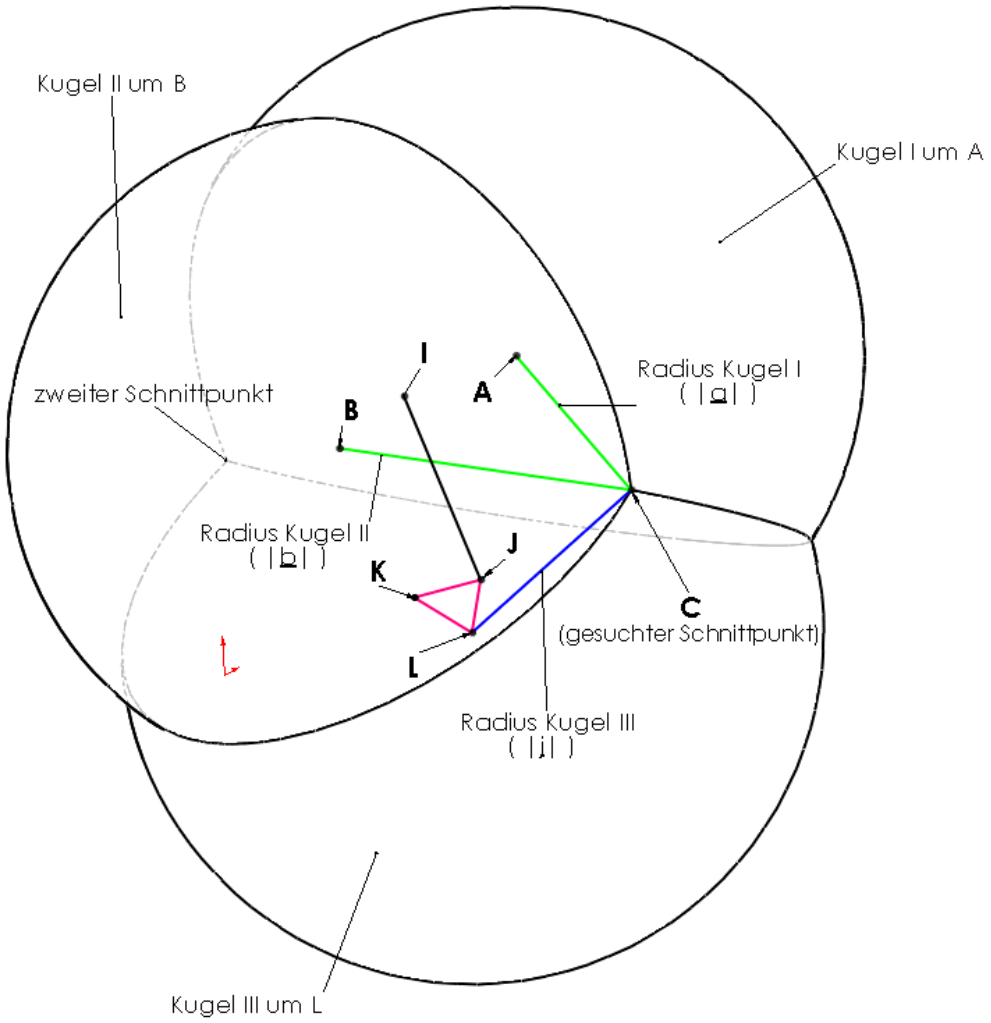


Abbildung 4: geometrische Lösung des Punktes C (räumlich)

Hier bilden nun A und B mit den Radien |a| und |b| die unabhängigen Kugeln und L mit |j| die von der Federlänge abhängige Kugel. Auch hier existieren immer zwei Schnittpunkte, da gilt: (|a| + |b|) > |AB|, (|a| + |j|) > |AL| und (|j| + |b|) > |LB|. Analog ist dieses Prinzip auf die restlichen Punkte übertragbar. Einzig spielt bei der Berechnung des Punktes H an der Vorderachse nun auch noch der Lenkeinschlagwinkel als zweiter Freiheitsgrad eine Rolle. Der Lenkeinschlagwinkel bewirkt eine Verschiebung des Punktes M parallel zur Z-Achse. Bei konstantem |c| bewirkt das die Lenkbewegung des Rades um die Lenkachse CF. Vom Lenkeinschlagwinkel werden damit die Punkte H, G und N beeinflusst. Tabelle 2 zeigt nun die Entstehung der einzelnen Punkte.

Punkt	Mittel-punkt I	Radius I	Mittel-punkt II	Radius II	Mittel-punkt III	Radius III	Bemerkung
A	-	-	-	-	-	-	festgelegter Punkt
B	-	-	-	-	-	-	festgelegter Punkt
D	-	-	-	-	-	-	festgelegter Punkt
E	-	-	-	-	-	-	festgelegter Punkt
I	-	-	-	-	-	-	festgelegter Punkt
K	-	-	-	-	-	-	festgelegter Punkt
M	-	-	-	-	-	-	Z-Komponente abhängig vom Lenkeinschlagwinkel
J	I	|f|	K	|g|	-	-	in Ebene, abhängig von |f|
L	J	|i|	K	|h|	-	-	In Ebene, J ist variabel
C	A	|a|	B	|b|	L	|j|	L ist variabel
F	C	|n|	D	|d|	E	|e|	C ist variabel
H	M	|c|	C	|o|	F	|p|	M, C und F sind variabel
G	C	|k|	F	|l|	H	|m|	C, F und H sind variabel

N	C	|g|	F	|s|	H	|r|	C, F und H sind variabel

Tabelle 2: Abhängigkeiten der Punkte des Fahrwerks

Die Tabelle 2 zeigt die sich ergebende Berechnungsreihenfolge ab dem Punkt J auf, wobei die Berechnungen der Punkte G und N vertauscht werden können, da sie von den selben variablen Punkten abhängen.

Nun gilt es, das gezeigte Problem zu lösen. Es soll möglich sein, abhängig der 5 Freiheitsgrade alle Fahrwerkspunkte der 4 Aufhängungen im Raum zu definieren. Das soll auch bei einer frei wählbaren Positionierung des Fahrzeugs im Raum möglich sein. D. h., die Positionen der Punkte hängen zusätzlich von 3 Translationen und 3 Rotationen ab.

3.3 Lösungsverfahren

Zur Bestimmung der Schnittpunkte von 3 Kugeln wird das Problem analytisch gelöst. Das Ergebnis ist ein Algorithmus zur Bestimmung der exakten Lösung. Zur Erläuterung soll die Skizze “Schnittpunktbildung“ (Abbildung 5) dienen.

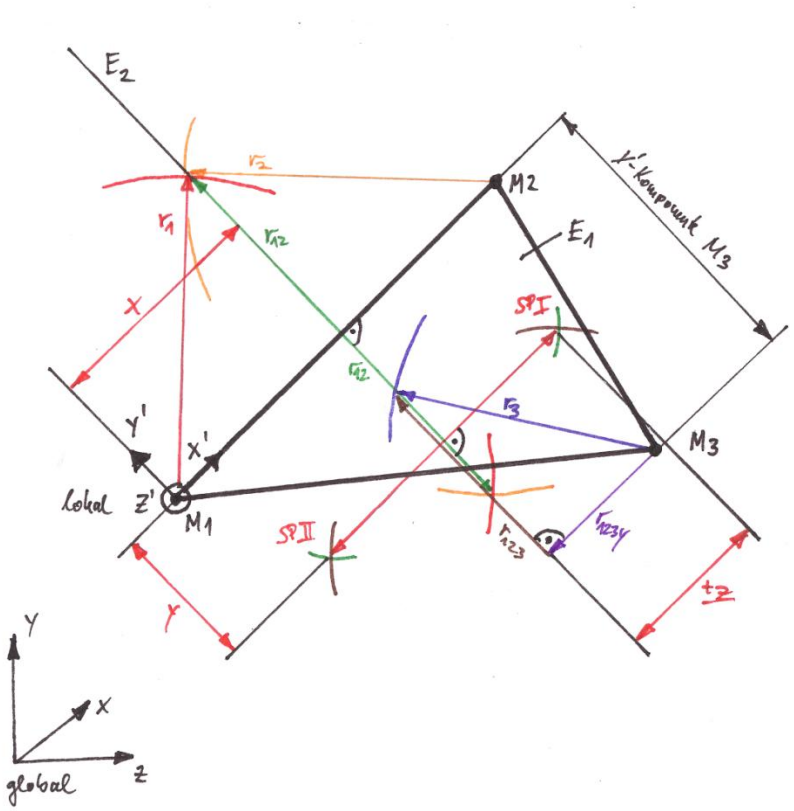


Abbildung 5: Schnittpunktbildung

Bekannt sind die drei Kugeln mit deren Mittelpunkten und Radien. Die Koordinaten der Mittelpunkte beziehen sich auf den globalen Ursprung.

Als erstes erfolgt eine Verschiebung aller Punkte so, dass der Mittelpunkt M1 auf dem globalen Ursprung liegt.

Als zweites wird der Verbindungsvektor von *M1M2* als lokale x'-Achse mit dem Einheitsvektor ex definiert.

Nun werden zwei Vektoren (*M1M2* und *M1M3*) gebildet, mit denen über das Vektorprodukt der senkrecht auf dieser Ebene 1 stehende Einheitsvektor *ez* erzeugt wird. Das Vektorprodukt von *ez* x *ex* ergibt den Einheitsvektor *ey*. Das gebildete Rechtssystem x'-y'-z' dient nun als lokales Koordinatensystem zur weiteren Betrachtung.

Nun folgt die Bildung einer Koordinatentransformationsmatrix, welche aus den Einheitsvektoren besteht. Mit dieser Matrix wird eine Koordinatentransformation durchgeführt, um die Mittelpunkte im lokalen System abzubilden. Dies vereinfacht die nachfolgenden Schritte, da die Geometrie nun in der Ebene *E1* bezogen auf das lokale System liegt.

Unter Verwendung des Satzes von Pythagoras und der Radien r_1 und r_3 kann nun der Radius r_{12} bestimmt werden. Dieser Radius beschreibt einen Schnittkreis der Kugeln um *M1* und *M2* auf der Ebene *E2*. Auf dieser Ebene befinden sich die beiden gesuchten Schnittpunkte. Der Abstand zwischen dem Mittelpunkt des Kreises mit dem Radius r_{12}, welcher auf der Geraden $\overline{M1M2}$ liegt, und dem Mittelpunkt *M1* ist somit die x-Koordinate der gesuchten Schnittpunkte im lokalen System.

Den Radius r_{123} des Kreises, der durch den Schnitt der dritten Kugel mit der Ebene *E2* erzeugt wird, gilt es nun zu berechnen. Dazu wird der Betrag von r_{123y} benötigt, mit dem dann unter Verwendung des Satzes von Pythagoras und dem Radius r_3 der projizierte Radius r_{123} ermittelt werden kann. Der Betrag von r_{123y} ergibt sich aus der Differenz der x'-Koordinate des Mittelpunktes *M3* und der bereits berechneten x-Koordinate. Der Abstand des Mittelpunktes des Kreises mit r_{123} zum Mittelpunkt des Kreises mit r_{12} entspricht der y'-Koordinate von *M3*.

Nun wird schließlich wieder der Satz von Pythagoras angewandt, um die y- und z-Koordinate der gesuchten Schnittpunkte zu berechnen. Bei den bisherigen Berechnungen waren nur die Beträge der Ergebnisse relevant. Bei der Berechnung der Schnittpunkte zwischen den Kreisen auf der Ebene *E2* sind nun aber beide Ergebnisse von Bedeutung. Der Unterschied zwischen den beiden Schnittpunkten ist aber nur das Vorzeichen der z-Koordinate.

Zuletzt wird der gewählte Schnittpunkt mit der transponierten Koordinatentransformationsmatrix zurück transformiert und anschließend mit dem globalen Ortsvektor von *M1* in das globale Koordinatensystem abgebildet. Dieses Verfahren funktioniert, solange die zwei Schnittpunkte existieren, was bei einem Fahrwerk von vorn herein Voraussetzung sein muss.

3.4 Implementierung in MATLAB®

Im Folgenden wird gezeigt, wie das analytische Lösungsverfahren in MATLAB® umgesetzt wurde.

3.4.1 Vorüberlegung

Beim Programmaufbau soll die Übersichtlichkeit der Programme und Unterprogramme gewährleistet sein. Dazu sind eine Hauptfunktion und mehrere Unterfunktionen zu erstellen. In der Hauptfunktion erfolgen die Ein- und Ausgaben. Es soll ein Skript mit den gesammelten Grundmaßen des Fahrwerks enthalten sein, da sonst der Eingabeaufwand zu groß wäre.

3.4.2 Programmablauf zur Lösung der Kinematik

Zur Lösung der Kinematik dient die Funktion “FWkomplett.m“ als Hauptfunktion. In ihr werden die Ein- und Ausgaben getätigt. Zunächst wird das script-File “InputKoord.m“ aufgerufen, um die Koordinaten der Punkte in Ausgangslage in den Workspace zu laden.

All diese Ortsvektoren werden nun so verschoben, dass der Schwerpunkt des Fahrzeugs deckungsgleich mit dem Ursprung des Koordinatensystems liegt.

Jetzt erfolgt die Übergabe der Längen der Feder-/Dämpfereinheiten und des Lenkeinschlagwinkels in die vier “solve_ij.m“ function- Files zur Berechnung der Punkte J bis N. Dabei wird in der Reihenfolge wie in Tabelle 2 vorgegangen. Die vorgegebenen bzw. berechneten Punkte bilden dabei die Mittelpunkte der Kreise/Kugeln. Die Radien ergeben sich immer aus den Beträgen zwischen den entsprechenden Punkten. Dazu werden die Punkte aus dem “InputKoord.m“ File genutzt, da sich die Distanzen zwischen den entsprechenden Punkten während der Bewegung nicht ändern.

Die “solve_ij.m“ function- Files übergeben die einzelnen Mittelpunkte und Radien an das “KugSchn.m“ function- File zur Berechnung der gesuchten Schnittpunkte, welche an die “solve_ij.m“ function- Files zurückgegeben werden. Die Punkte werden nun in einer Matrix zusammengestellt und an die Hauptfunktion übergeben.

Von hier aus erfolgt nun eine Koordinatentransformation in dem “Kootrans.m function-File, welche von den eingegebenen Drehwinkeln alpha1, alpha2 und alpha3 abhängt.

Abschließend werden alle Ortsvektoren der Punkte mit dem Vektor des Schwerpunkts addiert, um anschließend durch die drei Translationen X, Y und Z in die gewünschte Lage verschoben zu werden.

Am Ende werden die Ortsvektoren der Punkte in vier Matrizen ausgegeben. Der beschriebene Ablauf wird in dem Flussdiagramm der Abbildung 6 noch einmal verdeutlicht. Die zum Verständnis dienliche Abkürzungstabelle, sowie die Quelltexte der einzelnen Funktionen und Skripte befinden sich in der Anlage Teil 1.

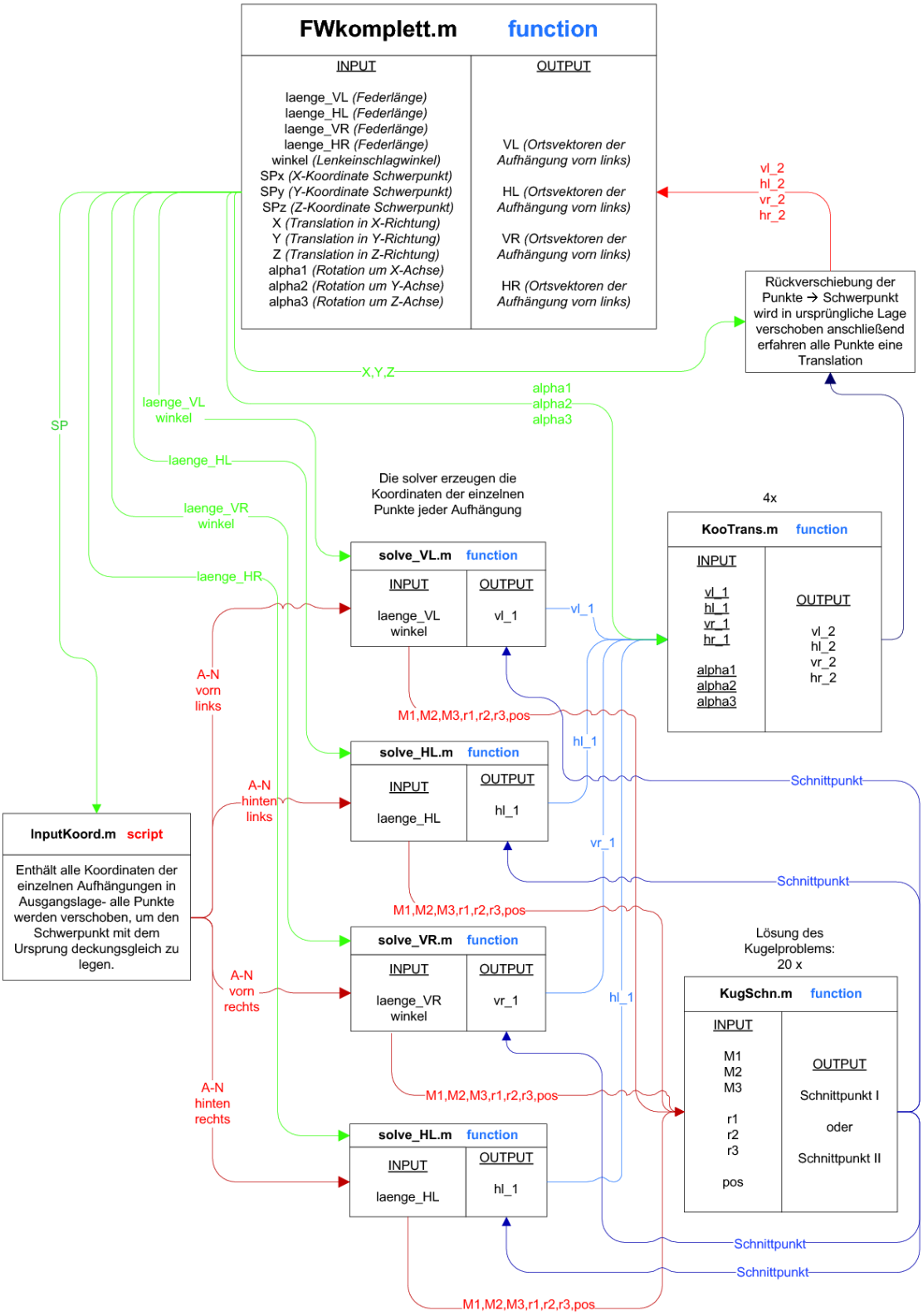


Abbildung 6: Flussdiagramm Kinematik

3.4.3 Ergebnisse

Als Ergebnis gibt MATLAB® Matrizen aus, welche die einzelnen Fahrwerkspunkte der Aufhängungen enthalten. Diese Lösungen stehen somit für den weiteren Gebrauch zur Verfügung.

Die Richtigkeit der Ergebnisse wurde mit einem, entsprechend der Freiheitsgrade parametrisiertem, CAD-Modell in SolidWorks® überprüft.

Die Quelltexte zur Kinematik befinden sich in der Anlage, Teil 1.

4 Lagrange 1. Art

Die grundlegende Theorie zur Berechnung der Problemstellung wird in diesem Kapitel gezeigt. Anhand eines Beispiels wird die Anwendung der Theorie erläutert. Anschließend folgt die Umsetzung in MATLAB®, wobei die Entstehung und der Aufbau eines systemunabhängigen Programms erläutert wird.

4.1 Theorie zu Lagrange 1. Art

Die Lagrangesche Gleichung 1. Art behandelt im Gegensatz zur 2. Art (Gleichung [1]) *auch kinematische Zwangsbedingungen.*[11] Dieser Vorteil soll bei der Berechnung des Fahrwerks genutzt werden um die Fahrsituation vorzugeben. Das heißt, die Zwänge bilden dann z. B. Fahrbahnunebenheiten ab. Allgemein lautet die Lagrangesche Gleichung 1. Art:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_\iota}\right) - \frac{\partial L}{\partial q_i} = R_i = Q_i + Z_i \qquad [3]$$

R_i steht für die Restkräfte, wobei Z_i die Zwangskräfte der i-ten Koordinate sind und Q_i die nichtkonservativen Kräfte der i-ten Koordinate. Die generalisierten Koordinaten werden mit q_i bezeichnet. Bei der Fahrwerksberechnung werden die Zwangskräfte Z_i und die Dämpferkräfte Q_i in die Berechnung mit eingehen.

4.1.1 Aufstellen der Zwangsbedingungen

Für alle folgenden Betrachtungen gilt die Einstein'sche Summenkonvention. *Zunächst wird untersucht, ob das System nicht holonome Zwangsbedingungen enthält. Bei nicht holonomen Systemen (solche mit nicht holonomen Zwängen) existiert eine unabhängige Anzahl von Geschwindigkeiten F_G und eine unabhängige Anzahl Koordinaten F_K, wobei $F_G < F_K$ ist. Bei holonomen Systemen ist die Anzahl von $F_G = F_K$, dies entspricht der maximalen Anzahl der Freiheitsgrade N mit $q = q_i = (q_1 , q_2 , \ldots, q_N)$. Wenn $N = F_K$ ist, gilt die Lagrange Gleichung II. Art (Siehe [1] Seite 11). Für den Fall das $N \geq F_K = F_G$ gilt die Lagrange Gleichung I. Art (Siehe [3] Seite 28), worauf sich das Folgende bezieht.*

[11] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, B.G. Teubner Verlag, Wiesbaden, 2.Auflage, 2006, S.228

Im Fall von Lagrange I. Art liegen nun noch Z = N - F_G Zwangsbedingungen vor. Im Folgenden wird sich nur auf Systeme beschränkt, für die sich die Zwangsbedingungen so formulieren lassen:

$$B_i^{\mu}(q,t) * \dot{q}_{\iota} + B_0^{\mu}(q,t) = 0 \qquad [4]$$

mit μ = 1, …, Z; i= 1, …, N

Es wird angenommen, dass sämtliche Zwangsbedingungen linear in der Geschwindigkeit sind. Als Sonderfall sind hierbei auch Zwangsbedingungen enthalten, die nur von den generalisierten Koordinaten und explizit von der Zeit abhängen (holonome Zwänge). Diese Teilmenge der Z Nebenbedingungen lassen sich dann in der Form:

$$f^{v}(q,t) = 0 \qquad [5]$$

$$v = 1, \dots, Z_H \leq Z \qquad \text{schreiben.}$$

Wird die Gleichung total nach der Zeit differenziert, so zeigt sich, dass sie eine Teilmenge der obigen Zwangsbedingungen darstellen, wenn man setzt:

$$B_i^{v}(q,t) = \partial_i f^{v} \ , \ B_0^{v}(q,t) = \partial_t f^{v} \qquad [6]$$

mit i = 1, …, N; v = 1, …, Z_H ≤ Z.

Mit der Einführung von λ ergeben sich dann die generalisierten Zwangskräfte Z_i zu:

$$Z_i = \lambda_{\mu} B_i^{\mu} \qquad [7]$$

mit μ = 1, …, Z; i = 1, …, N.

Bei der Anwesenheit von nichtkonservativen Kräften, welche nicht aus der potentiellen Energie hervorgehen, wird dieser Anteil zu den Zwangskräften hinzu addiert. Die vollständigen Restkräfte lauten dann:

$R_i = Z_i + Q_i$ mit i = 1, …, N. [8]

So erhält man die N + Z Gleichungen. Jedoch können die Differentiationen nun noch nicht ausgeführt werden. Zunächst müssen die Zwangsbedingungen aufbereitet werden.

4.1.2 Aufstellen des Gesamtgleichungssystems

Wird die totale Zeitableitung der Zwangsbedingung [7] gebildet, so ergeben sich Z in den zweiten Zeitableitungen der Koordinaten lineare Gleichungen:

$$B_i^{\mu}(q,t)\ddot{q}^i + \dot{B}_i^{\mu}(q,t)\dot{q}^i + \dot{B}_0^{\mu}(q,t) = 0 \qquad [9]$$

mit μ = 1, …, Z; i = 1, …, N.

Hierbei ist nun $\ddot{q}_i$ in linearem Zusammenhang vertreten. Auch nach Einsetzen in die Lagrange-Gleichungen tritt $\ddot{q}_i$ nur noch linear auf. Somit können nun die Zwänge eingesetzt werden und $\ddot{q}_i$ mit $\dot{\varphi}_i$ substituiert werden. Die Gleichungen werden nun explizit nach $\dot{\varphi}_i$ umgestellt und können gelöst werden. [12]

4.2 Anwendung am Beispiel

Um die Theorie aus 4.1 zu testen, wurde ein einfaches Beispiel genutzt, um die Anwendbarkeit auf das Fahrwerk zu prüfen. Hierbei handelt es sich um einen horizontalen Zweimassenschwinger, bei dem die gleich schweren Massen zwischen Federn mit gleicher Steifigkeit eingespannt sind. Die äußeren Federn sind dabei fest gelagert. In Abbildung 7 wird dieses System veranschaulicht.

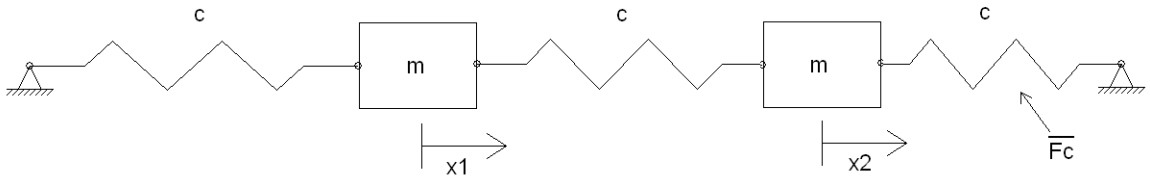


Abbildung 7: System Zweimassenschwinger

In Zusammenarbeit mit Prof. Zimmermann entstand nun folgende Umsetzung der Theorie am Beispiel.

4.2.1 Systemuntersuchung

Ohne Zwang hat das System zwei Freiheitsgrade q_1 und q_2, welche als generalisierte Koordinaten festgelegt werden. Der nun willkürlich festgelegte Zwang bewirkt den Stillstand der rechten Masse, indem auf die Feder rechts außen eine Kraft aufgegeben wird, die genauso groß ist wie die Federkraft, welche beim Schwingen wirken würde, jedoch mit umgekehrtem Vorzeichen.

[12] http://mechanik.tu-berlin.de/brunk/index.htm, verfügbar am 10.08.2011, 14:00 Uhr

Freiheitsgrade: N=2

Anzahl der Zwänge: Z=1

Zwang: $F_c = c * x_2$

$$f^1 = F_c - c * x_2 = 0 \quad [10]$$

Der Zwang ist holonom, d. h. die Anzahl der unabhängigen Geschwindigkeiten ist gleich der Anzahl der unabhängigen Koordinaten (F_G=F_K). Durch Einführung des Zwangs ist nun der Freiheitsgrad der rechten Masse eingeschränkt. Nun gilt N > F_K = F_G. Damit existiert noch N – F_G = 1 Zwangsbedingung, welche bereits unter [10] festgelegt wurde. Im System sind die nichtkonservativen Kräfte Q_i=0. Die Restkräfte bestehen also nur aus den Zwangskräften.

4.2.2 Aufbereitung der Zwangsbedingungen

Zunächst werden die Zwangsbedingungen aus [10] abgeleitet.

$$B_1^1 = \frac{\partial f^1}{\partial x_1} = 0$$

$$B_2^1 = \frac{\partial f^1}{\partial x_2} = -c$$

$$B_0^1 = \frac{\partial f^1}{\partial t} = 0$$

$$\dot{B}_1^1 = 0$$

$$\dot{B}_2^1 = 0$$

$$\dot{B}_0^1 = 0 \quad [11]$$

Um Z_i zu ermitteln werden die Ableitungen nun in [7] eingesetzt.

$$Z_1 = \lambda_1 * B_1^1 = 0$$

$$Z_2 = \lambda_2 * B_2^1 = -\lambda_2 * c \quad [12]$$

Die Gleichungen [4] und [9] werden nun bestückt um sie später mit in das Gleichungssystem einfließen zu lassen:

$$B_1^1 * \dot{x}_1 + B_0^1 = 0 \rightarrow \dot{x}_1 = 0$$
$$B_2^1 * \dot{x}_2 + B_0^1 = 0 \rightarrow \dot{x}_2 = 0 \quad [13]$$

$$\dot{B}_1^1 * \dot{x}_1 + B_1^1 * \ddot{x}_1 + \dot{B}_0^1 = 0 \rightarrow \ddot{x}_1 = 0$$
$$\dot{B}_2^1 * \dot{x}_2 + B_2^1 * \ddot{x}_2 + \dot{B}_0^1 = 0 \rightarrow \ddot{x}_2 = 0 \quad [14]$$

4.2.3 Aufbau des Grundgleichungssystems

Zur Bestimmung der Bewegungsgleichungen muss die Lagrange-Funktion aufgestellt werden. Für das Beispiel lautet sie wie folgt:

$$L = 0{,}5 * m * \left(\dot{x}_1{}^2 + \dot{x}_2{}^2\right) - 0{,}5 * c * (\dot{x}_1{}^2 + \dot{x}_2{}^2 + (\dot{x}_2 - \dot{x}_1)^2) \quad [15]$$

Nach den Ableitungen nach der Lagrange-Gleichung [3] entstehen die beiden Bewegungsgleichungen unter Einbeziehung der Zwangskraft:

$$m * \ddot{x}_1 + c * x_1 - c * (x_2 - x_1) = 0 \quad [16]$$

$$m * \ddot{x}_2 + c * x_2 - c * (x_2 - x_1) = -\lambda_2 * c \quad [17]$$

Nun wird durch die Einführung von $\dot{\varphi}_i = \ddot{q}_i$ das Gleichungssystem lösbar formuliert.

$$\varphi = \begin{matrix} x_1 \\ \dot{x}_1 \\ x_2 \\ \dot{x}_2 \\ \lambda \end{matrix} \qquad \dot{\varphi} = \begin{matrix} \dot{x}_1 \\ \ddot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_2 \\ \dot{\lambda} \end{matrix} \quad [18]$$

Durch Einsetzten von [18] in [16] und [17] und Umstellen nach $\dot{\varphi}_i$ erhält man das gesuchte Gleichungssystem.

$$\dot{\varphi}_1 = \varphi_2$$

$$\dot{\varphi}_2 = \frac{c(-2\varphi_1+\varphi_3)}{m}$$

$$\dot{\varphi}_3 = \varphi_4$$

$$\dot{\varphi}_4 = \frac{c(\varphi_1-2\varphi_3-\varphi_5)}{m}$$

$$\dot{\lambda} = -2\varphi_4 + \varphi_2 \qquad [19]$$

Nun kann das Gleichungssystem numerisch gelöst werden. Die gewählten Anfangswerte müssen dazu die Gleichung [4] erfüllen, um sinnvoll zu sein. Als Ergebnis erhält man nach der Vereinbarung unter [18] schließlich $\varphi_1, \varphi_2, \varphi_3, \varphi_4, \lambda.$ φ_1 entspricht somit dem Weg der Masse 1 zu jedem Zeitpunkt t und φ_2 entspricht der Geschwindigkeit der Masse 1 zu jedem Zeitpunkt t. Das gilt analog für die Masse 2 mit φ_3, φ_4. Die Ergebnisse von λ entsprechen im betrachteten Beispiel dem virtuellen Federspannweg der Feder 3.

5 Programmentwicklung in MATLAB®

Zunächst soll das Problem aus 4.2 soweit in MATLAB® implementiert werden, dass die Lösung erzeugt wird. Danach soll eine Universalisierung des Lösungsalgorithmus erfolgen, um Probleme beliebiger Art, welche auch getestet werden sollen, lösen zu können. Erst danach folgt die Anpassung des Programms an das Komplexbeispiel, den Rennwagen.

In Debatten mit Professor Zimmermann entstanden folgende Überlegungen zur Umsetzung der unter 4.1 behandelten Theorie.

5.1 Verwendung des ode45 Solver

In MATLAB® stehen zur Lösung von Differenzialgleichungssystemen die ode-Solver zur Verfügung. *MATLAB® empfiehlt die Verwendung des ode45-Solver bei nichtlinearen Problemen und bei einer mittleren geforderten Genauigkeit.*[13] Die Syntax des Solvers lautet

$$[\ T,\ Y\ ] = \text{ode45}\ (\ @\ \text{Handlefunction},\ \text{tspan},\ y_0\ )$$

In der Handlefunction, wobei es sich um eine MATLAB® Funktion handelt, wird das zu lösende Gleichungssytem bereitgestellt.

Form des Gleichungssystems:

$$y' = f(t, y)$$

Das Zeitintervall wird mit der tspan-Variable festgelegt. Dazu wird ein Vektor der Form [t_0 t_{end}] mit der Start- und Endzeit erzeugt. Die Anfangswerte werden dem Solver als Vektor y_0 übergeben. Durch Integration löst der ode45-Solver nun das Gleichungssytem und gibt mit der Variable T die Zeitschritte, an denen eine Berechnung erfolgte, zurück. Die zugehörigen Ergebnisse werden in Vektor Y geschrieben. Der Nutzer hat nun verschiedene Möglichkeiten, die Ergebnisse weiterzuverarbeiten. Unter anderem ist es möglich, Diagramme zu erstellen oder die Datensätze zur Weiterverwendung in Tabellenform zu speichern.

[13] MATLAB® Hilfe der Version 7.10.0.499 (R2010a)

Der ode45 soll nun als Hauptprogramm zur Lösung des Problems dienen. Als MATLAB® Skript-File werden hier zu jedem Freiheitsgrad des Systems die Anfangswerte festgelegt. Danach folgt der Aufruf des Solvers und zum Schluss die Aufbereitung der Ergebnisse.

In der Handlefunction soll durch den Zugriff auf verschiedene Unterfunktionen das Gleichungssystem erzeugt werden, welches dann zu jedem Zeitschritt gelöst wird.

5.2 Anpassung der Lösungsstrategie

Um das Verfahren nach Lagrange 1. Art aus Kapitel 4 in MATLAB® auf universelle Weise anwenden zu können, muss die Lösungsstrategie angepasst werden.

Die Lagrange Gleichung [3] wurde bisher nur für die Anwendung an einer generalisierten Koordinate formuliert. Nun muss ein Formalismus erarbeitet werden, mit dem eine programmtechnisch sinnvolle Herangehensweise an das Aufstellen des Gleichungssystems möglich wird.

5.2.1 Kompilation des Grundgleichungssystems

Ausgehend von der Lagrange Gleichung [1] erfolgt die Auseinandersetzung zunächst ohne die Betrachtung von Zwängen. Bei einer Anzahl von n generalisierten Koordinaten muss die Lagrange Gleichung ebenso oft gebildet werden.

Es gilt: $$L(q_i, \dot{q}_i) \quad [20]$$

Somit kann aus $$\frac{\partial L}{\partial q_i} \quad [21]$$

der Vektor $$\begin{bmatrix} \frac{\partial L}{\partial q_1} \\ \vdots \\ \frac{\partial L}{\partial q_n} \end{bmatrix}$$ gebildet werden. [22]

Der gleiche Schritt wird nun mit dem zweiten Teil der Lagrange Gleichung vollzogen.

Aus $$\frac{\partial L}{\partial \dot{q}_i} \quad [23]$$

wird nun der Vektor $$\begin{bmatrix} \frac{\partial L}{\partial \dot{q}_1} \\ \vdots \\ \frac{\partial L}{\partial \dot{q}_N} \end{bmatrix}$$ gebildet. [24]

Nach der Lagrange Gleichung [1] muss der Vektor [23] nun noch nach der Zeit abgeleitet werden. Dabei entsteht ein Vektor, welcher entsprechende partielle Ableitungen enthält.

$$\left(\frac{\partial \dot{L}}{\partial \dot{q}_i}\right) = \begin{bmatrix} \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_1} * \dot{q}_1 + \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_1} * \ddot{q}_1 + & \dots & + \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_N} * \dot{q}_N + \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_N} * \ddot{q}_N \\ \vdots & & \vdots \\ \frac{\partial^2 L}{\partial \dot{q}_N \partial q_1} * \dot{q}_1 + \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_1} * \ddot{q}_1 + & \dots & + \frac{\partial^2 L}{\partial \dot{q}_N \partial q_N} * \dot{q}_n + \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_N} * \ddot{q}_N \end{bmatrix}$$

[25]

Als nächstes werden die $\dot{q}_N$ und die $\ddot{q}_N$ aus dem Vektor [25] ausgeklammert. Dadurch entsteht die systembeherrschende Matrix. Gleichzeitig erfolgt die Substitution der generalisierten Koordinaten wie in [18],

$$\begin{bmatrix} \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_1} & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_1} & \dots & \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_i} & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_i} & \dots & \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_N} & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_N} \\ \vdots & \vdots & & & & & & \\ \frac{\partial^2 L}{\partial \dot{q}_i \partial q_1} & \frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_1} & \dots & \frac{\partial^2 L}{\partial \dot{q}_i \partial q_i} & \frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_i} & \dots & \frac{\partial^2 L}{\partial \dot{q}_i \partial q_N} & \frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_N} \\ \vdots & \vdots & & & & & & \\ \frac{\partial^2 L}{\partial \dot{q}_N \partial q_1} & \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_1} & \dots & \frac{\partial^2 L}{\partial \dot{q}_N \partial q_i} & \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_i} & \dots & \frac{\partial^2 L}{\partial \dot{q}_N \partial q_N} & \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_N} \end{bmatrix} * \begin{bmatrix} \dot{q}_1 \\ \ddot{q}_1 \\ \vdots \\ \dot{q}_i \\ \ddot{q}_i \\ \vdots \\ \dot{q}_N \\ \ddot{q}_N \end{bmatrix}$$

[26]

wodurch

$$\begin{bmatrix} \dot{q}_1 \\ \ddot{q}_1 \\ \vdots \\ \dot{q}_i \\ \ddot{q}_i \\ \vdots \\ \dot{q}_N \\ \ddot{q}_N \end{bmatrix} = \begin{bmatrix} \dot{\varphi}_1 \\ \vdots \\ \dot{\varphi}_i \\ \vdots \\ \dot{\varphi}_{2*N} \end{bmatrix} = \underline{\dot{\varphi}} \text{ ist.}$$

[27]

Um im Gleichungssystem auch die Geschwindigkeiten zu berücksichtigen, müssen der systembeherrschende Vektor [24] und die systembeherrschende Matrix [26] noch erweitert werden. Der Vektor [24] erhält weiter Zeilen, die der Anzahl der generalisierten Geschwindigkeiten entsprechen und mit jenen bestückt werden. Der endgültige systembeherrschende Vektor lautet dann wie folgt.

$$\underline{b} = \begin{bmatrix} \frac{\partial L}{\partial \dot{q}_1} \\ \vdots \\ \frac{\partial L}{\partial \dot{q}_N} \\ \dot{q}_1 \\ \vdots \\ \dot{q}_N \end{bmatrix} \quad [28]$$

Um die systembeherrschende Matrix anzupassen, werden auch hier Zeilen angefügt. Die Anzahl der hinzukommenden Zeilen entspricht der Anzahl der generalisierten Koordinaten. Die Zeilen enthalten Nullen, bis auf die Positionen, an denen sie bei der Multiplikation mit dem Vektor [28] die Geschwindigkeiten eliminieren würden. An diesen Stellen sind die Zeilen mit einer 1 besetzt. Das bedeutet, das in jeder Zeile eine 1 vorkommt, die aufsteigen in den ungeraden Spalten steht. Konkret ist die systembeherrschende Matrix wie hier am Beispiel des Zweimassenschwingers besetzt:

$$\underline{\underline{A}} = \begin{bmatrix} \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_1} & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_1} & \frac{\partial^2 L}{\partial \dot{q}_1 \partial q_2} & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_2} \\ \frac{\partial^2 L}{\partial \dot{q}_2 \partial q_1} & \frac{\partial^2 L}{\partial \dot{q}_2 \partial \dot{q}_1} & \frac{\partial^2 L}{\partial \dot{q}_2 \partial q_2} & \frac{\partial^2 L}{\partial \dot{q}_2 \partial \dot{q}_2} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad [29]$$

Nach Umstellen und Einsetzen in die Lagrange Gleichung [1] erhält man somit:

$$\underline{\underline{A}} * \underline{\dot{\varphi}} = \underline{b} \quad [30]$$

Um das Gleichungssystem nun explizit nach $\underline{\dot{\varphi}}$ umzustellen, wird dividiert:

$$\underline{\dot{\varphi}} = \underline{\underline{A}}^{-1} * \underline{b} \quad [31]$$

Damit ist eine Strategie entwickelt, mit der die Lösung von beliebigen restkraftlosen dynamischen Systemen nur noch von der Lagrange Funktion mit der Anzahl ihrer generalisierten Koordinaten abhängt. Die systembeherrschende Matrix und der systembeherrschende Vektor müssen nun in Unterfunktionen der Handlefunction erzeugt werden, um in der Handlefunction nach [31] aufbereitet zu werden. Jetzt kann das Gleichungssystem dem ode45-Solver zum Lösen bereitgestellt werden.

5.2.2 Automatisierte Implementation der Zwangskräfte

Nun muss noch die strategische Aufbereitung der Berechnung der Zwangsbedingungen erfolgen, um der Gleichung [3] der Lagrange Gleichung 1. Art zu entsprechen. Dazu muss die Berechnung der Zwangskräfte aus [7] universell formuliert werden. Hier werden als erstes die Matrix und der Vektor formuliert:

$$\underline{B_i^\mu}(q,t) = \partial_i f^\mu = 0 \qquad [32]$$

$$\underline{B_0^\mu}(q,t) = \partial_t f^\mu = 0 \qquad [33]$$

Nun folgt das Bilden der Ableitungen aus Gleichung [9]. Diese Ableitungen werden nun ebenfalls als Matrix/ Vektor formuliert:

$$\underline{\dot{B}_i^\mu} = \sum_j \frac{\partial B_i^\mu}{\partial q_j} * \dot{q}_j + \frac{\partial B_i^\mu}{\partial t} \qquad [34]$$

$$\underline{\dot{B}_0^\mu} = \sum_j \frac{\partial B_0^\mu}{\partial q_j} * \dot{q}_j + \frac{\partial B_0^\mu}{\partial t} \qquad [35]$$

Nun erfolgt der Bau des Gleichungssystems nach [9]. Dabei sind die $\ddot{q}_i$ jedoch noch unbekannt. In Matrixschreibweise lautet das Gleichungssystem nun so:

$$\begin{bmatrix} B_1^1 & \dots & B_i^1 & \dots & B_N^1 \\ \vdots & & \vdots & & \vdots \\ B_1^\mu & \dots & B_i^\mu & \dots & B_N^\mu \\ \vdots & & \vdots & & \vdots \\ B_1^Z & \dots & B_i^Z & \dots & B_N^Z \end{bmatrix} * \begin{bmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_i \\ \vdots \\ \ddot{q}_N \end{bmatrix} = \begin{bmatrix} -\sum_i \dot{B}_i^1 * \dot{q}_j - \dot{B}_0^1 \\ \vdots \\ -\sum_i \dot{B}_i^\mu * \dot{q}_j - \dot{B}_0^\mu \\ \vdots \\ -\sum_i \dot{B}_i^Z * \dot{q}_j - \dot{B}_0^Z \end{bmatrix} \qquad [36]$$

Dieses Gleichungssystem wird vorerst nach $\underline{\ddot{q}_i}$ aufgelöst. Nun müssen noch die λ_μ berechnet werden. Dazu wird unter Verwendung der Berechnungen aus 5.2.1 und der Lagrange Gleichung 1. Art [3] wiederum ein Gleichungssystem aufgestellt. Die Gleichung [3] wird Z_i umgestellt und in Matrizenform formuliert:

$$\begin{bmatrix} B_1^1 & \dots & B_1^\mu & \dots & B_1^Z \\ \vdots & & \vdots & & \vdots \\ B_i^1 & \dots & B_i^\mu & \dots & B_i^Z \\ \vdots & & \vdots & & \vdots \\ B_N^1 & \dots & B_N^\mu & \dots & B_N^Z \end{bmatrix} * \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_\mu \\ \vdots \\ \lambda_Z \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial \dot{L}}{\partial \dot{q}_1}\right) - \frac{\partial L}{\partial q_1} \\ \vdots \\ \left(\frac{\partial \dot{L}}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} \\ \vdots \\ \left(\frac{\partial \dot{L}}{\partial \dot{q}_N}\right) - \frac{\partial L}{\partial q_N} \end{bmatrix} \qquad [37]$$

Nun kann nach $\underline{\lambda_\mu}$ aufgelöst werden.

Zusätzlich zu [36] wird der Vektor $\underline{\ddot{q}_i}$ nun auch noch wie folgt ausgedrückt:

$$\begin{bmatrix} \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_1} & \cdots & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_i} & \cdots & \frac{\partial^2 L}{\partial \dot{q}_1 \partial \dot{q}_N} \\ \vdots & & & & \\ \frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_1} & \cdots & \frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_i} & \cdots & \frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_N} \\ \vdots & & & & \\ \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_1} & \cdots & \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_i} & \cdots & \frac{\partial^2 L}{\partial \dot{q}_N \partial \dot{q}_N} \end{bmatrix} * \begin{bmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_i \\ \vdots \\ \ddot{q}_N \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{\partial^2 L}{\partial \dot{q}_1 \partial q_1} & \cdots & -\frac{\partial^2 L}{\partial \dot{q}_1 \partial q_i} & \cdots & -\frac{\partial^2 L}{\partial \dot{q}_1 \partial q_N} \\ \vdots & & \vdots & & \vdots \\ -\frac{\partial^2 L}{\partial \dot{q}_i \partial q_1} & \cdots & -\frac{\partial^2 L}{\partial \dot{q}_i \partial q_i} & \cdots & -\frac{\partial^2 L}{\partial \dot{q}_i \partial q_N} \\ \vdots & & \vdots & & \vdots \\ -\frac{\partial^2 L}{\partial \dot{q}_N \partial q_1} & \cdots & -\frac{\partial^2 L}{\partial \dot{q}_N \partial q_i} & \cdots & -\frac{\partial^2 L}{\partial \dot{q}_N \partial q_N} \end{bmatrix} * \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_i \\ \vdots \\ \dot{q}_N \end{bmatrix} + \begin{bmatrix} \frac{\partial L}{\partial q_1} \\ \vdots \\ \frac{\partial L}{\partial q_i} \\ \vdots \\ \frac{\partial L}{\partial q_N} \end{bmatrix}$$

[38]

In kurzschreibweise lautet das Gleichungssystem so:

$$\underline{\underline{C_1}} * \underline{\dot{q}_i} = \underline{C_2} \qquad [39]$$

Dieses Gleichungssystem geht wiederum aus der Lagrange Gleichung 2. Art [1] hervor und stellt durch Division die zweiten Ableitungen der generalisierten Koordinaten zur Verfügung. Die Matrix auf der linken Seite entspricht dabei den geraden Spalten der systembeherrschenden Matrix [29] ohne die Erweiterung der Zeilen. Die Matrix auf der linken Seite entspricht den ungeraden Spalten der systembeherrschenden Matrix [26] ebenfalls ohne die Erweiterung der Zeilen.

Zur letztendlichen Bestimmung der $\underline{\lambda_\mu}$ erfolgt durch die Konstruktion eines Gleichungssystems, welches aus den eben erstellten Systemen resultiert:

$$\begin{bmatrix} \left[\underline{\underline{C_1}}\right] & \left[\left[\underline{\underline{B_i^\mu}}\right]^T\right] \\ \left[\underline{\underline{B_i^\mu}}\right] & [0] \end{bmatrix} * \begin{bmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_N \\ \lambda_1 \\ \vdots \\ \lambda_Z \end{bmatrix} = \begin{bmatrix} \left[\underline{C_2}\right] \\ \begin{bmatrix} -\sum_i \dot{B}_i^1 \dot{q}_i - \dot{B}_0^1 \\ \vdots \\ -\sum_i \dot{B}_i^\mu \dot{q}_i - \dot{B}_0^\mu \\ \vdots \\ -\sum_i \dot{B}_i^Z - \dot{B}_0^Z \end{bmatrix} \end{bmatrix} \qquad [40]$$

Die durch Division entstehenden λ_μ können nun mit den B_i^μ multipliziert werden und ergeben damit die Z_i, welche in die Lagrange Gleichung [3] eingehen. Die entstandenen $\ddot{q}_i$

werden vorerst nicht benötigt. Es wird deutlich, dass zur Erzeugung der Zwangskräfte die komplette Betrachtung des Problems wie in 5.2.1 zu Grunde liegen muss [38]. Im Programm wird es daher sinnvoll sein, während eines Zeitschrittes darauf zurück zugreifen.

5.2.3 Bau des Gesamtgleichungssystems

Mit den gewonnen Z_i kann das Gleichungssystem [31] nun noch ergänzt werden, indem vom Vektor b aus [28] die jeweiligen Z_i subtrahiert werden. Das Gesamtgleichungssystem lautet damit:

$$\underline{\dot{\varphi}} = \underline{\underline{A}}^{-1} * (\underline{b} - \underline{Z}) \qquad [41]$$

Das damit erzeugte Gesamtgleichungssystem kann somit dem ode45-Solver übergeben werden. Die Erzeugung von $\underline{\underline{A}}$, $\underline{b}$ und $\underline{Z}$ wird in Unterfunktionen der Handlefunction vollzogen.

5.3 Programmaufbau

Nun wird die Herangehensweise zum Aufbau des Grundprogramms zur Lösung dynamischer Probleme am Beispiel des Zweimassenschwingers in MATLAB® erläutert.

5.3.1 Symbolische Herangehensweise

Als erstes stand die Idee, den Programmaufbau in MATLAB® symbolisch zu realisieren. Dazu steht in MATLAB® die Symbolic Toolbox zur Verfügung. *Die Symbolic Math Toolbox in MATLAB® ermöglicht die symbolische, d. h. exakte Behandlung mathematischer Probleme in der numerischen Umgebung. Dies ist insbesondere für die symbolische Herleitung von Bewegungsgleichungen und deren numerische Auswertung von Nutzen*.[14] Die Symbolic Math Toolbox behandelt die in Tabelle 3 aufgezeigten Themen. Für die dynamischen Berechnungen am Beispiel werden unter anderem Operationen wie die Differentiation, das Lösen von Gleichungssystemen und das numerische Auswerten mathematischer Ausdrücke benötigt.

Die Vorteile der Nutzung der bestehenden Ressourcen und die Exaktheit der Lösungen sind für die symbolische Herangehensweise ausschlaggebend.

[14] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, B.G. Teubner Verlag, Wiesbaden, 2.Auflage, 2006, S.67

Zuordnung	Operationen
Analysis	Differentiation, Integration, Grenzwertbil-dung, Summation, TAYLOR-Polynome
Lineare Algebra	Determinante, Inverse, Eigenwerte, Singulär Werte SVD, kanonische Formen
Lösen von Gleichungssys-temen	Symbolische und numerische Lösungen linearer algebrai-scher Gleichungen und Differentialgleichungen
Spezielle mathematische Funktionen	Spezielle Funktionen der klassischen angewandten Ma-thematik, z.B. Cos-Integral
Variable Exaktheits-Arithmetik	Numerische Auswertung mathematischer Ausdrücke unter-schiedlicher Genauigkeit
Transformationen	Fourier-, Laplace-, z-Transformationen mit zugehörigen inversen Transformationen

Tabelle 3: Themen der Symbolic Math Toolbox[15]

Nun wird der Aufbau des Programms mit den zu Grunde liegenden Gedankengängen erläutert. Dazu dient auch die Abbildung 8, auf welcher der Programmablauf als Flussdiagramm dargestellt ist. Die Quelltexte zum Programm befinden sich im Anhang, Teil 2. Die numerischen Konstanten die Anfangswerte und der Zwang sind identisch mit denen in 4.2.

Nach dem Start des Hauptprogramms “ODEsolver.m“ in MATLAB® werden die vereinbarten symbolischen und numerischen Variablen geladen um die Anfangswerte aufzubereiten. Danach wird der ode45-Solver aufgerufen, dem das zu berechnende Zeitintervall, das Functionhandle und der y-Vektor übergeben werden. Nach der Integration gibt er die Lösung als Wertetabelle Y zu der entsprechenden Zeit T aus. Die Lösung wird dann als Plot dargestellt. Die Figure enthält die Geschwindigkeiten, die Wege der beiden Massen und zur Kontrolle das Lambda über der Zeit.

[15] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, B.G. Teubner Verlag, Wiesbaden, 2.Auflage, 2006, S.68

Bevor jedoch Ergebnisse erzielt werden können, muss die Handlefunction das Gleichungssystem zur Verfügung stellen. Das geschieht in der Funktion “Handlefun.m“. Hier werden zunächst wieder die numerisch vereinbarten Variablen aus dem Script “Load_n.m“ bezogen, um abhängig der Anzahl der Freiheitsgrade (n) die einzelnen Unterfunktionen mit dem geforderten Input zu versorgen. Nach Aufruf der Unterfunktionen wird das Gleichungssystem mit yp=B\b gebildet und dem ode45 bereitgestellt.

In der Unterfunktion “Matbuild.m“ wird nun die systembeherrschende Matrix wie in Gleichung [29] erzeugt. Dazu wird als erstes auf die symbolisch vereinbarten Variablen und die Lagrange- Funktion aus dem Script “Load_s.m“ zugegriffen. Da der charakteristische Teil der systembeherrschenden Matrix eine Jacobi Matrix ist, kann zur Bildung der doppelten Ableitungen der Lagrange- Funktion auf den “jacobian“ Befehl aus der Symbolic Toolbox zurückgegriffen werden. Der triviale Teil wird über eine for- Schleife erzeugt.

Nun wird in der Funktion “Vectorb.m“ der systembeherrschende Vektor aus Gleichung [28] erzeugt. Hierzu wird als erstes auf die Unterfunktion “Zwang.m“ zugegriffen, welche die Z_i erzeugt, welche von den Bewegungsgleichungen subtrahiert werden. Die Funktion “Zwang.m“ bezieht die Zwangsbedingung ebenfalls vom Script “Load_s.n“. Die B_i^{μ} werden hier durch Differentiation mit dem Befehl “diff“ gebildet. Die Bewegungsgleichungen werden symbolisch in der Unterfunktion “BewGl.m“ erzeugt. Nun wird in mehreren Schritten durch Umstellen und durch Substituieren der systembeherrschende Vektor aufbereitet.

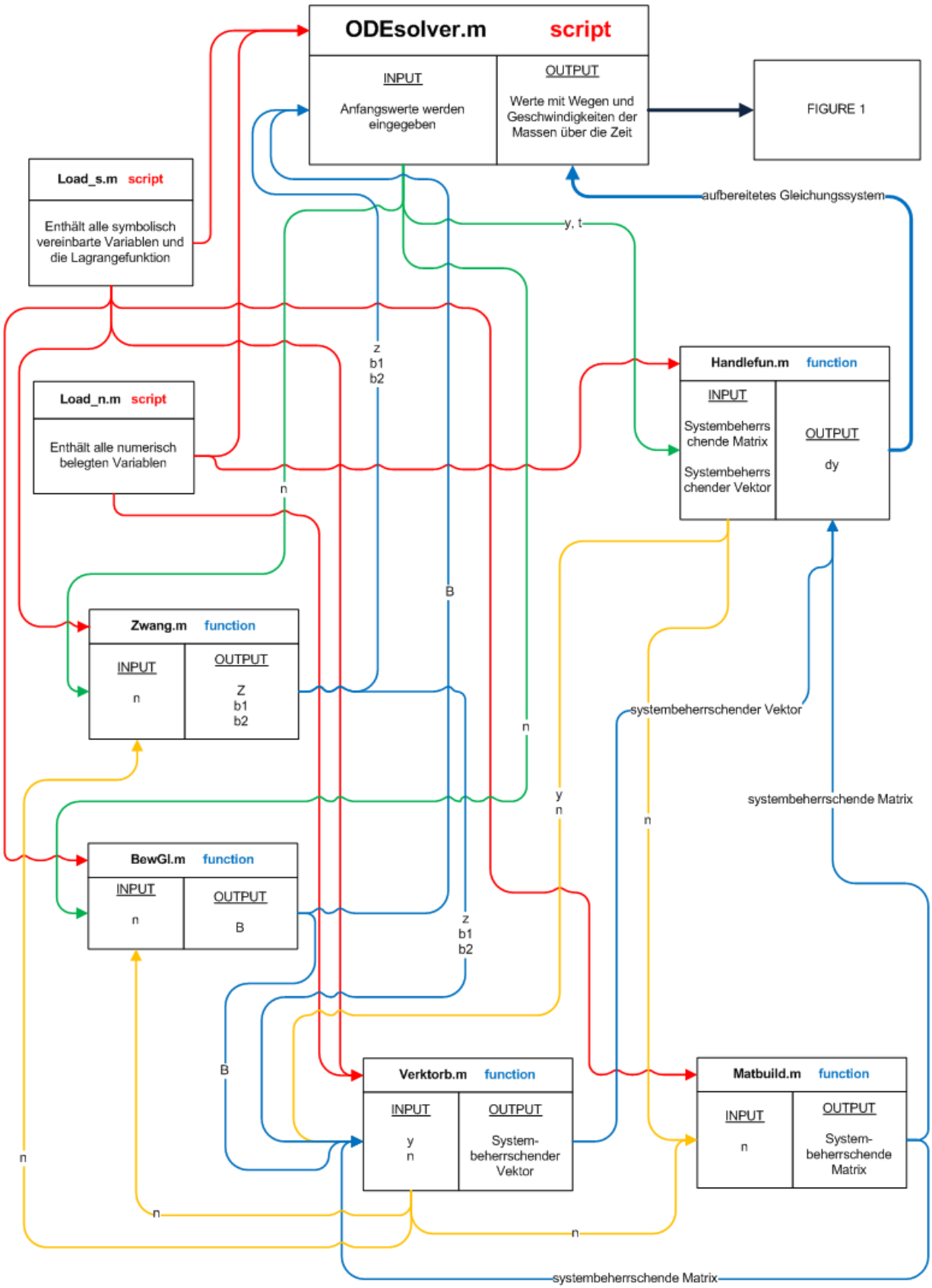


Abbildung 8: Flussdiagramm symbolischer Programmablauf

Als Lösung des Beispiels Zweimassenschwingers entsteht folgendes Diagramm:

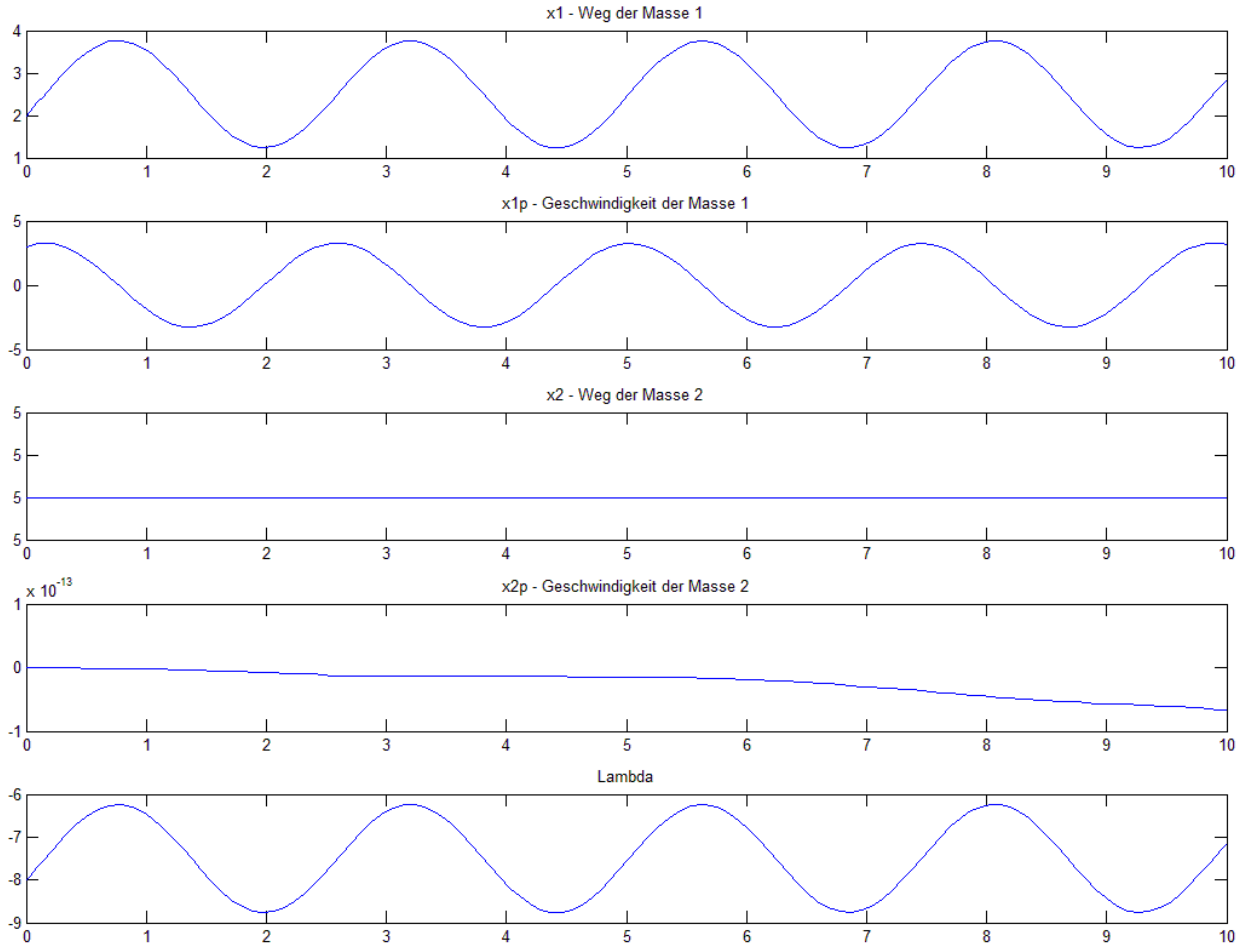


Abbildung 9: Ergebnis für die symbolische Lösung des Zweimassenschwingers

Das Diagramm Abbildung 9 zeigt plausible Ergebnisse für das betrachtete Problem. Der Weg der Masse 1 beginnt beim festgelegten Anfangswert. Das Gleiche gilt für den virtuellen Vorspannweg der Feder 3 (Lambda). Laut Zwang darf sich die Masse 2 nicht bewegen, was laut Diagramm auch nicht geschieht.

Für diese Anwendung am einfachen Beispiel scheint das symbolische Verfahren geeignet zu sein. Jedoch wird es dem universellen Anspruch nicht gerecht und es wird auch bei der Anwendung am Komplexbeispiel Rennwagen aus folgenden Gründen nicht zur Nutzung kommen.

Nachteile des symbolischen Verfahrens:

- lange Rechenzeit beim einfachen Beispiel (30 s mit derzeit durchschnittlicher Rechenleistung)
- Anfangswerte müssen im Hauptprogramm von Hand aufwendig angepasst werden
- alle symbolischen Variablen müssen von Hand deklariert werden
- an mehreren Stellen lässt sich die Substitution von symbolischen Variablen in numerische nicht automatisieren
- unübersichtlicher Programmaufbau (Abbildung 8)
- durch die symbolisch vereinbarte Lagrange Funktion, welche am Rennwagen von einer aufwendigen, numerisch arbeitenden Kinematik abhängt, wird es nicht ohne Probleme möglich sein, diese zu differenzieren
- beim Zugriff auf die Ressourcen der Symbolic Toolbox werden Build-In Funktionen aufgerufen, deren Funktionsweise und Grenzen der Anwender nicht kennt

Nun muss ein neuer Weg zur Herangehensweise gefunden werden.

5.3.2 Gewährleistung des universellen Anspruches

Um dem universellen Anspruch besser gerecht zu werden als in 5.3.1, wird nun eine neue Strategie entwickelt.

Um die Differentiationen der komplexen Kinematik des Fahrwerks erzeugen zu können, muss das Lösungsverfahren numerisch umgesetzt werden. So entfällt auch das Problem der Substitution der symbolischen- mit den numerischen Variablen. Dabei muss jedoch ein Weg gefunden werden, die Deklaration der Variablen zu automatisieren und ein geeignetes Verfahren zur numerischen Differentiation muss zur Anwendung kommen.

5.3.2.1 *Automatisierung der Variablendeklaration*

Da die Lagrange Funktion nur von Geschwindigkeiten und Wegen abhängt (siehe Gleichung [20]) und deren Anzahl der Anzahl der Freiheitsgrade, bzw. der Anzahl der generalisierten Koordinaten entspricht, können diese Variablen in einer Variablenmatrix formuliert werden. Außerdem wird in diese Matrix die Zeitvariable t mit eingehen, um die zeitlichen Ableitungen zu ermöglichen. Die Matrix besteht nun aus drei Zeilen mit N Spalten.

$$var = \begin{bmatrix} \dot{q}_1 & \dots & \dot{q}_N \\ q_1 & \dots & q_N \\ t & 0 & 0 \end{bmatrix} \qquad [42]$$

Damit ist es möglich, abhängig der Anzahl der Freiheitsgrade N alle Variablen der Lagrange Funktion zuzuordnen. Die Nachteile der symbolischen Variante sind nicht mehr vorhanden und eine Automatisierung ist leicht möglich.

Die Umsetzung in MATLAB® bringt folgende Funktion hervor.

```
function[var]=Variablen(t,y)
global n;
var=zeros(3,n);
for i=1:n
    var(1,i)=y(2*i);
end
for i=1:n
    var(2,i)=y(2*i-1);
end
var(3,1)=t;
var(3,2:n)=0;
```

Die Funktion Namens “Variablen.m“ erhält den y- Vektor, welcher alle Geschwindigkeiten und Wege der generalisierten Koordinaten zum Zeitpunkt t enthält, als Input. Die globale Variable n steht für die Anzahl der generalisierten Koordinaten. Mit dem Befehl “zeros“ wird eine Matrix erzeugt, welche drei Zeilen und n Spalten aufweist. In der ersten for-Schleife wird nun die erste Zeile der Matrix mit den Geschwindigkeiten bestückt. Die Geschwindigkeiten entsprechen den geraden Komponenten des y- Vektors. Die zweite Zeile wird, ebenfalls mit Hilfe einer for-Schleife, mit den ungeraden y- Komponenten, den Wegen, bestückt. Die letzte Zeile erhält nun in der ersten Spalte den Zeitpunkt t, der Rest wird mit Nullen ergänzt. Die fertig entwickelte Matrix “var“ ist nun der Output der Funktion.

5.3.2.2 *Numerische Differentiation*

Zur Differentiation wird sich der Theorie des Differenzenquotienten bedient. Diese besagt laut Definition:

$$f' = \frac{f(x_0+(x_1-x_0))-f(x_0)}{(x_1-x_0)} \qquad [43]$$

Zur Veranschaulichung dient folgende Abbildung:

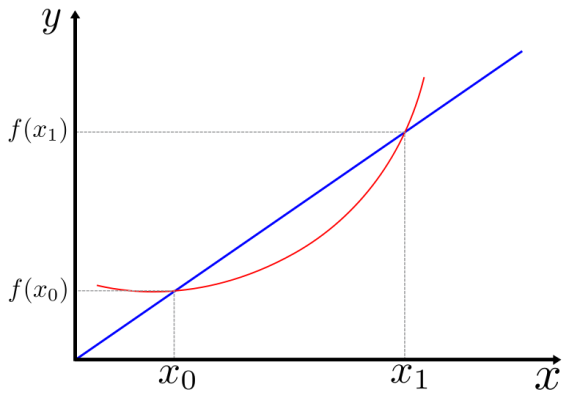


Abbildung 10: Differenzenquotient[16]

[16] http://de.wikipedia.org/w/index.php?title=Datei:Afgeleide.svg&filetimestamp=20110410115333, verfügbar am 04.09.2011, 17:00 Uhr

Der Differenzenquotient entspricht der blauen Gerade in Abbildung 10, also dem Anstieg der Funktion im Intervall [x_0 x_1]. Je kleiner die Differenz $h=x_1-x_0$ ist, umso genauer wird der Differenzenquotient an der gewählten Stelle.

In MATLAB® wird dazu eine allgemein anwendbare Funktion implementiert, die auf der Formel [43] beruht.

```
function [returnval] = diff_1(func,number1,number2,arg,step1)

% Funktionswert an der Stelle
res1=func(arg);

% Funktionswert ein wenig weiter
temp=arg(number1,number2)+step1;
arg(number1,number2)=temp;

% Funktionswert rechnen
res2=func(arg);

% Differenzenquotient zurück
returnval=(res2-res1)/step1;
```

Als Input erhält die Funktion “diff_1.m“ die Funktion “func“ als Handle übergeben. Die Wahl, nach welcher Variablen abgeleitet werden soll, erfolgt über “number1“ und “number2“. Die beiden Nummern stehen für die Zeile und Spalte der Variablenmatrix aus 5.3.2.1. Die Variable “arg“ steht für die Variablenmatrix. “step1“ enthält die Schrittweite in der der Differenzenquotient gebildet werden soll. Als erstes wird der Funktionswert an der Stelle x_0 gebildet. Danach wird zum Argument die Schrittweite addiert und der Funktionswert bei x_1 berechnet. Als letztes wird der Differenzenquotient gebildet und als Output der übergeordneten Funktion übergeben.

5.3.3 Numerischer Programmablauf

Mit den unter 5.3.2 eingeführten Strategien lässt sich nun ein rein numerischer Ablauf zur Lösung dynamischer Probleme realisieren. Die Quelltexte zum Grundprogramm befinden sich im Anhang, Teil 3. Zur anschaulicheren Erläuterung dient zusätzlich die Abbildung 11.

5.3.3.1 *Das Hauptprogramm*

Als Hauptprogramm dient wieder das Skript “ODEsolver.m“. Hier werden wiederum die Anfangswerte festgelegt, der ode45-Solver wird aufgerufen und die Ausgabe der Ergebnisse wird organisiert. Zusätzlich werden noch globale Variablen vereinbart. Das sind die Anzahl der Freiheitsgrade n und die Anzahl der Zwänge u. Durch die Globalisierung dieser Variablen können sie in jeder Unterfunktion bereitgestellt werden. Für das Beispiel “Zweimassenschwinger“ aus 4.2 lauten n=2 und u=1. Die Anfangswerte sind genauso gewählt wie in 4.2. Das Zeitintervall beträgt wie bei der symbolischen Untersuchung 10 s. Neben den Diagrammen werden noch die Datensätze über den “csvwrite“ Befehl in Tabellenform ausgegeben, um später für eine Bewegungsstudie in SolidWorks® genutzt zu

werden. Diese Tabellen beinhalten die Wege der generalisierten Koordinaten zum jeweiligen Zeitpunkt.

5.3.3.2 *Die Handle Funktion*

In der vom Hauptprogramm aufgerufenen Functionhandle “Handlefun.m“ wird nun das zu lösende Gleichungssystem [41] aufbereitet. Dazu werden, wie Abbildung 11 zeigt, nacheinander die Unterfunktionen “Matbuild_n.m“,“Vektorb_n.m“ und “AufZwang.m“ aufgerufen. Der erzeugte Vektor dy wird dem ode45-Solver zur Lösung bereitgestellt.

5.3.3.3 *Das Aufstellen der systembeherrschenden Matrix*

Nun wird die systembeherrschende Matrix in der Funktion “Matbuild_n.m“ erzeugt. Dazu wird als erstes die Variablenmatrix var durch den Aufruf der Funktion “Variablen.m“, abhängig vom y Vektor und dem Zeitpunkt t, wie unter 5.3.2.1 besprochen, berechnet.

Die Matrix selbst wird in einer doppelten for-Schleife bestückt. Die quadratischen Ableitungen werden nach dem Schema aus Gleichung [26] berechnet und abwechselnd in die geraden und ungeraden Spalten der Matrix eingesetzt.

Die quadratischen Ableitungen werden in der “diff_2.m“ Funktion gebildet. Ihr werden das Functionhandle (“Lagrange.m“), welches abgeleitet werden soll, die beiden Zuweisungen der Variablen der Variablenmatrix, nach denen nacheinander differenziert werden soll, die Variablenmatrix selbst und die beiden Schrittweiten der Differentiationen als Input übergeben. Die einzelnen Ableitungen erfolgen über das unter 5.3.2.2 besprochene Verfahren der “diff_1.m“ Funktion. Als erstes erfolgt die Berechnung der ersten Ableitung der Lagrange Funktion zur Bestimmung des Funktionswerte an der Stelle x_0 (Abbildung 10). Nun werden die Komponenten des Argumentes, nach denen differenziert werden soll, mit der gewünschten Schrittweite addiert und erneut mit unter der “diff_1.m“ Funktion differenziert. Damit erhält man den Funktionswert an der Stelle x_1 (Abbildung 10). Als letztes wird der Differenzenquotient nach der Gleichung [43] gebildet.

Die Funktion “Lagrange.m“ hängt laut Vereinbarung [20] nur von den Geschwindigkeiten und Wegen ab. Hier kann der Anwender die zu seinem zu betrachtenden System gehörende Lagrange Funktion einbauen (im Beispiel Gleichung [15]). Um die Energiebilanz zu komplettieren müssen noch die Konstanten deklariert werden. Hier sind das die Masse m und die Federsteifigkeit c. Die Energiedifferenz des jeweiligen Zeitpunktes wird nun der “diff_1.m“ Funktion zur Differentiation zur Verfügung gestellt.

Der triviale Teil der systembeherrschenden Matrix muss nun noch an den nicht trivialen Teil angefügt werden. Die Konstruktion des trivialen Teils geschieht in der “Matbuild_n.m“ Funktion durch eine for-Schleife. Zum Schluss wird die fertig aufbereitete systembeherrschende Matrix der “Handlefun.m“ Funktion zur Verfügung gestellt.

5.3.3.4 *Das Aufstellen des systembeherrschenden Vektors*

Der systembeherrschende Vektor wird nach der Gleichung [22] in der Funktion “Vektorb_n.m“ gebildet. Nach dem Aufruf der Funktion “Variablen.m“ wird die Lagrange Funktion in der “diff_1.m“ Funktion nach den Wegen differenziert. Durch die Zuweisung der zweiten Zeile der Variablenmatrix kann an der richtigen Variablen die Schrittweite addiert werden. Nach erfolgter Differentiation wird der systembeherrschende Vektor wie in [28] um die Geschwindigkeiten erweitert. Der endgültige systembeherrschende Vektor wird nun der “Handlefun.m“ Funktion übergeben.

5.3.3.5 *Die Aufbereitung der Zwänge*

Die Zwangaufbereitung erfolgt im Grundprogramm in der Funktion “AufZwang.m“ nach dem Schema aus 5.2.2. Zunächst werden die globalen Variablen u und n geladen, nachdem die Variablenmatrix aufgebaut wird. Zur Automatisierung der Zwang Funktionen wird ein globaler Vektor “handles“ angelegt, der alle festgelegten Zwänge enthält. So kann später die Berechnung der $\underline{\underline{B_i^\mu}}$ Matrix und des $\underline{B_0^\mu}$ Vektors problemlos automatisiert werden.

Die Zwangsfunktionen werden in einzelnen Funktionen, welche die selbe Charakteristik wie die Beispielfunktion “Zwang_1.m“ haben, vom Anwender festgelegt. Die Zwänge dürfen, um holonom zu sein, nach 4.1.1 nur von der Zeit oder von Wegen abhängen. Jedoch niemals von der Geschwindigkeit. Die erforderlichen Konstanten muss der Anwender selbst festlegen. Die Zwangsfunktion selbst muss, wie im Beispiel, explizit aufgestellt werden. Im Beispiel entspricht sie der Gleichung [10].

Die Matrix $\underline{\underline{B_i^\mu}}$ aus Gleichung [32] wird in einer doppelten for-Schleife mit Hilfe der Funktion “Bcreate.m“ erstellt. Die Hilfsfunktion “Bcreate.m“ bestimmt mit der Laufvariable mue, nach welchem Zwang in der “diff_1.m“ Funktion abgeleitet wird. Hier wird nämlich nun auf den handles-Vektor zugegriffen, der die Zwänge zum Ableiten nach den Wegen bereitstellt. Die erzeugte Matrix lautet im Quelltext Biu. Der Vektor $\underline{B_0^\mu}$ (Gleichung [33]) wird auf dem gleichen Weg erzeugt, nur das hier nach der Zeit differenziert wird (dritte Zeile, erste Spate der Variablenmatrix). Im Quelltext wird der Vektor Bou genannt.

Die zeitlichen Ableitungen der eben erstellten $\underline{\underline{B_i^\mu}}$ und $\underline{B_0^\mu}$ müssen nach [34] und [35] nun erbracht werden. Zum Berechnen der partiellen Ableitungen wird sich folgender mathematischer Grundlage bedient:

$$\dot{B}_i^\mu = \frac{\partial B_i^\mu}{\partial q_i} * \dot{q}_i \qquad [44]$$

Zur Differentiation wir sich nun der Funktion “diff_B.m“ bedient. Im Unterschied zur “diff_1.m“ Funktion werden hier zusätzlich die Laufvariable mue und die Information, nach welchen Geschwindigkeiten differenziert werden soll übergeben. Das Functionhandle ist in diesem Fall die Funktion “Bcreate.m“. Damit ist auch bei diesem Berechnungsschritt die

geforderte Automation gewährleistet. Durch die Multiplikation mit den jeweiligen Geschwindigkeiten, der Aufsummierung und der Addition mit den zeitlichen Ableitungen erfüllt $\underline{\dot{B}}_i^{\mu}$ die Gleichung [34]. Im Quelltext wird die Matrix Biup genannt. Für $\underline{\dot{B}}_0^{\mu}$ gilt das Verfahren analog, nur eben wird nach der Zeit differenziert (im Quelltext Boup).

Nun wird der untere Teil des Vektors auf der rechten Seite des Gleichungssystems [40] mit den berechneten Biup und Boup gebaut, indem die Biup mit den Geschwindigkeiten der Variablenmatrix multipliziert und summiert werden. Danach wird das Vorzeichen des erzeugten Vektors invertiert um davon den Boup Vektor zu subtrahieren.

Auf die Funktion “Matsort.m“ wird hier nicht eingegangen. Ausführungen hierzu finden sich im kommentierten Quelltext.

Nach Aufruf und Anpassung der systembeherrschenden Matrix und des systembeherrschenden Vektors wird das Gleichungssystem [40] konstruiert.

Der zu übergebende Vektor mit den Beschleunigungen der generalisierten Koordinaten und den gesuchten Lambdas wird durch die “Linksdivision“ gewonnen und der Handle Funktion zur Verfügung gestellt.

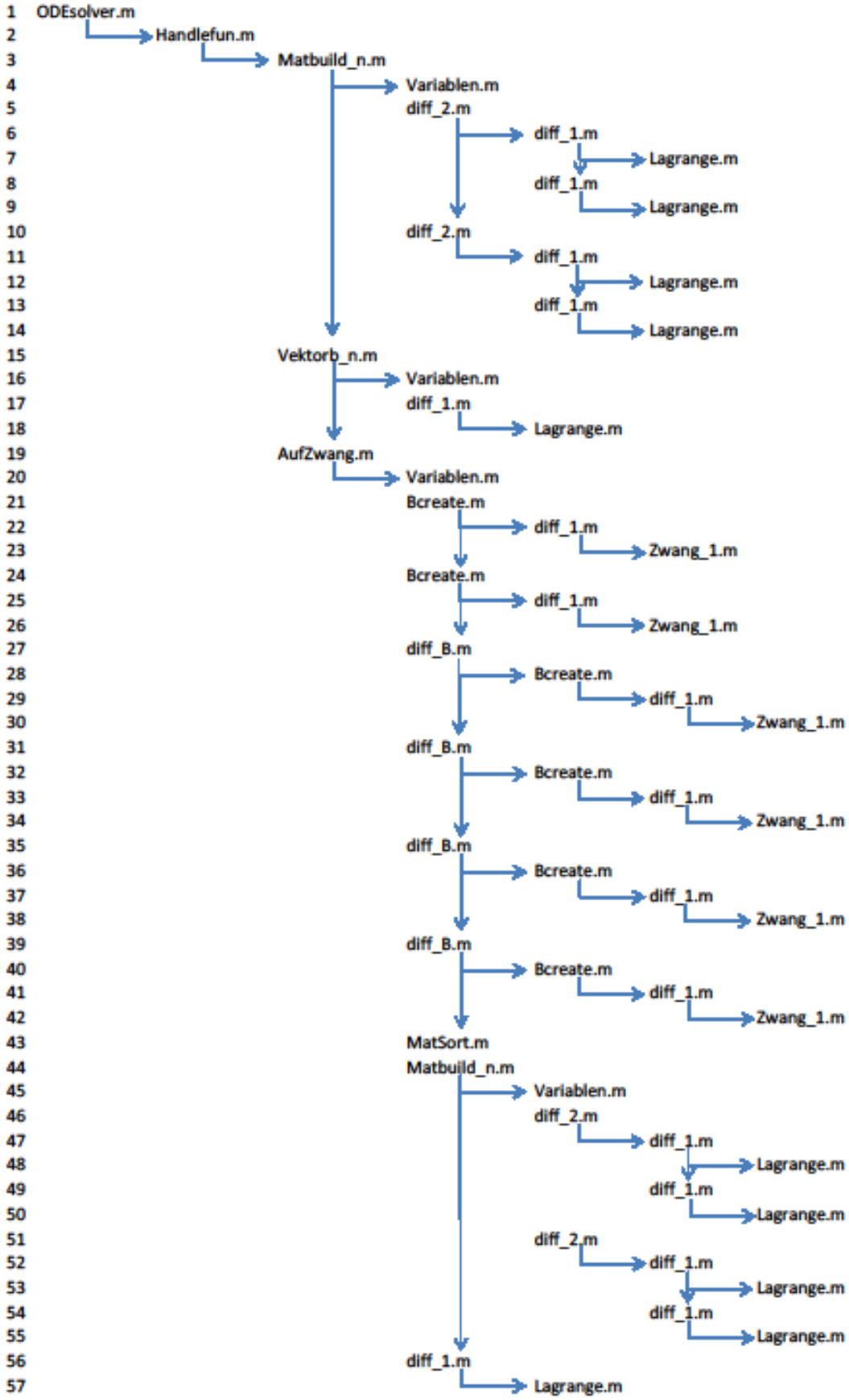


Abbildung 11: Hierarchie des Grundprogramms

5.3.4 Inbetriebnahme und Visualisierung des Beispiels im Grundprogramm

Unter 5.3.3 wurde der Programmaufbau des Grundprogramms am Beispiel Zweimassenschwinger gezeigt. Nun wird zusammengefasst, welche Schritte zum Anpassen des Grundprogramms auf ein beliebiges dynamisches System notwendig sind.

1. in der Funktion “ODEsolver.m“ die Anzahl der Freiheitsgrade n und die Anzahl der Zwänge u festlegen (hier n=2, u=1)
2. sinnvoll gewählte Anfangswerte vorgeben (müssen Gleichung [4] entsprechen/ siehe Quelltext in Anlage, Teil 3)
3. beim Aufruf des ode45-Solver das gewünschte Zeitintervall angeben (hier von 0 bis 10 s)
4. die gewünschten Ausgabeformate der Ergebnisse festlegen

5. in den “Zwang_X.m“ Funktionen die gewünschten Zwänge in expliziter Form und eventuell benötigte Konstanten festlegen (hier `f1=Fc-c*q(2)` aus [10] in “Zwang_1.m“)

6. in “Lagrange.m“ die von q_i und $\dot{q}_i$ abhängige Lagrange Funktion aufstellen und benötigte Konstanten definieren

Nach Vollziehen dieser sechs Schritte wird durch die Eingabe von “ODEsolver“ in das Command Window in MATLAB® die Berechnung gestartet.

Nach Ablauf der Berechnung erscheinen für das Beispiel folgende Diagramme:

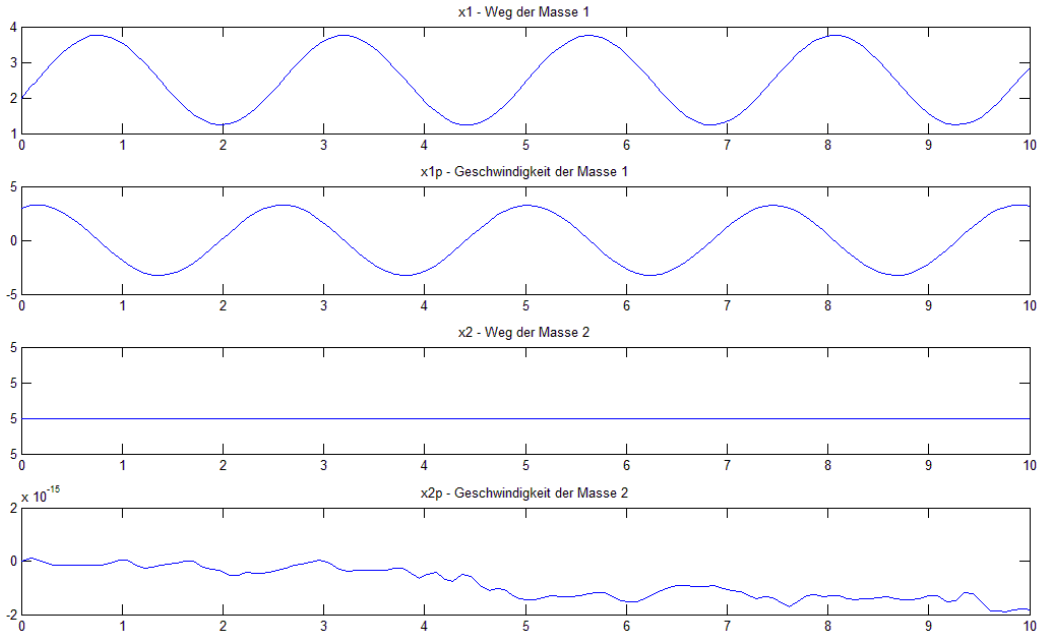


Abbildung 12: Ergebnis für die numerische Lösung des Zweimassenschwingers

Die Ergebnisse entsprechen den symbolisch Erzeugten aus Abbildung 9. Die Schwingungen der linken Masse beginnen beim gleichen Anfangswert, haben die gleiche Amplitude und Frequenz. Die rechte Masse steht still. Die minimale Geschwindigkeit im Bereich von 10e-15 entsteht durch die numerische Differentiation, ist dabei aber noch genauer als beim symbolischen Verfahren (10e-13), was zusätzlich für die Verwendung des numerischen Verfahrens spricht.

Zur visuellen Veranschaulichung und Nachvollziehbarkeit der Ergebnisse werden nun die Datensätze, welche in der Funktion “ODEsolver.m“ mit dem Befehl “csvwrite“ erzeugt wurden, weiterverwendet. Die Datensätze enthalten die berechneten q_i zum jeweiligen Zeitpunkt t. Mit diesen Datensätzen lässt sich in einer SolidWorks® Baugruppendatei eine Bewegungsstudie ausführen. Dazu müssen die Systemkomponenten konstruiert und in eine Baugruppe geladen werden. Um eine Bewegungsstudie zu starten, klickt man bei geöffneter Baugruppe unten links auf “Bewegungsstudie“ (siehe Abbildung 13).

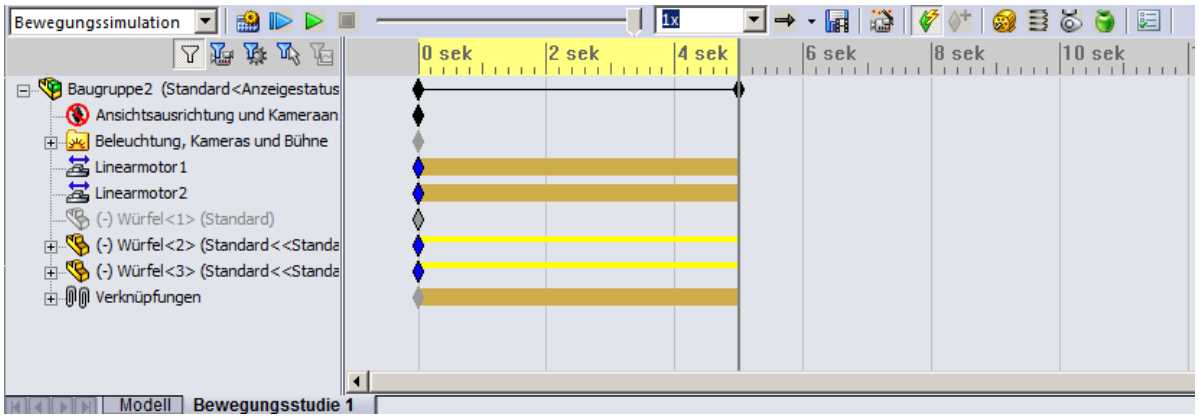


Abbildung 13: Einrichten einer Bewegungsstudie

Dabei öffnet sich das Fenster Abbildung 13. Von diesem Fenster aus kann nun die dynamische Visualisierung des Systems gesteuert werden. Für das Beispiel Zweimassen wurden drei Würfel erstellt. Der Erste dient als fixer Bezugskörper, auf den sich die beiden anderen losen Würfel referenzieren. Die losen Würfel stellen nun die beiden Massen des Zweimassenschwingers dar. Da der Erste nur zum Referenzieren dient, kann er unterdrückt werden, um nicht zu irritieren. In Abbildung 14 wird nun die Menüführung zur Übertragung der Datensätze auf die Würfel gezeigt. Nach dem Klicken auf den gelben Elektromotor (Abbildung 13 oben rechts) öffnet sich das Menü. Hier wird nun für jeden Körper der “Motor“ zugewiesen, die Richtung bestimmt (muss der festgelegten Richtung der jeweiligen generalisierten Koordinate entsprechen), der Bezugskörper ausgewählt und die Datensätze eingeladen. Zu Letzterem empfiehlt es sich, die “Bewegung“ als “Interpolierung“ zu wählen. Der Datensatz enthält die Verschiebungen, also ist dies auszuwählen. Als Interpolationstyp ist es ausreichend, “Linear“ anzuklicken. Nun kann die entsprechende erzeugte dat-Datei ausgewählt werden. SolidWorks® lädt nun den Datensatz für den entsprechenden Körper ein. Wenn das geschehen ist, lässt sich durch Klicken auf das grüne Dreieck (Abbildung 13) die Bewegungsstudie starten. Wenn sich nun die Körper bewegen, kann die Plausibilität der Ergebnisse aus der Berechnung in MATLAB® besser abgeschätzt werden.

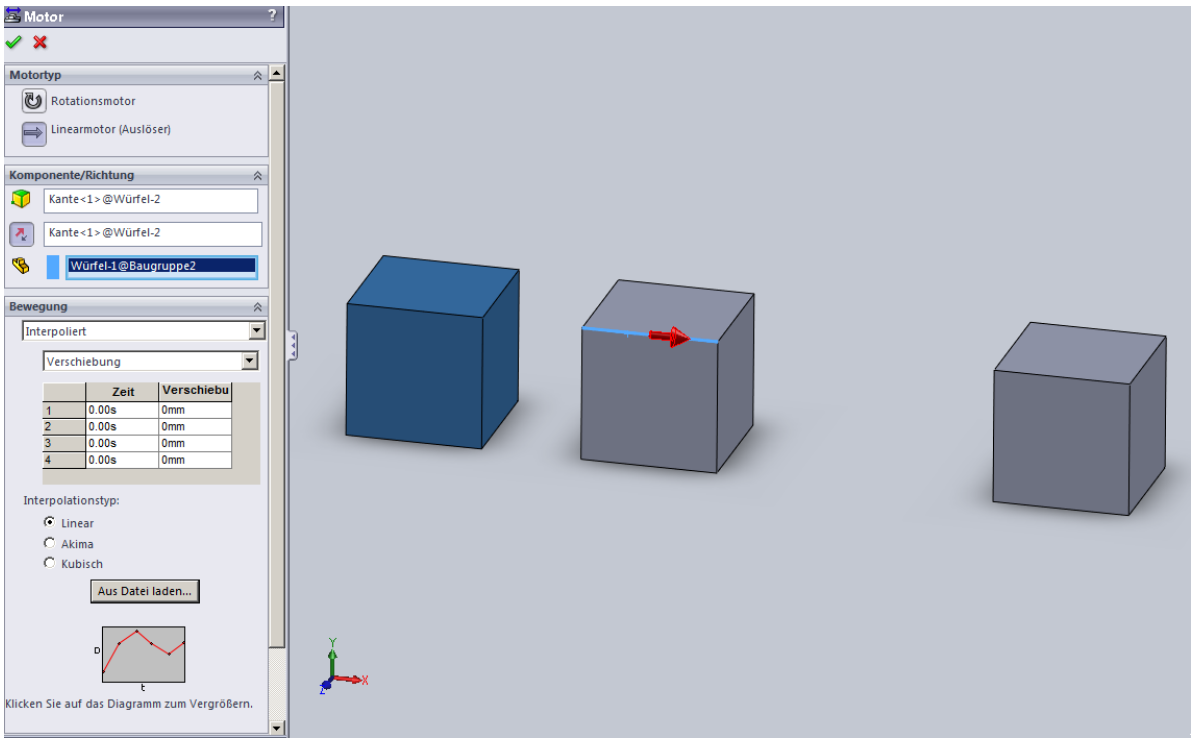


Abbildung 14: Menüführung zur Datenübertragung

Die animierte Bewegungsstudie lässt sich auch als avi-Datei festhalten, um unabhängig von SolidWorks® betrachtet werden zu können.

5.3.5 Weiter Beispiele zum Testen der Leistungsfähigkeit des Grundprogramms

Um die Leistungsfähigkeit und Richtigkeit des Grundprogramms zu prüfen, wurden verschiedene Untersuchungen durchgeführt, bevor das Programm am Fahrzeug Anwendung findet.

5.3.5.1 *Zweimassenschwinger ohne Zwang*

Hier wird überprüft, ob das Grundprogramm ohne das Festlegen von Zwängen funktionsfähig ist. Dazu wird n=2 und u=0 gesetzt. Die Anfangswerte werden willkürlich wie folgt festgelegt.

```
y(1)=2;         %   Vorspannweg von x1
y(2)=3;         %   Anfangsgeschwindigkeit von x1
y(3)=5;         %   Vorspannweg von x2
y(4)=0;         %   Anfangsgeschwindigkeit von x2
```

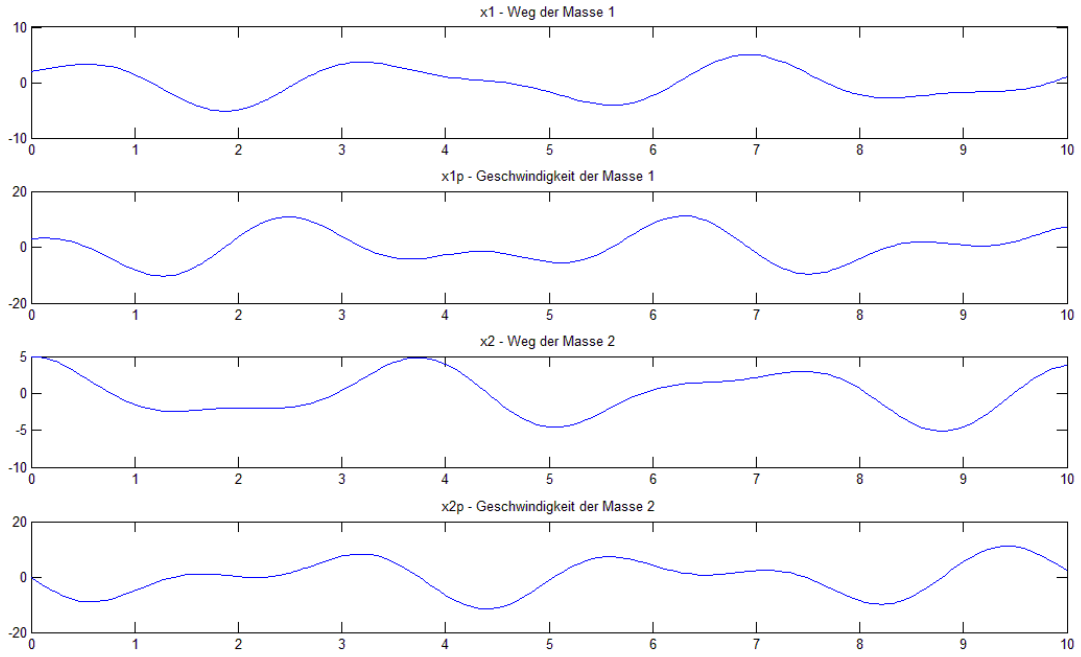


Abbildung 15: Zweimassenschwinger ohne Zwang

Die entstandenen Diagramme in Abbildung 15 zeigen die Anfangswerte entsprechend der Festlegung. Die Schwingungen weisen keine bestimmte Periodendauer auf. Das rührt daher, dass sich die beiden Schwingungen der Massen gegenseitig beeinflussen. Das Grundprogramm funktioniert demnach auch an dynamischen Systemen ohne Zwänge.

5.3.5.2 *Dreimassenschwinger mit zwei Zwängen*

Um das System zu verkomplizieren und um das Programm mit mehreren Zwängen zu testen, wird nun ein Dreimassenschwinger untersucht. Hierbei beträgt die Anzahl der Freiheitsgrade nun n=3 und die Anzahl der Zwänge u=2. Die Lagrange Funktion wird nun etwas aufwendiger:

```
Lag=0.5*m*(qp(1))^2+0.5*m*(qp(2))^2+0.5*m*(qp(3))^2-
0.5*c*((q(1))^2+(q(2)-q(1))^2+(q(3)-q(2))^2+q(3)^2);
```

Der erste Zwang ist der bekannte Zwang aus den vorherigen Beispielen. Er sorgt dafür, dass die Masse ganz rechts fest steht: `f1=Fc-c*q(3)`. Der zweite Zwang soll nun bewirken, dass die anderen beiden Massen starr verbunden sind und dadurch gleich schwingen: `f2=q(1)-q(2)`.

Bei der Festlegung der Anfangswerte wird nun nur die erste Masse ausgelenkt.

```
y(1)=3;         %   Vorspannweg von x1
y(2)=0;         %   Anfangsgeschwindigkeit von x1
y(3)=0;         %   Vorspannweg von x2
y(4)=0;         %   Anfangsgeschwindigkeit von x2
y(5)=5;         %   Vorspannweg von x3
y(6)=0;         %   Anfangsgeschwindigkeit von x3
```

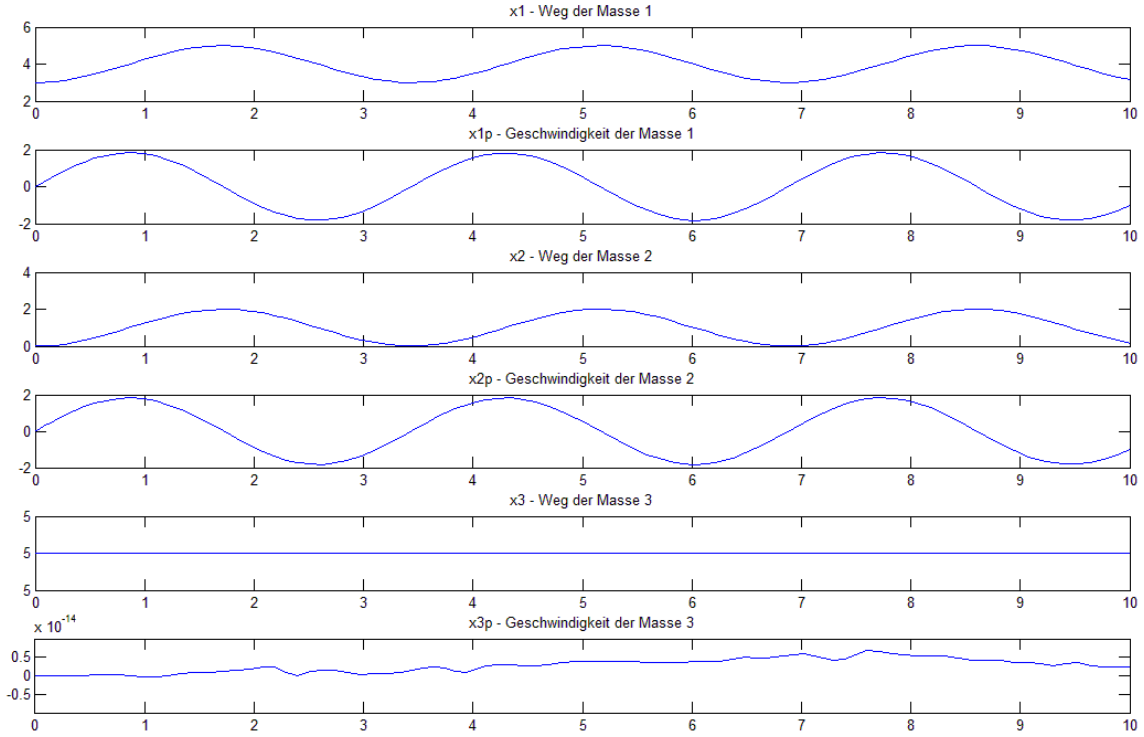


Abbildung 16: Dreimassenschwinger mit zwei Zwängen

Die Diagramme in Abbildung 16 zeigen eindeutig, dass die erwarteten Ergebnisse eintreten. Die Kurven starten bei den vorgegebenen Anfangswerten und die ersten beiden Massen schwingen mit der gleichen Frequenz und Amplitude ohne Phasenverschiebung. Die Masse 3 steht dabei, wie im Zwang 1 festgelegt, still.

5.3.5.3 *Dreimassenschwinger mit zeitabhängigem Zwang*

Als Letztes wird noch ein zeitabhängiger Zwang getestet. Getestet wird unter den gleichen Bedingungen wie in 5.3.5.2, nur dass der zweite Zwang zeitabhängig gemacht wird, indem zur Wegvariable der Masse 2 die Geschwindigkeit quadratisch multipliziert wird: `f2=q(1)-(t^2)*q(2)`. Dadurch sollte die Schwingung der mittleren Masse mit der Zeit abklingen, da nach Gleichung [10] die Zwangsgleichungen explizit nach 0 aufgestellt werden. Das Quadrat wird das Abklingen noch weiter beschleunigen. Die Anfangswerte zu diesem Test lauten:

```
y(1)=0;         %   Vorspannweg von x1
y(2)=0;         %   Anfangsgeschwindigkeit von x1
y(3)=0;         %   Vorspannweg von x2
y(4)=2;         %   Anfangsgeschwindigkeit von x2
y(5)=5;         %   Vorspannweg von x3
y(6)=0;         %   Anfangsgeschwindigkeit von x3
```

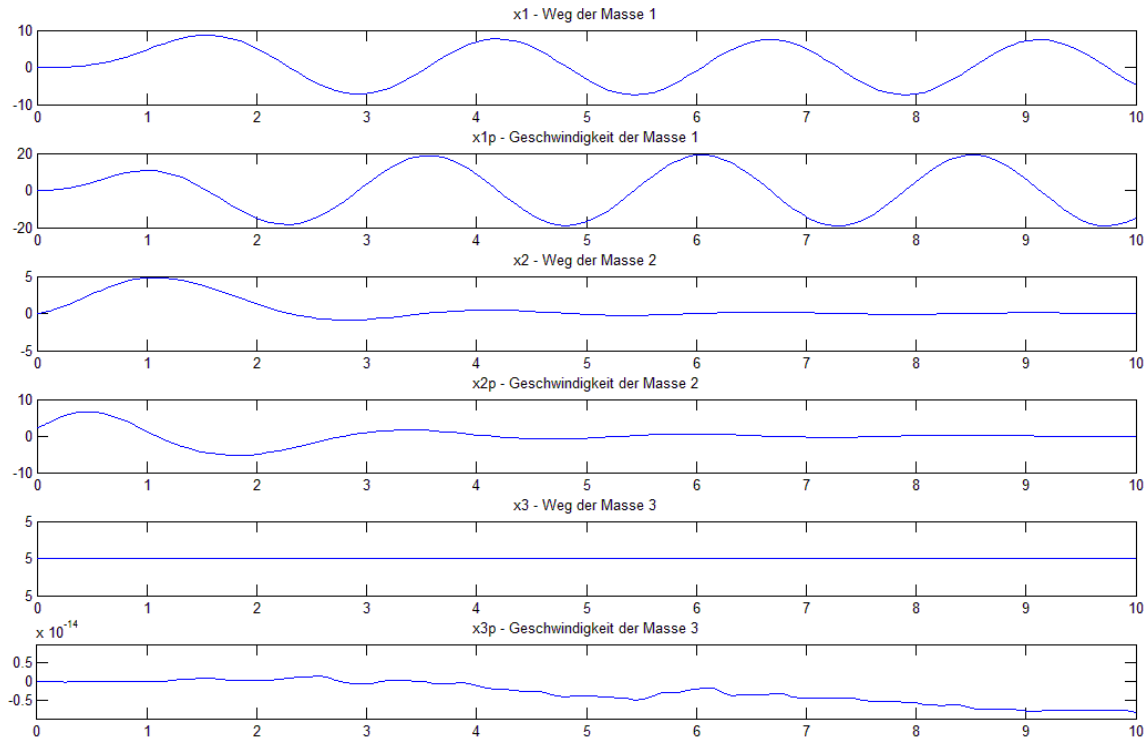


Abbildung 17: Dreimassenschwinger mit Zeitabhängigem Zwang

Auch in den hier (Abbildung 17) gezeigten Diagrammen werden die Erwartungen erfüllt. Die Kurven beginnen bei den festgelegten Anfangswerten, die Masse 3 bleibt fest stehen, die Masse 1 schwingt und die Amplitude der Masse 2 verringert sich über die Zeit nahezu quadratisch.

5.4 Ergebnisse

Die unter 5.3.5 durchgeführten Tests wurden zusätzlich mit dem unter 5.3.4 behandelten Verfahren in SolidWorks® visualisiert und für plausibel befunden.

Damit ist die erfolgreiche Umsetzung eines Analyseverfahrens für dynamische Systeme in MATLAB® gezeigt worden. Dabei genügt das entstandene Grundprogramm auch dem universellen Anspruch.

Das Grundprogramm hat durch seine gezeigte Leistungsfähigkeit einen hohen Wert bei der Untersuchung dynamisch belasteter Systeme, da das Programm recht problemlos auf die unterschiedlichsten Systeme angepasst werden kann und die Ergebnisse für den weiteren Gebrauch gut nutzbar sind.

Als Nächstes kann das Komplexbeispiel “Rennwagen“ mit dem Grundprogramm untersucht werden.

6 Anwendung am Rennwagen

In diesem Kapitel kommt nun das in Kapitel 5 entwickelte Grundprogramm am Rennwagen zur Anwendung. Es wird eine Herangehensweise an dieses Komplexproblem erarbeitet, die zur Lösung führen soll.

Die Untersuchung soll Erkenntnisse zur Anwendbarkeit des Grundprogramms am Komplexbeispiel bringen. Dabei soll insbesondere geklärt werden, ob die Nutzung der numerischen Differentiation an einem solch aufwendigen Beispiel geeignet ist. Ob der universelle Charakter des Grundprogramms auch hier ausreicht oder ob aufwendige Veränderungen und Anpassungen notwendig sind, muss geklärt werden.

6.1 Anwendung am Grundprogramm

Vor der Durchführung von Berechnungen am Rennwagen muss ein Systemmodell aufgebaut werden, auf das sich die Berechnungen stützen. Hierzu werden die sechs Schritte aus 5.3.4 abgearbeitet. Der Quelltext der angepassten bzw. der neuen M-Files befindet sich im Anhang, Teil 4.

6.1.1 Modellaufbau

Der Rennwagen wird zur Modellbildung auf fünf Massen reduziert. Das sind die vier Räder mit Radträger und der Rumpf. Diese Massen werden als Massepunkte im Schwerpunkt der Komponenten angesiedelt. Zusätzlich erhalten die Komponenten ihren aus dem CAD ermittelten Trägheitstensor in Bezug auf ihr lokales Koordinatensystem (Abbildung 18). Die Aufhängungsteile wie Querlenker, Zugstange und Feder/Dämpfereinheit werden vernachlässigt, da sie nur einen Bruchteil der Fahrzeuggesamtmasse ausmachen und das Modell unnötig verkomplizieren würden. Der Rumpf wird als unendlich steifer Starrkörper angenommen.

Wie Abbildung 18 zeigt, befindet sich der Ursprung des globalen Koordinatensystems auf Fahrbahnniveau zwischen den Radaufstandspunkten der Vorderachse. Hier gilt damit wieder die gleiche Koordinatisierung wie bei der Berechnung Kinematik in Kapitel 3. Somit kann die dynamische Berechnung des Fahrzeugs darauf aufbauen.

Die generalisierten Koordinaten des Systems entsprechen dem Input des Hauptprogramms der Kinematik (FWkomplett.m). Damit kann für das Komplexbeispiel festgelegt werden: N=10. Um erst einmal die Lauffähigkeit das Analyseverfahren am Rennwagen zu testen, erfolgt eine Betrachtung mit Zwangsbedingungen hier noch nicht. Deswegen ist Z=0.

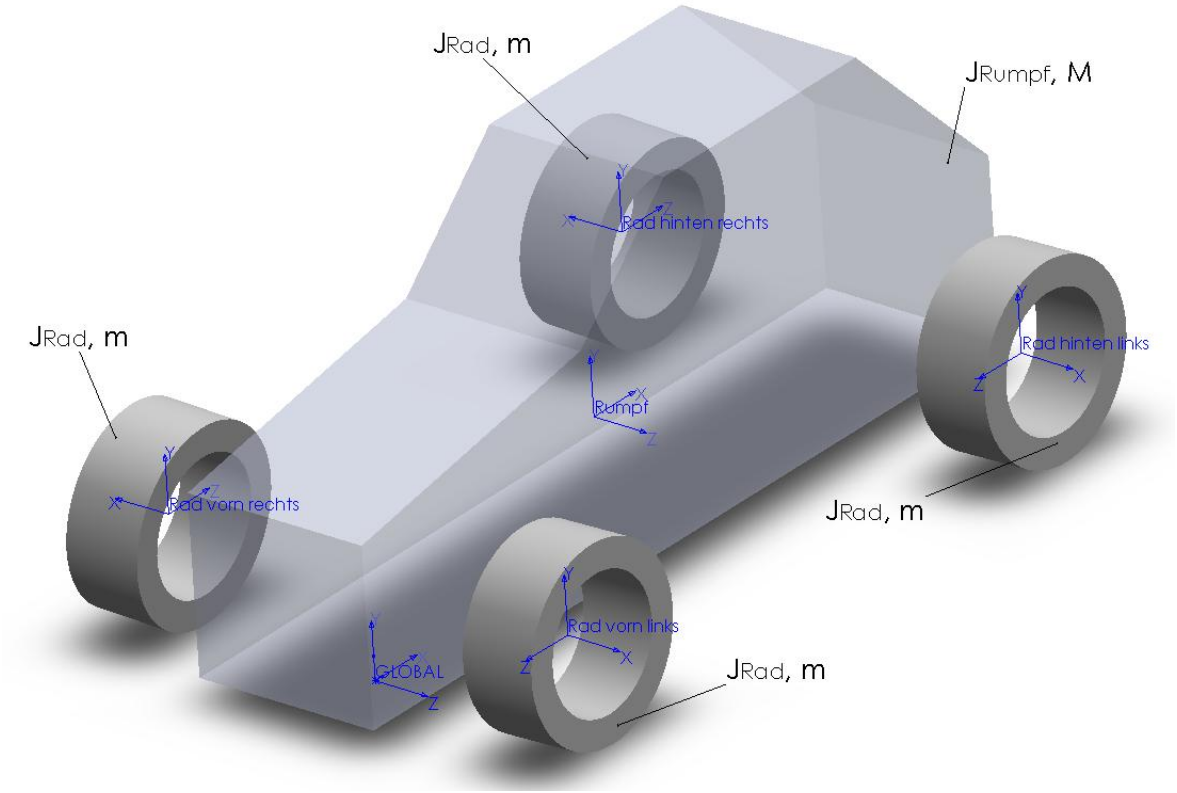


Abbildung 18: Berechnungsmodell des Rennwagens

Für diese erste Berechnung werden die Anfangswerte wie folgt gewählt:

```
y(1)=160;      %   Länge der Feder vorn links
y(2)=0;        %   Anfangsgeschwindigkeit der Feder vorn links
y(3)=188.66;   %   Länge der Feder hinten links
y(4)=0;        %   Anfangsgeschwindigkeit der Feder hinten links
y(5)=188.66;   %   Länge der Feder vorn rechts
y(6)=0;        %   Anfangsgeschwindigkeit der Feder vorn rechts
y(7)=188.66;   %   Länge der Feder hinten rechts
y(8)=0;        %   Anfangsgeschwindigkeit der Feder hinten rechts
y(9)=0;        %   Translation in X-Richtung
y(10)=0;       %   Translationsgeschwindigkeit in X-Richtung
y(11)=0;       %   Translation in Y-Richtung
y(12)=0;       %   Translationsgeschwindigkeit in Y-Richtung
y(13)=0;       %   Translation in Z-Richtung
y(14)=0;       %   Translationsgeschwindigkeit in Z-Richtung
y(15)=0;       %   Rotationswinkel um X-Achse
y(16)=0;       %   Winkelgeschwindigkeit um X-Achse
y(17)=0;       %   Rotationswinkel um Y-Achse
y(18)=0;       %   Winkelgeschwindigkeit um Y-Achse
y(19)=0;       %   Rotationswinkel um Z-Achse
y(20)=0;       %   Winkelgeschwindigkeit um Z-Achse
```

Um den ersten Test nachvollziehbar zu gestalten, wird nur die Feder vorne links gespannt, wodurch sich das Rad in einer höheren Position befindet. Die restlichen Räder bleiben in Ausgangslage. Dadurch sollte sich das Rad aufschwingen und durch die fehlenden Zwänge den Rumpf und somit auch die restlichen Räder in eine ungedämpfte Schwingung versetzen.

Das Schwingverhalten soll über einen längeren Zeitraum ausgewertet werden, um ein kontinuierliches Aufschwingen der Komponenten ausschließen zu können. Das Zeitintervall wird deswegen mit [0 35s] angesetzt.

Zur Auswertung werden Diagramme von allen q_i und $\dot{q}_i$ über die Zeit ausgegeben. Zusätzlich werden Datensätze erstellt, welche die q_i zum jeweiligen Berechnungszeitpunkt enthalten, um in SolidWorks® Bewegungsstudien durchzuführen. Damit soll eine pauschale Bewertung der Ergebnisse vereinfacht werden.

Zwänge müssen nicht festgelegt werden, da Z=0 bereits festgelegt wurde. Die Abarbeitung der sechs Schritte aus 5.3.4 ist nun bis auf das Aufstellen der Lagrange Funktion vollzogen.

6.1.2 Aufstellen der Lagrange Funktion

Die Lagrange Funktion ist die Differenz aus kinetischer und potenzieller Energie (Gleichung [2]). Der Quelltest zum Aufstellen der Lagrange Funktion in MATLAB® befindet sich im Anhang, Teil 4 (Lagrange.m).

Beim Rennwagen tritt potentielle Energie nur in den vier Federn des Fahrwerks auf. Diese lässt sich sofort aufstellen, da die Länge jeder Feder gleichzeitig eine generalisierte Koordinate des Systems ist.

$$E_{pot} = \sum_{i=1}^{4} \frac{c}{2} * |q_i - a|^2 \qquad [45]$$

In Gleichung [45] steht a für die Länge der Feder in Ausgangslage, d. h. wenn das Auto mit Fahrer still steht und 0° Lenkeinschlagwinkel hat.

Das Aufstellen der kinetischen Energien ist nun etwas aufwendiger. Jeder der fünf Körper weist drei translatorische und eine rotatorische kinetische Energie auf. Die drei Translationen des Rumpfes sind hier die entsprechenden generalisierten Koordinaten 5 bis 7 des Systems.

$$E_{kin,Rumpf,trans} = \sum_{i=5}^{7} \frac{M}{2} \dot{q}_i \qquad [46]$$

Die Rotationsenergie des Rumpfes wird nun so formuliert:

$$E_{kin,Rumpf,rot} = \underline{\omega}_{Rumpf}^{T} * \underline{\underline{J}}_{Rumpf} * \underline{\omega}_{Rumpf} \qquad [47]$$

Der Trägheitstensor J ist im lokalen Koordinatensystem des Rumpfes bereits bekannt. Jedoch muss nun eine Transformation erfolgen, um die Energie in einem ausgewählten Koordinatensystem zu berechnen. Hier wird dazu die Winkelgeschwindigkeit $\underline{\omega}$, welche in globalen Koordinaten vorliegt in das lokale Koordinatensystem des Rumpfes transfor-

miert, indem eine Transformationsmatrix erstellt wird, mit der der Vektor $\underline{\omega}$ multipliziert wird. Das Bilden der Transformationsmatrix (siehe Gleichung [48]) geschieht durch die Skalarprodukte der Einheitsvektoren des bisherigen Koordinatensystems mit den Einheitsvektoren im neuen System (formuliert im bisherigen System).

$$\underline{\omega'} = \begin{bmatrix} ex * e'x & ey * e'x & ez * e'x \\ ex * e'y & ey * e'y & ez * e'y \\ ex * e'z & ey * e'z & ez * e'z \end{bmatrix} * \underline{\omega} \qquad [48]$$

Damit ist die Energiebilanz für den Rumpf vollzogen. Für die vier Räder müssen nun neue Überlegungen angestellt werden, da die Winkelgeschwindigkeit $\underline{\omega}$ und die drei Translationen des Rades nicht aus den gewählten generalisierten Koordinaten hervorgehen. Benötigt werden konkret der Geschwindigkeitsvektor des Radschwerpunkts mit seinen drei Komponenten im lokalen Koordinatensystem und der Vektor $\underline{\omega}$ um die Rotationsenergie zu bestimmen. Zur Veranschaulichung des Problems dient die Abbildung 19. Hier kommt nun die Kinematik aus Kapitel 3 zum Einsatz. Da die generalisierten Koordinaten gleichzeitig der Input der Kinematik Hauptfunktion sind, kann zu jedem Zeitpunkt die Position aller Punkte bestimmt werden. Die Stellung eines Rades hängt von den Positionen der drei Punkte C, F und H ab. Durch die Lösung der Kinematik sind die drei Ortsvektoren r_{xy} jedes Rades bekannt. Die entsprechenden Geschwindigkeitsvektoren entstehen durch die partielle Ableitung der Ortsvektoren nach der Zeit:

$$\underline{v_{xy}} = \frac{d\underline{r_{xy}}}{dt} = \frac{\partial \underline{r_{xy}}}{\partial q_i} * \dot{q}_i \qquad [49]$$

Nach dem gleichen Schema werden die Geschwindigkeitsvektoren der vier Radschwerpunkte (Punkt N in der Kinematik, Kapitel 3) gebildet.

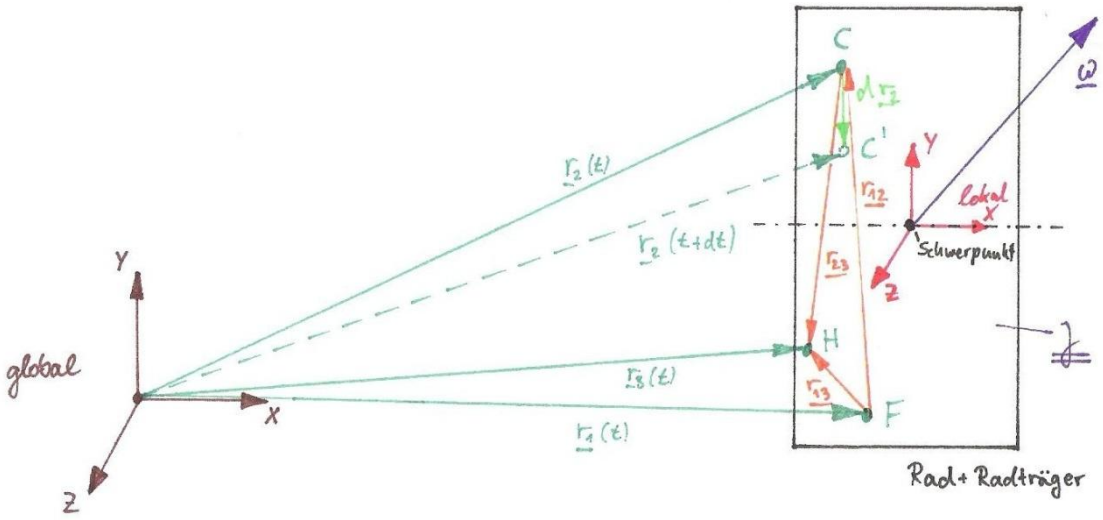


Abbildung 19: Berechnung von Vektor $\underline{\omega}$ am Rad

Damit kann nun die kinetische Energie der Translation der vier Räder aufgestellt werden:

$$E_{kin,Räder,trans} = \sum_{i=1}^{4} \frac{m}{2} v_{SPx_i} + \frac{m}{2} v_{SPy_i} + \frac{m}{2} v_{SPz_i} \quad [50]$$

Zur Berechnung der noch ausstehenden Rotationsenergie werden zur Bestimmung von $\underline{\omega}$ Differenzvektoren der Geschwindigkeitsvektoren $\underline{v}_{xy}$ und der Ortsvektoren $\underline{r}_{xy}$ gebildet (in Abbildung 19 orange). Nun wird jeder Differenzvektor $\underline{r}_{dxy}$ mit dem zugehörigen Differenzvektor $\underline{v}_{dxy}$ vektoriell multipliziert. Die Vektoren, die so den größten Betrag erzeugt haben, werden zur weiteren Berechnung von $\underline{\omega}$ verwendet. Diese Auswahl soll Unstetigkeiten während der Berechnung verhindern, was auftreten kann, wenn der Betrag des Vektorproduktes nahe 0 liegt. Nun wird die Winkelgeschwindigkeit $\underline{\omega}$ mit den ausgewählten $\underline{v}_{dxy}$ und $\underline{r}_{dxy}$ berechnet:

$$\underline{\omega}_{Rad} = \frac{\underline{r}_{dxy} \times \underline{v}_{dxy}}{|\underline{r}_{dxy}|^2} \quad [51]$$

Die gewonnene Winkelgeschwindigkeit, welche im globalen Koordinatensystem vorliegt, muss nun auch noch in das lokale System des Rades transformiert werden. Dazu werden lokale Einheitsvektoren erzeugt, welche im globalen System formuliert werden. Mit diesen erfolgt die Transformation wie beim Rumpf nach Gleichung [48]. Die Rotationsenergie der vier Räder lautet damit nun:

$$E_{kin,Räder,rot} = \sum_{i=1}^{4} \underline{\omega}_{Rad_i}^{T} * \underline{\underline{J}}_{Rad} * \underline{\omega}_{Rad_i} \quad [52]$$

Alle Energien sind nun bestimmt, womit die Lagrange Funktion nun aufgestellt werden kann:

$$L = E_{kin,Räder,rot} + E_{kin,Räder,trans} + E_{kin,Rumpf,rot} + E_{kin,Rumpf,trans} - E_{pot} \quad [53]$$

Jetzt sind alle Überlegungen zur Anwendung des Komplexbeispiels am Grundprogramm getroffen. Am Grundprogramm selbst müssen keine Änderungen vorgenommen werden. Es ist nur notwendig, das Lagrange.m File entsprechend zu erweitern. Diese Tatsache unterstreicht nochmals den universellen Charakter des Grundprogramms.

6.1.3 Programmergänzung

Die programmergänzenden Maßnahmen, welche durch die Überlegungen aus Kapitel 6.1.2 notwendig zur Implementierung des Komplexprogramms sind, werden nun erläutert. Im Prinzip wird der Programmaufbau, welcher aus Abbildung 11 bekannt ist, um die Abfolge aus Abbildung 20 ergänzt. D. h. jedes Mal wenn das Lagrange.m File aufgerufen wird, wird der Ablauf aus Abbildung 20 ausgeführt. Die Quelltexte hierfür sind ebenfalls unter der Anlage, Teil 4 hinterlegt.

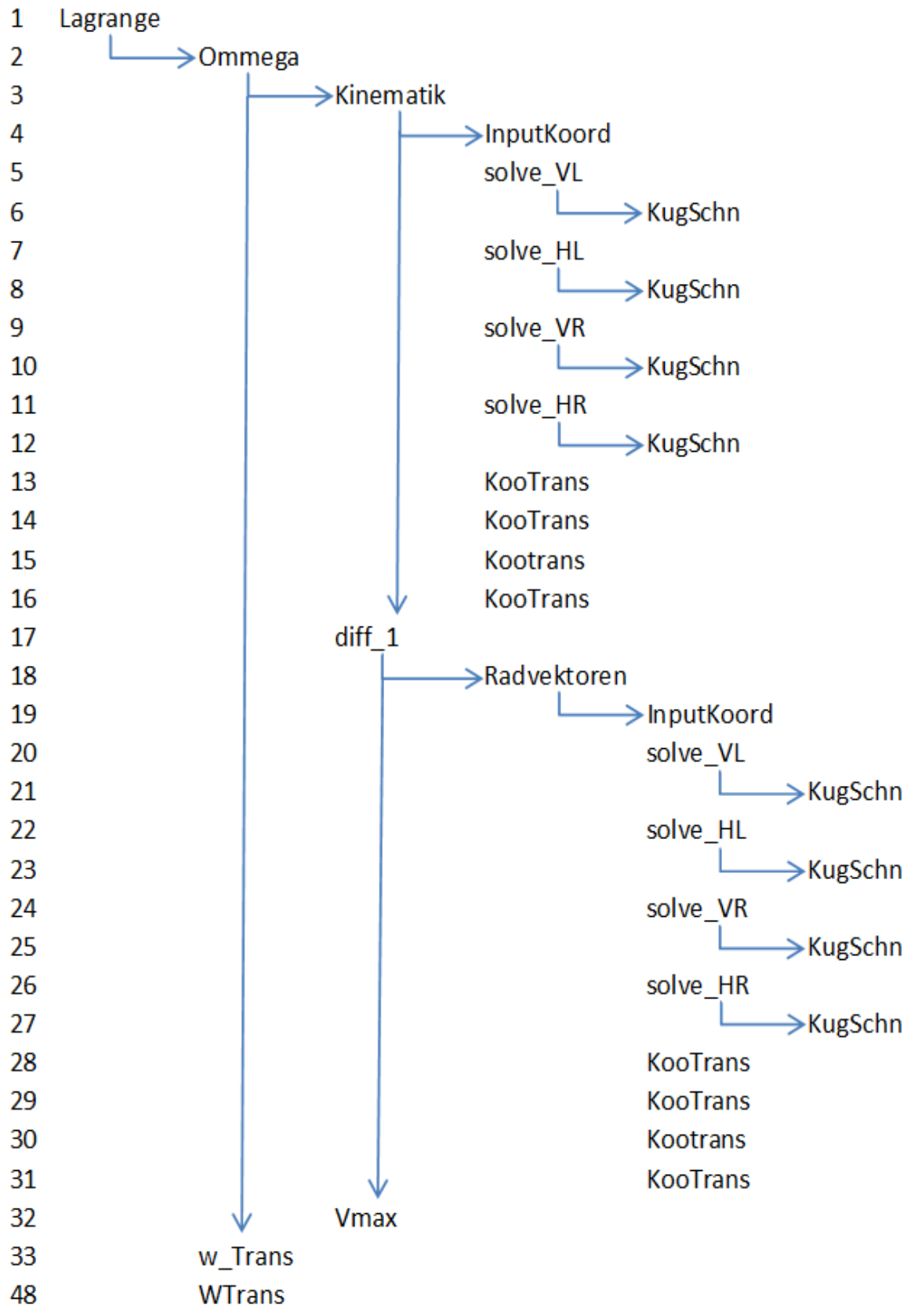


Abbildung 20: Programmergänzung

Die Programmergänzung beinhaltet neue und bereits bekannte Unterfunktionen. Hier wird sich nur auf die Beschreibung der neuen Funktionen beschränkt.

In Lagrange.m werden zuerst die Konstanten definiert. Das sind die Masse der Räder, die Masse des Rumpfes, die Federsteifigkeit, die Trägheitstensoren der Räder und des Rumpfes, sowie die Länge der Federn in Ausgangslage.

Anschließend wird die Omega.m Funktion aufgerufen. Hier geht der Lenkeinschlagwinkel, welcher in der Hauptfunktion festgelegt wurde, mit als Freiheitsgrad der Kinematik ein. Die Lösung der kompletten Kinematik folgt in der nächsten aufgerufenen Funktion, der Kinematik.m Funktion. Hierbei handelt es sich um eine Kopie der Funktion FWkomplett.m. Es werden lediglich die Ergebnisse sinnvoll aufbereitet. Die berechneten Vektoren werden global gespeichert, um bei späterem Gebrauch die Rechnung nicht wiederholen zu müssen.

Die Ableitung zur Berechnung der $\underline{v}_{xy}$ wird jetzt nach Gleichung [49] in der Funktion diff_1.m ausgeführt. Dabei findet die Funktion RadVek.m Anwedung. Diese Funktion ist wiederum eine Kopie FWkomplett.m mit dem Unterschied, dass sie nur die Ortsvektoren der Punkte C, F und H zurückgibt. Jetzt werden die Differenzvektoren der Ortsvektoren und der Geschwindigkeitsvektoren gebildet, um in der Funktion VMax.m zur Auswahl der geeigneten Vektorenpaare zur Nutzung zu kommen. Bevor die letztendlichen Winkelgeschwindigkeiten nach Gleichung [51] ausgerechnet werden, geht eine Aufbereitung der Vektoren voraus.

Die gewonnen $\underline{\omega}$ Vektoren werden jetzt in der Lagrange.m Funktion mit dem Aufruf der Funktion w_Trans.m transformiert, um in die Lagrange Funktion einzugehen. Analog erfolgt diese Transformation in der Funktion WTrans.m für die Winkelgeschwindigkeit des Rumpfes, bei der die generalisierten Koordinaten 8 bis 10 eingehen.

Nun sind alle Winkelgeschwindigkeiten und translatorischen Geschwindigkeiten bestimmt, womit die Lagrange Funktion voll definiert ist. Die Berechnung am Komplexbeispiel kann nun gestartet werden.

6.2 Ergebnisse

Die entstandenen Ergebnisse werden nun aufgezeigt und interpretiert.

6.2.1 Rechendauer

Es zeigte sich, dass die Rechendauer für einen vom ode45 festgelegten Zeitschritt unvertretbar lang war. Die Rechendauer für das komplette Zeitintervall hätte damit mehrere Wochen[17] in Anspruch genommen. Deswegen wurde der Programmablauf analysiert und optimiert. Rechenschritte, die mehrmals für das gleiche Ergebnis ausgeführt wurden, werden nun nur noch einmal ausgeführt. Deren Ergebnisse wurden globalisiert, um an anderer Stelle darauf zugreifen zu können.

Da in den Differentiationsschritten immer wieder auf die aufwendige Kinematik zurückgegriffen wird, macht es Sinn, in weiterführenden Untersuchungen das Differentiationsverfahren zu überdenken.

Die Unterfunktion KugSchn.m wird in einem Rechenschritt ca. 65.000-mal aufgerufen. Nach dieser Erkenntnis wurde diese Funktion beschleunigt, indem z. B. die Aufrufe der Build-In Funktion “cross“ durch einen selbst programmierten Vektorprodukt-Algorithmus ersetzt wurde. Jedoch brachte dieser Schritt keine annehmbare Verkürzung der Rechendauer.

Eine Kompilierung in die Programmiersprache C® zum Erhalt einer exe-Datei hat, wie die vorangegangenen Unternehmungen auch, keinen erheblichen Fortschritt erbracht. Deshalb wurde die relative Fehlertoleranz des ode45-Solver auf 0,1 heraufgesetzt. Dadurch wurden die Zeitschritte während der Berechnung merklich vergrößert. Damit wurde eine Berechnungsdauer für das Komplexbeispiel von zumutbaren 25 Minuten[17] erreicht. Jedoch ist nicht abschätzbar, wie groß der Einfluss auf die Genauigkeit der Ergebnisse ist.

6.2.2 Diagramme

Die folgenden Diagramme zeigen die generalisierten Koordinaten über der Zeit für den Rennwagen mit den unter 6.1.1 gewählten Anfangswerten.

[17] Rechner mit folgender Leistung: Intel® Core™ i5 CPU 760 @ 2.80GHz, 8,00 GB RAM

Betriebssystem: Windows 7 Enterprise 64-Bit

MATLAB® Version: 7.10.0.499 (R2010a)

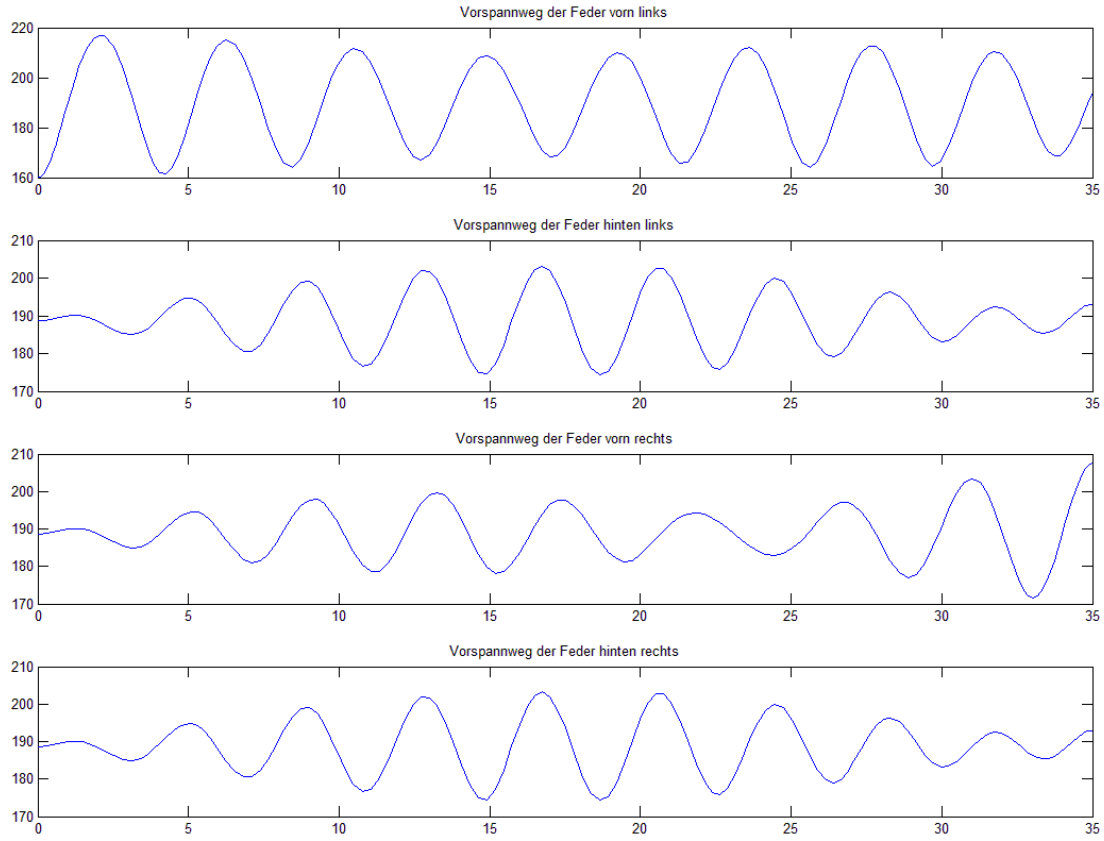


Abbildung 21: Federlängen über der Zeit

In Abbildung 21 werden die vier Federlängen über der Zeit gezeigt. Die Startwerte bei t=0 entsprechen den gewählten Anfangswerten. Das Rad vorn links beginnt zu schwingen und überträgt die Schwingung auf den Rumpf, was die anderen Räder erregt. Dabei sinkt die Amplitude des Rades vorn links leicht. Bei den anderen drei Rädern ist ebenfalls ein Auf- und Abschwingverhalten erkennbar. Die Schwingungen sind also zyklisch. Das Verhalten der Federlängen am Komplexbeispiel erscheint damit plausibel.

Beim realen Rennwagen würden die Schwingungen durch die Dämpferkräfte gedämpft werden. Diese Kräfte würden dann als Q_i (nichtkonservative Kräfte der i-ten Koordinate) mit in die Lagrange Gleichung [3] eingehen.

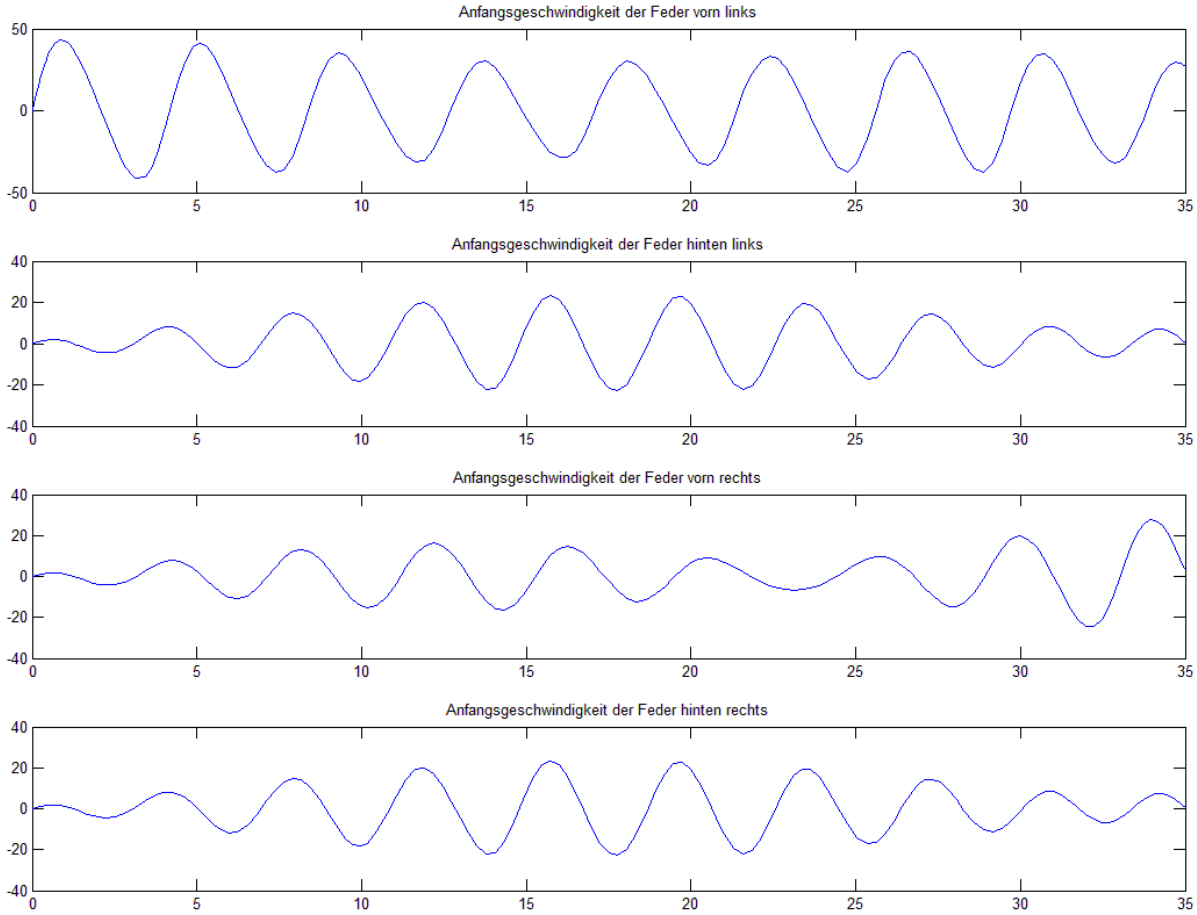


Abbildung 22: Federgeschwindigkeiten

Auch die Federgeschwindigkeiten erscheinen plausibel. Alle Geschwindigkeiten beginnen bei 0 mm/s und weisen ähnliche Zyklen wie die Schwingungen in Abbildung 21 auf.

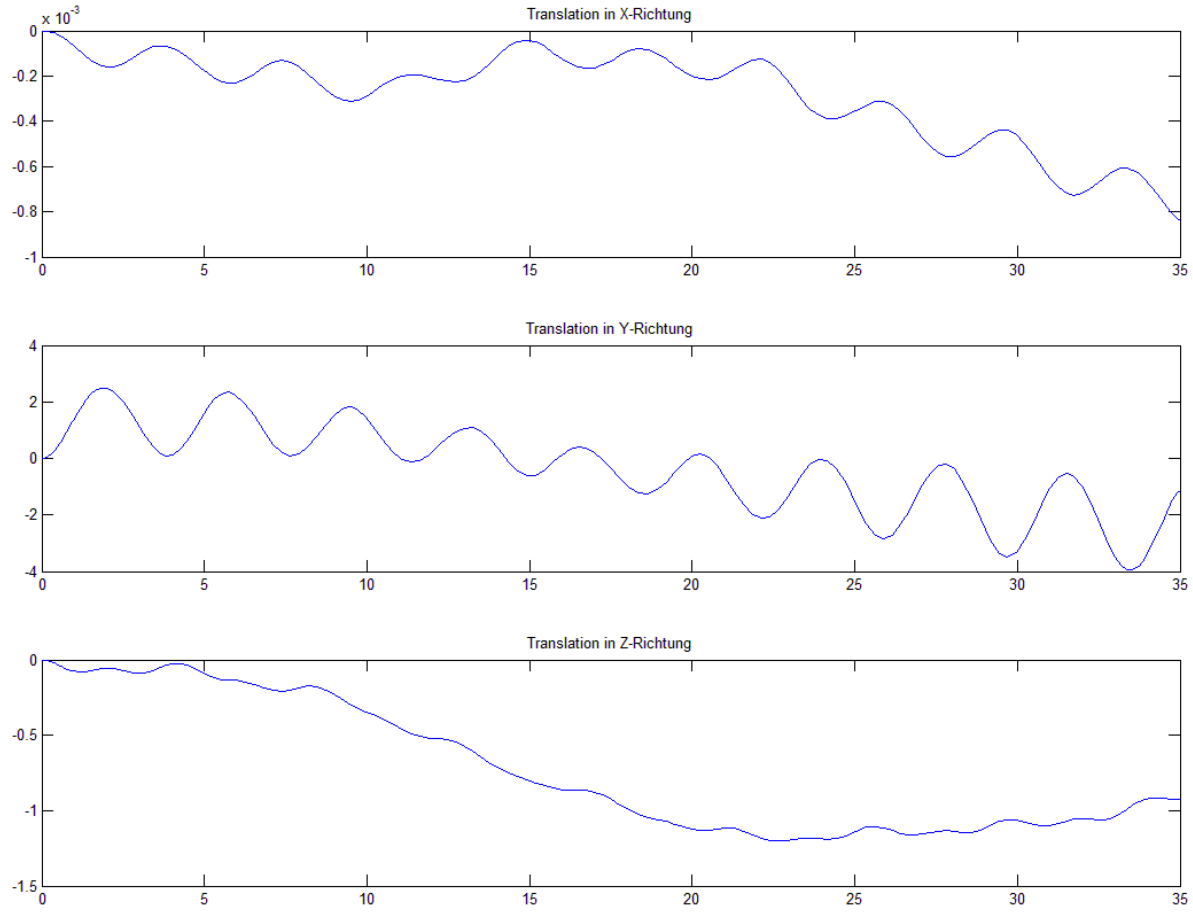


Abbildung 23: Translation des Rumpfes

Der Rumpf bewegt sich während des Intervalls nur minimal, wie in den Abbildung 23 bis 26 zu sehen ist. Ein Grund dafür kann ein unrealistisch gewählter Trägheitstensor sein. Es ist jedoch erkennbar, dass die erzeugte Schwingung durch den Rumpf an die restlichen Räder übertragen wird.

In Abbildung 25 wird deutlich, dass der größte Rotationswinkel um die x-Achse erreicht wird. Um diese Achse ist das Trägheitsmoment des Rumpfes schließlich auch am geringsten.

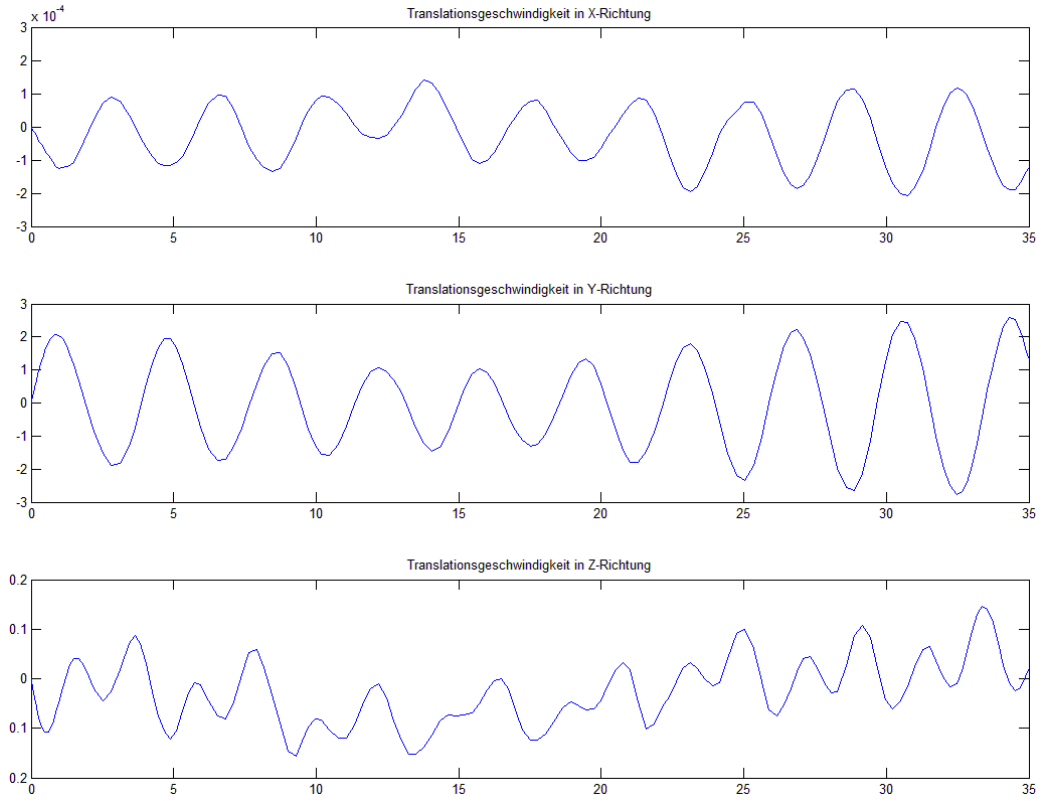


Abbildung 24: Translationsgeschwindigkeit des Rumpfes

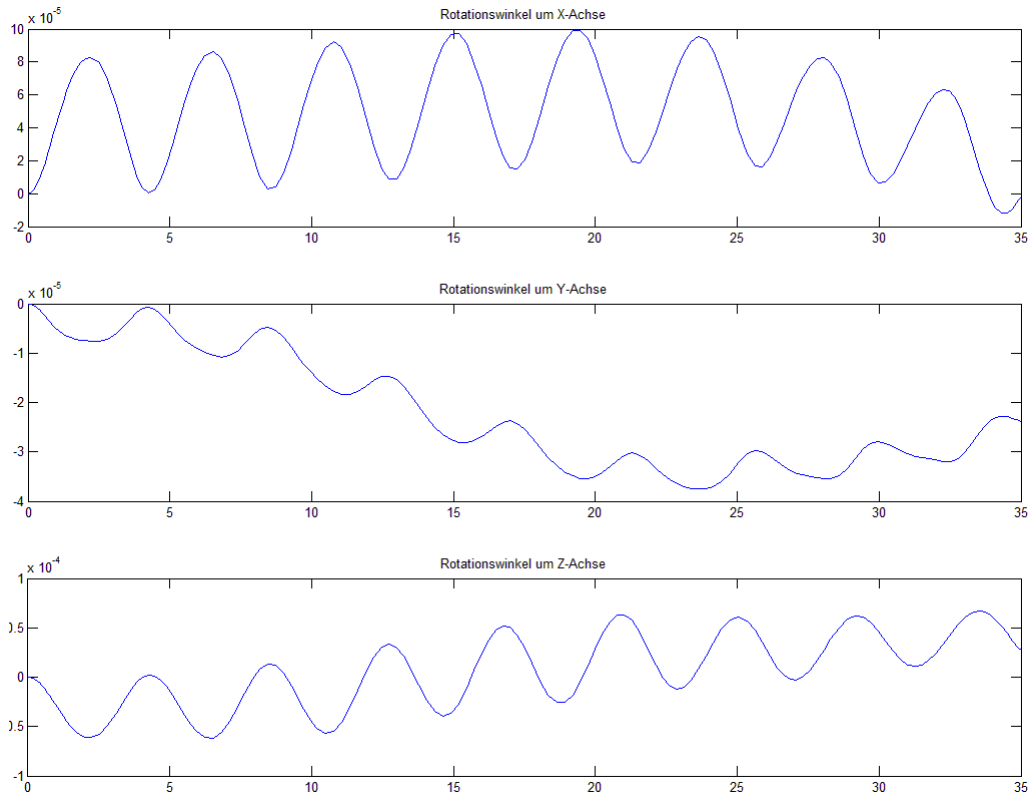


Abbildung 25: Rotation des Rumpfes

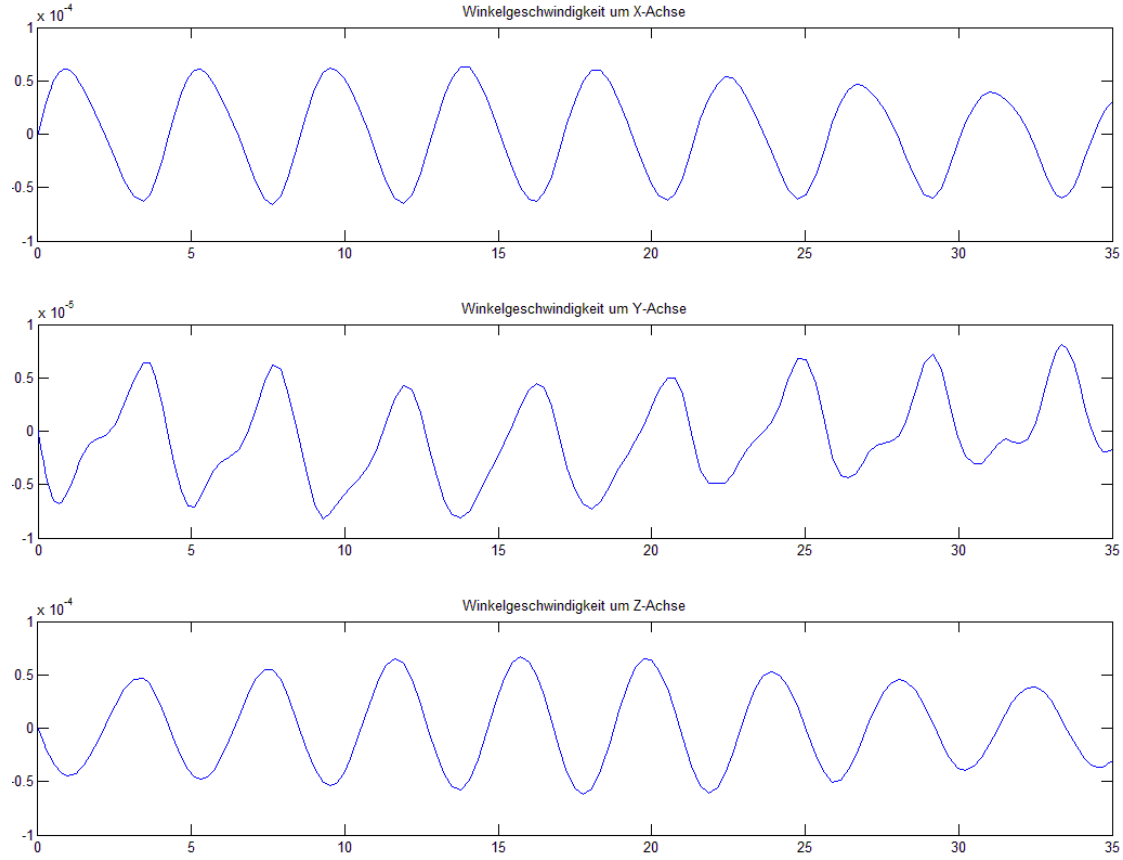


Abbildung 26: Rotationsgeschwindigkeit des Rumpfes

6.2.3 Animation in SolidWorks®

Um die Richtigkeit der Ergebnisse der Berechnung besser abschätzen zu können, wurde eine Bewegungsstudie in SolidWorks® durchgeführt. Dabei kamen die im ODEsolver.m Skript erzeugten Datensätze zum Einsatz. Der Aufbau der Animation erfolgte wie in 5.3.4 beschrieben.

In Abbildung 27 ist das schemenhafte Modell des Rennwagens, welches animiert wurde, zu sehen. Dazu gehören der Rumpf, die vier Räder und die Fahrbahn. Um die Verschiebung der Räder in alle drei Koordinatenachsen animieren zu können, mussten nochmals mit der Kinematik Funktion FWkomplett.m die Positionen der Punkte für jeden Zeitschritt berechnet werden.

Die Animation festigte den Eindruck, dass die Ergebnisse plausibel erscheinen. Die Radbewegungen erfolgten wie erwartet.

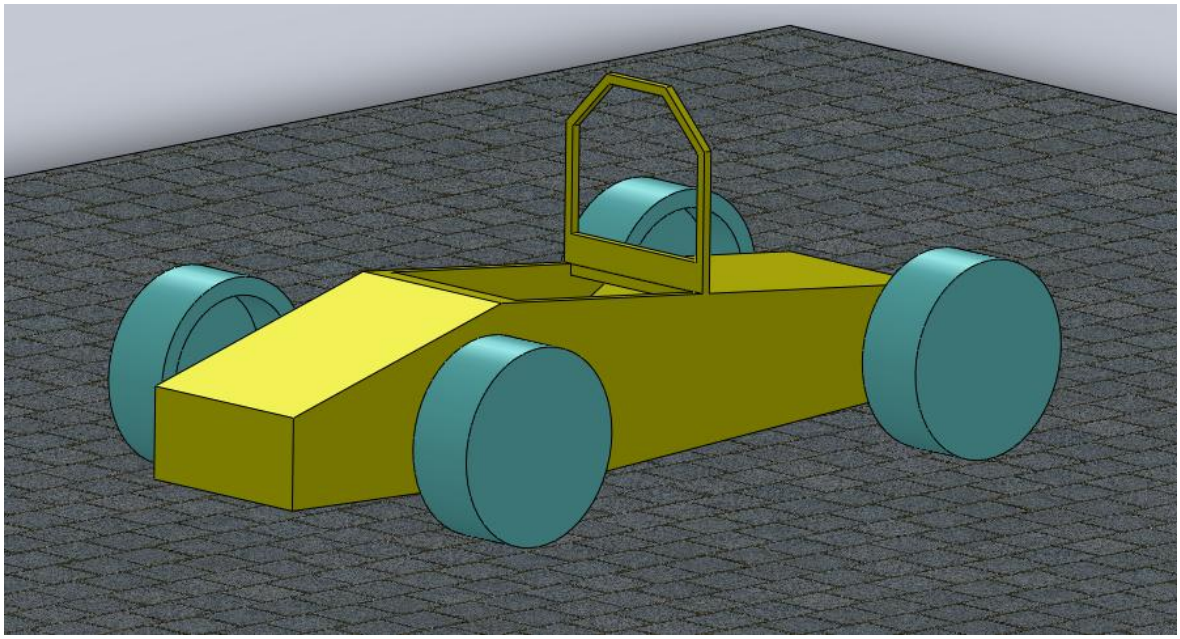


Abbildung 27: Animation des Rennwagens

7 Ergebnisse und Ausblick

Im letzten Kapitel werden nun zusammenfassend Kernaussagen zu den gewonnen Ergebnissen getroffen und ein Ausblick zur weiteren Nutzung der Erkenntnisse geliefert.

7.1 Ergebnisse

Die Ergebnisse der behandelten Themen werden hier nochmals zusammengefasst.

7.1.1 Kinematik

Mit der vollen Abbildung der Kinematik des Rennwagens (Kapitel 3) ist es nun möglich, abhängig der Freiheitsgrade die Positionen der Aufhängungspunkte sicher zu bestimmen. Diese Ergebnisse sind durch den entwickelten Algorithmus exakt und mit Hilfe eines CAD Modells auf ihre Richtigkeit hin geprüft.

Auf die Kinematik konnte bei der dynamischen Berechnung erfolgreich zurückgegriffen werden. Die generalisierten Koordinaten wurden wie die Freiheitsgrade der Kinematik Funktion gewählt, wodurch nur geringfügige Programmergänzungen notwendig wurden.

7.1.2 Lagrange 1. Art im Grundprogramm

Die rechentechnische Anwendung der in Kapitel 4 erarbeiteten Theorie zur Lagrange Gleichung 1. Art wurde erfolgreich im Grundprogramm in Kapitel 5 gezeigt.

Damit ist ein Programm zur Analyse beliebiger dynamisch belasteter Systeme entstanden. Dieses mächtige Werkzeug lässt sich ohne großen Aufwand an jedes System mit beliebiger Anzahl von Freiheitsgraden und Zwängen anpassen. Die Möglichkeiten zur Auswertung sind dabei ausreichend. In mehreren Tests wurde die Richtigkeit der Ergebnisse gezeigt.

7.1.3 Anwendung Komplexbeispiel

Grundsätzlich ist das Analyseverfahren nach der Theorie aus Kapitel 4 zur Untersuchung eines Rennwagens mit Doppelquerlenkeraufhängung als Beispiel eines komplexen dynamisch belasteten Systems geeignet, da das Programm lauffähig ist und plausible Ergebnisse liefert.

Jedoch können bisher nur pauschale Aussagen über die Richtigkeit der Ergebnisse getroffen werden, da sie noch nicht validiert sind. Auch der Einfluss durch das Heraufsetzen der relativen Fehlertoleranz ist noch nicht abschätzbar.

7.2 Ausblick

Die Möglichkeiten zur weiteren Nutzung der gewonnen Ergebnisse werden nun aufgezeigt.

7.2.1 Weitere Nutzung der Kinematik

Das erzeugte Programm zur Abbildung der Kinematik in MATLAB® kann so erweitert werden, dass man ein nützliches Werkzeug bei der Auslegung der Fahrwerkskinematik für folgende Rennwagenkonstruktionen erhält. Ein kleiner Ausblick wurde dahingehend schon in meinem Praktikumsbericht gegeben.

7.2.2 Weitere Nutzung des Grundprogramms

Die Leistungsfähigkeit des Grundprogramms sollte noch durch das Inbetrachtziehen von nichtkonservativen Kräften, wie z. B. Dämpferkräften, gesteigert werden.

Eine noch größere Anwenderfreundlichkeit wäre wünschenswert. Bei der Betrachtung von Systemen, die Zwängen unterliegen, sollte die Aufbereitung von sinnvollen Anfangswerten automatisiert werden.

7.2.3 Anwendung am Rennwagen

Hier sollte zunächst eine Validierung der Ergebnisse erfolgen. Danach sollte die numerische Differentiationsmethode auf eine zeitliche Optimierungskapazität hin untersucht werden, um die Rechendauer auf ein zumutbares Niveau zu reduzieren.

Danach sollte das Berechnungsmodell mit Dämpferkräften und sinnvollen Zwängen bestückt werden. Die Zwänge sollen dabei extreme Fahrsituationen simulieren. Mit den Ergebnissen können dann quasistatische Analysen der d´Alambertschen Trägheitskräfte angestellt werden, um die dynamischen Kräfte in den belasteten Aufhängungskomponenten zu berechnen. Diese Kräfte können dann mit Messungen am realen Fahrzeug verglichen werden, um das gesamte Berechnungsmodell zu validieren.

Anlagen

Anlagen, Teil 1

Quelltexte der Kinematik:

FWkomplett.m

```
% Hauptprogramm zur vollständigen Bestimmtheit des Fahrzeugs im Raum
% Input: Länge der Federn: l_VL, l_HL, l_VR, l_HR [mm] (Max 200 /Min 130)
%       Winkel Lenkeinschlag: winkel [°] (Max 120 /Min -120)
%       Koordinaten des Schwerpunktes SP: SPx,SPy,SPz (SPz sollte 0 sein)
%       Translation des Rahmens in X, Y, Z
%       Rotationswinkel alpha1 (um x-Achse), alpha2 (um y), alpha3 (um z)
% Output: Position aller Fahrwerkspunkte abhängig von den Eingabewerten
% integrierte Programme: "solve_VL.m", "solve_HL.m", "solve_VR.m",
%                        "solve_HR.m", "Kootrans.m", "InputKoord.m"
function[VL,HL,VR,HR]=FWkomplett(laenge_VL,laenge_HL,laenge_VR,...
   laenge_HR,SPx,SPy,SPz,X,Y,Z,alpha1,alpha2,alpha3)
% Bildung des Schwerpunktvektors:
SP=[SPx;SPy;SPz];
% Bezug der Fahrwerkspunkte in Ausgangslage:
InputKoord;
% Festlegen des Lenkeinschlagwinkels
winkel=0;
%   Übergabe der Eingabewerte in Unterfunktionen
%   Punkte werden so verschoben, das SP auf Urschprung liegt
%   Alle Punkte werden berechnet:
vl_1=solve_VL(laenge_VL,winkel);
hl_1=solve_HL(laenge_HL);
vr_1=solve_VR(laenge_VR,winkel);
hr_1=solve_HR(laenge_HR);
%   Koordinatentransformation-Rotation um alle 3 Achsen:
vl_2=KooTrans(vl_1,alpha1,alpha2,alpha3);
hl_2=KooTrans(hl_1,alpha1,alpha2,alpha3);
vr_2=KooTrans(vr_1,alpha1,alpha2,alpha3);
hr_2=KooTrans(hr_1,alpha1,alpha2,alpha3);
%   Zurückverschiebung der Punkte:
Z=SP+[X;Y;Z];
%   Ausgabe aller Punkte A-N aller 4 Aufhängungen
vl_3=[(vl_2(:,1))+Z,(vl_2(:,2))+Z,(vl_2(:,3))+Z,(vl_2(:,4))+Z,...
   (vl_2(:,5))+Z,(vl_2(:,6))+Z,(vl_2(:,7))+Z,(vl_2(:,8))+Z,...
   (vl_2(:,9))+Z,(vl_2(:,10))+Z,(vl_2(:,11))+Z,(vl_2(:,12))+Z,...
   (vl_2(:,13))+Z,(vl_2(:,14))+Z,];

hl_3=[(hl_2(:,1))+Z,(hl_2(:,2))+Z,(hl_2(:,3))+Z,(hl_2(:,4))+Z,...
   (hl_2(:,5))+Z,(hl_2(:,6))+Z,(hl_2(:,7))+Z,(hl_2(:,8))+Z,...
   (hl_2(:,9))+Z,(hl_2(:,10))+Z,(hl_2(:,11))+Z,(hl_2(:,12))+Z,...
   (hl_2(:,13))+Z,(hl_2(:,14))+Z,];

vr_3=[(vr_2(:,1))+Z,(vr_2(:,2))+Z,(vr_2(:,3))+Z,(vr_2(:,4))+Z,...
   (vr_2(:,5))+Z,(vr_2(:,6))+Z,(vr_2(:,7))+Z,(vr_2(:,8))+Z,...
   (vr_2(:,9))+Z,(vr_2(:,10))+Z,(vr_2(:,11))+Z,(vr_2(:,12))+Z,...
   (vr_2(:,13))+Z,(vr_2(:,14))+Z,];

hr_3=[(hr_2(:,1))+Z,(hr_2(:,2))+Z,(hr_2(:,3))+Z,(hr_2(:,4))+Z,...
```

```
    (hr_2(:,5))+Z,(hr_2(:,6))+Z,(hr_2(:,7))+Z,(hr_2(:,8))+Z,...
    (hr_2(:,9))+Z,(hr_2(:,10))+Z,(hr_2(:,11))+Z,(hr_2(:,12))+Z,...
    (hr_2(:,13))+Z,(hr_2(:,14))+Z,];
% Ausgabe als Matrix:
VL=vl_3
HL=hl_3
VR=vr_3
HR=hr_3
```

InputKoord.m

```
% Skript, welches alle Punkte der Aufhängungen in Ausgangslage enthält.
% Dieses Skript wird in der Hauptfunktion "FWkomplett.m" aufgerufen
global Afixvl Bfixvl Cfixvl Dfixvl Efixvl Ffixvl Gfixvl Hfixvl Ifixvl ...
    Jfixvl Kfixvl Lfixvl Mfixvl Nfixvl Afixhl Bfixhl Cfixhl Dfixhl ...
    Efixhl Ffixhl Gfixhl Hfixhl Ifixhl Jfixhl Kfixhl Lfixhl Mfixhl ...
    Nfixhl Afixvr Bfixvr Cfixvr Dfixvr Efixvr Ffixvr Gfixvr Hfixvr ...
    Ifixvr Jfixvr Kfixvr Lfixvr Mfixvr Nfixvr Afixhr Bfixhr Cfixhr ...
    Dfixhr Efixhr Ffixhr Gfixhr Hfixhr Ifixhr Jfixhr Kfixhr Lfixhr ...
    Mfixhr Nfixhr;

% vorn links:
Afixvl=[161.77;368.71;250]-SP;
Bfixvl=[-72.36;368.71;250]-SP;
Cfixvl=[23.38;392;545.88]-SP;
Dfixvl=[-72.36;148.91;250]-SP;
Efixvl=[161.77;148.91;250]-SP;
Ffixvl=[0;140;590.31]-SP;
Gfixvl=[0;0;585]-SP;
Hfixvl=[-66.28;180.13;583.24]-SP;
Ifixvl=[23.38;380.6;240.55]-SP;
Jfixvl=[23.38;216.15;333]-SP;
Kfixvl=[23.38;162.85;240.7]-SP;
Lfixvl=[23.38;153.1;317.4]-SP;
Mfixvl=[-48.71;182.94;250]-SP;
Nfixvl=[0;267.73;573.3]-SP;

%   hinten links:
Afixhl=[1786.59;352.84;310]-SP;
Bfixhl=[1502.26;352.84;310]-SP;
Cfixhl=[1600;373.75;549.10]-SP;
Dfixhl=[1501.67;165;310]-SP;
Efixhl=[1790.46;165;310]-SP;
Ffixhl=[1609.68;155.31;587.62]-SP;
Gfixhl=[1650;0;585]-SP;
Hfixhl=[1730;304.82;561.25]-SP;
Ifixhl=[1600;376;294.52]-SP;
Jfixhl=[1600;206.72;378.24]-SP;
Kfixhl=[1600;125;279.53]-SP;
Lfixhl=[1600;139.85;385.7]-SP;
Mfixhl=[1801.47;292.83;309.98]-SP;
Nfixhl=[1650;267.73;573.3]-SP;

%   vorn rechts (vorn links wird gespiegelt):
Afixvr=Afixvl.*[1;1;-1];
Bfixvr=Bfixvl.*[1;1;-1];
Cfixvr=Cfixvl.*[1;1;-1];
Dfixvr=Dfixvl.*[1;1;-1];
Efixvr=Efixvl.*[1;1;-1];
Ffixvr=Ffixvl.*[1;1;-1];
```

```
Gfixvr=Gfixvl.*[1;1;-1];
Hfixvr=Hfixvl.*[1;1;-1];
Ifixvr=Ifixvl.*[1;1;-1];
Jfixvr=Jfixvl.*[1;1;-1];
Kfixvr=Kfixvl.*[1;1;-1];
Lfixvr=Lfixvl.*[1;1;-1];
Mfixvr=Mfixvl.*[1;1;-1];
Nfixvr=Nfixvl.*[1;1;-1];

%   hinten rechts (hinten links wird gespiegelt):
Afixhr=Afixhl.*[1;1;-1];
Bfixhr=Bfixhl.*[1;1;-1];
Cfixhr=Cfixhl.*[1;1;-1];
Dfixhr=Dfixhl.*[1;1;-1];
Efixhr=Efixhl.*[1;1;-1];
Ffixhr=Ffixhl.*[1;1;-1];
Gfixhr=Gfixhl.*[1;1;-1];
Hfixhr=Hfixhl.*[1;1;-1];
Ifixhr=Ifixhl.*[1;1;-1];
Jfixhr=Jfixhl.*[1;1;-1];
Kfixhr=Kfixhl.*[1;1;-1];
Lfixhr=Lfixhl.*[1;1;-1];
Mfixhr=Mfixhl.*[1;1;-1];
Nfixhr=Nfixhl.*[1;1;-1];
```

solve_VL.m

```
% Berechnung der Punkte der Aufhängung vorn links
% Unterprogramm von "FWkomplett.m"
% Input: Länge der Feder-/Dämpfereinheit,
%        Lenkeinschlagwinkel in °
%        Bezug der Punkte aus "InputKoord.m"
% Output: Punkte der Aufhängung
% integrierte Programme: "KugSchn.m"
function[VL]=solve_VL(laenge,winkel)
% Umrechnung des Lenkeinschlagwinkels in die Verschiebung der Zahnstange im
% Lenkgetriebe:
spur=winkel/6;
global Afixvl Bfixvl Cfixvl Dfixvl Efixvl Ffixvl Gfixvl Hfixvl Ifixvl ...
    Jfixvl Kfixvl Lfixvl Mfixvl Nfixvl;
% Berechnung des Punktes J:
x1=Ifixvl(3);
x2=Kfixvl(3);
y1=Ifixvl(2);
y2=Kfixvl(2);
R1=laenge;
R2=norm(Jfixvl-Kfixvl);
m_=(x1-x2)/(y2-y1);                          % Schnittgerade
n_=(-R2^2+R1^2+y2^2-y1^2+x2^2-x1^2)/(2*(y2-y1));    %   -"-
p_=(2*(-x1+m_*(n_-y1)))/(1+m_^2);                % Quad-Gleichung
q_=(x1^2+(n_-y1)^2-R1^2)/(1+m_^2);               %   -"-
z1=-(p_/2)+sqrt((p_/2)^2-q_);                    % Lösung der Gleichung
z2=-(p_/2)-sqrt((p_/2)^2-q_);                    %   -"-
y1=m_*z1+n_;                                 % Einsetzen in Gerade
y2=m_*z2+n_;
J=[Jfixvl(1);y1;z1];  % Lösung 1
%J=[Jfixvl(1);y2;z2];  % Lösung 2

% Berechnung des Punktes L:
```

```
x1=J(3);
x2=Kfixvl(3);
y1=J(2);
y2=Kfixvl(2);
R1=norm(Lfixvl-Jfixvl);
R2=norm(Lfixvl-Kfixvl);
m_=(x1-x2)/(y2-y1);                          % Schnittgerade
n_=(-R2^2+R1^2+y2^2-y1^2+x2^2-x1^2)/(2*(y2-y1));    %   -"-
p_=(2*(-x1+m_*(n_-y1)))/(1+m_^2);                % Quad-Gleichung
q_=(x1^2+(n_-y1)^2-R1^2)/(1+m_^2);                %   -"-
z1=-(p_/2)+sqrt((p_/2)^2-q_);                % Lösung der Gleichung
z2=-(p_/2)-sqrt((p_/2)^2-q_);                %   -"-
y1=m_*z1+n_;                                 % Einsetzen in Gerade
y2=m_*z2+n_;                                 %   -"-
L=[Lfixvl(1);y1;z1];  % Lösung 1
%L=[Lfixvl(1);y2;z2];  % Lösung 2

% Berechnung des Punktes C:
C=KugSchn(Afixvl,Bfixvl,L,norm(Cfixvl-Afixvl),norm(Cfixvl-Bfixvl)...
   ,norm(Cfixvl-Lfixvl),1);
% Berechnung des Punktes F:
F=KugSchn(Dfixvl,Efixvl,C,norm(Ffixvl-Dfixvl),norm(Ffixvl-Efixvl)...
   ,norm(Ffixvl-Cfixvl),1);
% Berechnung des Punktes H:
Mtemp=Mfixvl+[0;0;spur];
H=KugSchn(Mtemp,C,F,norm(Hfixvl-Mfixvl),norm(Hfixvl-Cfixvl),norm...
   (Hfixvl-Ffixvl),0);
% Berechnung des Punktes G:
G=KugSchn(C,F,H,norm(Gfixvl-Cfixvl),norm(Gfixvl-Ffixvl),norm...
   (Gfixvl-Hfixvl),1);
% Berechnung des Punktes N:
N=KugSchn(C,H,F,norm(Nfixvl-Cfixvl),norm(Nfixvl-Hfixvl),norm...
   (Nfixvl-Ffixvl),1);

% Übergabe der Punkte als Matrix:
VL=[Afixvl(1) Bfixvl(1) C(1) Dfixvl(1) Efixvl(1) F(1) G(1) H(1)...
   Ifixvl(1) J(1) Kfixvl(1) L(1) Mtemp(1) N(1);Afixvl(2) Bfixvl(2)...
   C(2) Dfixvl(2) Efixvl(2) F(2) G(2) H(2) Ifixvl(2) J(2) Kfixvl(2)...
   L(2) Mtemp(2) N(2);Afixvl(3) Bfixvl(3) C(3) Dfixvl(3) Efixvl(3)...
   F(3) G(3) H(3) Ifixvl(3) J(3) Kfixvl(3) L(3) Mtemp(3) N(3)];
```

Die Funktionen solve_HL.m, solve_VR.m und solve_HR.m sind analog zu solve_VL.m, nur das sie sich der fixen Punkte der jeweiligen Aufhängung aus dem InputKoord.m script-File bedienen.

KugSchn.m

```
% Ermittelt gewünschten Schnittpunkt von drei Kugeln
% Unterprogramm von: "solve_VL.m", "solve_HL.m", "solve_VR.m", "solve_HR.m"
% Input: 3 Mittelpunkte der Kugeln, 3 jeweilige Radien, Wahl des
%        Schnittpunktes mit pos=1 oder 0: M1,M2,M3,r1,r2,r3,pos
%        Mittelpunkte als Spaltenvektor
% Output: gewählter Schnittpunkt
% Siehe Skizze "Schnittpunktbildung"
function [Ergebnis]=KugSchn(M1,M2,M3,r1,r2,r3,pos)
```

```
% Verschiebung der Mittelpunkte in den Ursprung:
M2v=M2-M1;
M3v=M3-M1;
M11=[0;0;0];
% bilden der Einheitsvektoren:
ex=(M2v-M11)/norm(M2v-M11);
ez=cross(M3v,M2v)/norm(cross(M3v,M2v));
ey=cross(ez,ex)/norm(cross(ez,ex));
% Koordinatentransformationsmatrix:
Mab=[ex ey ez]';
% Mittelpunkte im lokalen Koordinatensystem:
M21=Mab*M2v;
M31=Mab*M3v;
% Berechnung der x-Komponente über Pythagoras:
x=(r1^2-r2^2+(norm(M21))^2)/(2*norm(M21));
r12=sqrt(abs(r1^2-x^2));
% Projizierter Radius des Schnittkreises von der Kugel 3:
if(x>=M31(1))
   r123y=x-M31(1);
else
   r123y=M31(1)-x;
end
r123=sqrt(abs(r3^2-r123y^2));
% Berechnung der y-Komponente:
y=(r12^2-r123^2+M31(2)^2)/(2*M31(2));
% Berechnung der z-Komponente über Phytagoras:
z=sqrt(abs(r12^2-y^2));
% Wahl des Ergebnisses:
if(pos==1)
   SP=[x;y;-z];
else
   SP=[x;y;z];
end
% Rücktransformation und Rückverschiebung:
Ergebnis=((Mab'*SP)+M1);
```

Kootrans.m

```
% Koordinatentransformation
% Unterfunktion von "FWkomplett.m"
% Input: zu transformierende Matrix, Rotationswinkel um x-, y-, z-Achse
% Output: transformierte Matrix
function [U]=KooTrans(u,winkel1,winkel2,winkel3)

%   Umrechnung deg in rad
alpha1=winkel1/180*pi;
alpha2=winkel2/180*pi;
alpha3=winkel3/180*pi;

%   Rotationsmatrizen werden aufgestellt
A1=[1 0 0;0 cos(alpha1) -sin(alpha1); 0 sin(alpha1) cos(alpha1)];
A2=[cos(alpha2) 0 sin(alpha2);0 1 0;-sin(alpha2) 0 cos(alpha2)];
A3=[cos(alpha3) -sin(alpha3) 0;sin(alpha3) cos(alpha3) 0;0 0 1];

%   Ausgabe des Transformierten Vektors/Matrix
U=A1*A2*A3*u;
```

Anlagen, Teil 2

Quelltexte der symbolischen Lösung des Beispiels:

ODEsolver.m

```
clc
clear

% Festlegung von Größen
Load_s;

%%%% Anfangswerte %%%
y(1)=2;        %   Vorspannweg von x1
y(2)=3;        %   Anfangsgeschwindigkeit von x1
a=solve(f(2),x2);  %   Vorspannweg von x2
% Laden der numerisch vereinbarten Variablen
Load_n;
y(3)=subs(a);
y(4)=0;        %   Anfangsgeschwindigkeit von x2

% Bildung von y(5) über Umstellen der Bewegungsgleichungen mit Zwang
[z b1 b2]=Zwang(n);
B=BewGl(n)-z;
A=solve(B(2),lambda);
A=subs(A,xp(4),b2);
A=subs(A,xp(2),b1);
A=subs(A,x1,y(1));
A=subs(A,x2,y(3));
y(5)=A;

% Solver
[T,Y] = ode45(@Handlefun,[0 10],y);

% Plot
subplot((2*n+1),1,1); plot(T,Y(:,1),'-'),title('x1 - Weg der Masse 1');
subplot((2*n+1),1,2); plot(T,Y(:,2),'-'),title...
    ('x1p - Geschwindigkeit der Masse 1');
subplot((2*n+1),1,3); plot(T,Y(:,3),'-'),title('x2 - Weg der Masse 2');
subplot((2*n+1),1,4); plot(T,Y(:,4),'-'),title...
    ('x2p - Geschwindigkeit der Masse 2');
subplot((2*n+1),1,5); plot(T,Y(:,5),'-'),title('Lambda');
```

Handlefun.m

```
% Handlefun bereitet das zu lösende Gleichungssystem für den ODEsolver
auf
function dy = Handlefun(~,y)

Load_n;
dy = zeros(2*n+1,1);     % Aufstellen des Vektors des GLS
```

```
A=Matbuild(n);          % Erzeugen der Systembeherrschenden Matrix
Load_n;
B=subs(A);              % Einsetzen der Werte

b=Vectorb(y,n);         % Erzeugen des Vektors "b"

yp=B\b;                 % Erzeugen des GLS


for i=1:2*n+1;
    dy(i)=yp(i);
end
```

Matbuild.m

```
% Erstellung der Systembeherrschenden Matrix
% Die Systembeherrschende Matrix B hängt von der Anzahl der
% Freiheitsgrade
% n und der Lagrangefunktion L mit deren Variablen ab.
% L wird über das Skript Load_s.m geladen.
function[B]=Matbuild(n)

Load_s;                         % Laden der Lagrangefunktion

x1=x(2:2:2*n);                  % Erzeugung des charakteristischen Teils
L1=jacobian(L,x1);
x2=x(1:2*n);
L2=jacobian(L1,x2);

L2=horzcat(L2,zeros(n,1));      % Erweiterung um eine Spalte

C=zeros(n+1,2*n+1);             % Erzeugung des trivialen Teils

i=1;
for j=1:2:2*n+1;
    C(i,j)=1;
    i=i+1;
end

B = vertcat(L2,C);             % Zusammenstellung der Matrizen
```

Vektorb.m

```
function[b]=Vectorb(y,n)

Load_s;                             % laden der symbolischen Vereinbarun-
gen

[z b1 b2]=Zwang(n);                     % Erzeugen des Zwangs

B=BewGl(n)-z;                       % Erzeugen der Bewegungsgleichung

for i=1:n
    if z(i)~=0
```

```
Bz=subs(B(i),x2pp,b2);                % Aufbereiten des Zwanges
Bz=solve(Bz,lambda);
Bz=diff(Bz,x2)*x2p+diff(Bz,x1)*x1p+diff(Bz,x2p)*x2pp+diff(Bz,x1p)*x1pp;
    end
end
xp=subs(xp,lambdap,Bz);

a=Matbuild(n)*xp;                     % Vektor wird erzeugt

% Zwischenvektor wird festgelegt
d=sym('[d1;d2;d3;d4;d5;d6;d7;d8;d9;d10;d11]');

for i=1:n
    B(i)=subs(B(i),a(i),d(i));        % Variable wird ersetzt
end

g=vpa(1:n);                           % Vektor wird erzeugt
v=g';

for i=1:n
    v(i)=solve(B(i),d(i));            % Umstellen
end

a=vertcat(v,a(n+1:2*n+1));            % Umgeformter Vektor

Load_n;

x1=y(1);                              % Einsetzen der Zahlenwerte
x1p=y(2);                             %%% von HAND!
x2=y(3);
x2p=y(4);
lambda=y(5);

b=subs(a);
```

BewGl.m

```
% Erzeugen der Bewegungsgleichungen:
function[B]=BewGl(~)

Load_s;

dLdx1=diff(L,x1);           % Ableitung L nach x1
dLdx2=diff(L,x2);           % Ableitung L nach x2

dLdx1p=diff(L,x1p);         % Ableitung L nach x1 Punkt
dLdx2p=diff(L,x2p);         % Ableitung L nach x2 Punkt

% Ableitung L nach x1 Punkt nach der Zeit
ddLdx1p=diff(dLdx1p,x1p)*x1pp+diff(dLdx1p,x1)*x1p;
% Ableitung L nach x2 Punkt nach der Zeit
ddLdx2p=diff(dLdx2p,x2p)*x2pp+diff(dLdx2p,x2)*x2p;

B1=(ddLdx1p-dLdx1);           % Bewegungsgleichung 1
B2=(ddLdx2p-dLdx2);           % Bewegungsgleichung 2
```

```
B=[B1;B2];
```

Zwang.m

```
% Zwang wird für die Einbindung in den Vektor b erzeugt
function[z,b1,b2]=Zwang(n)

Load_s;                                 % Laden der Variablen

zw=vpa(1:n);

for i=1:n                               % Erzeugen des Zwang-Vektors
    zw(i)=lambda*diff(f(i),x(2*i-1));
end
a=conj(zw);                             % Umformung des Vektors
z=a';

% Unautomatisierte Ablauf:
B21=diff(f(2),x2);
B01=diff(f(2),x2)*x2p;

B=B21*x2p+B01;
b1=solve(B,x2p);

b2=diff(B,x2p)*x2pp+diff(B,x2)*x2p+diff(B,x1p)*x1pp+diff(B,x1)*x1p;

b2=subs(b2,x2p,b1);
b2=solve(b2,x2pp);
```

Load_s.m

```
syms x1 x1p x1pp x2 x2p x2pp m c Fc lambda lambdap;

Ekin=(1/2)*m*x1p^2+(1/2)*m*x2p^2;                   % kin E
Epot=(1/2)*c*x1^2+(1/2)*c*x2^2+(1/2)*c*(x2-x1)^2;   % pot E

L=Ekin-Epot;                                        % Lagrange Funktion

f=[0;Fc-c*x2];                                      % Festgelegter Zwang

x=[x1;x1p;x2;x2p;lambda];                  % Festlegung der Freiheits-
grade
xp=[x1p;x1pp;x2p;x2pp;lambdap];
```

Load_n.m

```
% Deklaration der numerischen Konstanten

n=2;        % Anzahl Freiheitsgrade
m=1.5;      % Masse in kg
c=5;        % Federsteifigkeit in N/mm
Fc=25;      % Vorspannkraft in N
```

Anlagen, Teil 3

Quelltexte des numerischen Grundprogramms:

ODEsolver.m

```
clc
clear
% Hauptprogramm zur numerischen Lösung von dynamischen Problemen
% Abhängig von der Anzahl der Freiheitsgrade, Anzahl der Zwänge, der
% Lagrangefunktion und von geeigneten Anfangswerten


% Beispiel für Zweimassenschwinger mit einem Zwang

% festlegen der Anzahl der Freiheitsgrade
global n;
n=2;
% festlegen der Anzahl der Zwänge
global u;
u=1;

%%%% Anfangswerte %%%%
% müssen sinnvoll gewählt werden

y(1)=2;        %   Vorspannweg von x1
y(2)=3;        %   Anfangsgeschwindigkeit von x1
y(3)=5;        %   Vorspannweg von x2
y(4)=0;        %   Anfangsgeschwindigkeit von x2

% Aufruf des Lösers für Differentialgleichungssysteme
% abhängig von der Handlefun-Funktion
% Zeitintervall ist beliebig wählbar
% Ausgabe der Lösungen und der zugehörigen Zeiten
[T,Y] = ode45(@Handlefun,[0 10],y);

% Plot muss vom Nutzer entsprechend angepasst werden
subplot((2*n+1),1,1); plot(T,Y(:,1),'-'),title('x1 - Weg der Masse 1');
subplot((2*n+1),1,2); plot(T,Y(:,2),'-'),...
       title('x1p - Geschwindigkeit der Masse 1');
subplot((2*n+1),1,3); plot(T,Y(:,3),'-'),...
       title('x2 - Weg der Masse 2');
subplot((2*n+1),1,4); plot(T,Y(:,4),'-'),...
       title('x2p - Geschwindigkeit der Masse 2');

% Ausgabe ausgewählter Ergebnisse in Tabellenform zur Visualisierung in
% CAD-Systemen
% Muss vom Nutzer entsprechend angepasst werden
M1=horzcat(T,Y(:,1));
M2=horzcat(T,Y(:,3));
csvwrite('ZMS1.dat',M1)
csvwrite('ZMS2.dat',M2)
```

Handlefun.m

```
% Handlefun bereitet das zu lösende Gleichungssystem für den ODEsolver
auf
function[dy]=Handlefun(t,y)

% Laden der Anzahl der Freiheitsgrade und der Anzahl der Zwänge
global n;
global u;

% Aufstellen des Vektors des GLS
dy=zeros(2*n,1);

% Erzeugen der systembeherrschenden Matrix in Unterfunktion
A=Matbuild_n(t,y);

% Erzeugen des systembeherrschenden Vektors in Unterfunktion
b=Vektorb_n(t,y);

% erzeugen der Zwänge in Unterfunktion
[qlambda Biu]=AufZwang(t,y);
% Auswahl der lambdas
lambda=qlambda(n+1:n+u,1);
% Bildung der Zwangskräfte
Zi=Biu*lambda;
% Vektor wird mit Nullen erweitert
z=zeros(2*n,1);
z(1:n,1)=Zi;

% Erzeugen des GLS mit Zwängen
yp=A\(b-z);

% ersetzen des Null-Vektors zur Übergabe an den ODE
for i=1:2*n;
    dy(i)=yp(i);
end
```

Matbuild_n.m

```
% Erstellung der systembeherrschende Matrix
function[B]=Matbuild_n(t,y)

% Laden der Anzahl der Freiheitsgrade
global n;

% Aufbereiten der Variablen
var=Variablen(t,y);

% erzeugen der Grundmatrix
A=zeros(n,2*n);

% Auffüllen der Matrix mit den jeweiligen zweifachen Ableitungen
for i=1:n
    for j=1:n
    A(i,2*j-1)=diff_2(@Lagrange,1,i,2,j,var,0.0001,0.0001);
    A(i,2*j)=diff_2(@Lagrange,1,i,1,j,var,0.0001,0.0001);
    end
end
```

```
% Erzeugung des trivialen Teils
C=zeros(n,2*n);

% Auffüllen des trivialen Teils mit 1en an den jeweiligen Stellen
i=1;
for j=1:2:2*n;
    C(i,j)=1;
    i=i+1;
end

% Zusammensetzen der Matrix
% Übergabe der systembeherrschenden Matrix
B = vertcat(A,C);
```

Variablen.m

```
% Aufbereiten der Variablen um dem universellen Anspruch gerecht zu wer-
den
function[var]=Variablen(t,y)

% Laden der Anzahl der Freiheitsgrade
global n;

% n Geschwindigkeiten, n Wege, eine Zeit
var=zeros(3,n);

% Geschwindigkeiten werden den y zugeordnet
for i=1:n
    var(1,i)=y(2*i);
end

% Wege werden den y zugeordnet
for i=1:n
    var(2,i)=y(2*i-1);
end

% Zeit
var(3,1)=t;

% auffüllen der Matrix
var(3,2:n)=0;
```

diff_2.m

```
% quadratische Differetiation nach ausgewählten Variablen
% Input: Name des Abzuleitenden Funktionshandle
%        Variable, nach der als erstes abgeleitet werden soll
%        Variable, nach der als zweites abgeleitet werden soll
%        Die Funktionsargumente
%        die Differentiationsschritteiten der einzelnen Differentiationen
function [returnval] =
diff_2(func,number11,number12,number21,number22,...
                        arg,step1,step2)

% Funktionswert (erste partielle Ableitung) an der Stelle
res1=diff_1(func,number11,number12,arg,step1);

% Funktionswert ein wenig weiter
temp=arg(number21,number22)+step2;
arg(number21,number22)=temp;
```

```
% Funktionswert (erste partielle Ableitung) an der Stelle
res2=diff_1(func,number11,number12,arg,step1);

% Rückgabe des Differenzenquotient
returnval=(res2-res1)/step2;
```

diff_1.m

```
% Differetiation nach ausgewählter Variable
% Input: Name des Abzuleitenden Funktionshandle
%        Variable, nach der abgeleitet werden soll
%        Die Funktionsargumente
%        die Differentiationsschrittwete der Differentiation
function [returnval] = diff_1(func,number1,number2,arg,step1)

% Funktionswert an der Stelle
res1=func(arg);

% Funktionswert ein wenig weiter
temp=arg(number1,number2)+step1;
arg(number1,number2)=temp;

% Funktionswert rechnen
res2=func(arg);

% Differenzenquotient zurück
returnval=(res2-res1)/step1;
```

Lagrange.m

```
% Lagrangefunktion wird vorgegeben
% wird mit Freiheitsgraden (Geschwindigkeiten & Wegen) und der Zeit
% bestückt
function[Lag]=Lagrange(in)

% Geschwindigkeiten werden neu deklariert
qp=in(1,:);

% Wege werden neu deklariert
q=in(2,:);

% Zeit wird neu deklariert
t=in(3,1);

% Konstanten werden festgelegt

% Masse
m=1.5;
% Federsteifigkeit
c=5;

% Lagrangefunktion wird vorgegeben
Lag=0.5*m*(qp(1))^2+0.5*m*(qp(2))^2-0.5*(c*(q(1))^2+...
    c*(q(2))^2+c*(q(2)-q(1))^2);
```

Vektorb_n.m

```
% bildet systembeherrschenden Vektor
function[v]=Vektorb_n(t,y)

% Laden der Anzahl der Freiheitsgrade
global n;

% Aufbereiten der Variablen
var=Variablen(t,y);

% oberer Teil des Vektors (trivialer Teil)
a=zeros(n,1);

% Ausführen der Differentiation nach den Wegen
for i=1:n
    a(i,1)=diff_1(@Lagrange,2,i,var,0.0001);
end

% unterer Teil des Vektors wird erzeugt
b=zeros(n,1);
for i=1:n
    b(i,1)=y(2*i);
end

% endgültiger Vektor
v=vertcat(a,b);
```

AufZwang.m

```
% Funktion zur Aufbereitung der Zwänge
% Hier werden die Ableitungen durchgeführt, um die lambdas zu bestimmen
% Auch die Biu werden erzeugt um die Zwangskräfte zu berechnen
% AufZwang hängt von der übergebenen Zeit und den übergebenen y ab
function[qlambda Biu]=AufZwang(t,y)

% laden der globalen Variablen
% u... Anzahl der Zwänge
% n... Anzahl der Freiheitsgrade/generalisierte Koordinaten
global u;
global n;

% globalisieren des handles-Vektors
global handles;

% 0. Variablen %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% erzeugen der Variablenmatrix mit den Geschwindigkeiten,
% Wegen und der Zeit- abhängig von n
var=Variablen(t,y);

% 1. Zwänge als Funktionhandle Vektor speichern %%%%%%%%%%%%%%%%%%%%%%%%%%%

% Die Zangsfunktionen werden zum Handle umgewandelt um das Bilden der
% Ableitungen zu verallgemeinern.
% Die einzelnen Zwangsfunktionen müssen vom Nutzer selbst festgelegt wer-
den
% Dazu ist wie in der Funktion "Zwang_1.m" vorzugehen
% Wenn u > 9 muss der Nutzer entsprechende Zeilen hinzufügen
```

```
handles{1}=@Zwang_1;
handles{2}=@Zwang_2;
handles{3}=@Zwang_3;
handles{4}=@Zwang_4;
handles{5}=@Zwang_5;
handles{6}=@Zwang_6;
handles{7}=@Zwang_7;
handles{8}=@Zwang_8;
handles{9}=@Zwang_9;

% 2. erste Ableitung der Zwänge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% B_i_mue wird erzeugt:

% erzeugen einer 0-Matrix welche später durch die Ableitungen bestückt
wird
Biu=zeros(n,u);

% B_i_mue wird durch ableiten der Zwänge nach den Wegen (Zwänge dürfen
ohne
% hin nur von den Wegen und der Zeit abhängen) erzeugt
% Der Zwang wird, um den universellen Charakter zu entsprechen, von der
% Funktion "Bcreate.m" aufgerufen

% In "Bcreate.m" gehen ein:
% mue: Anzahl der Zwänge um die Handles aufzurufen
% var: um die Variablenmatrix bereit zu halten
% 2: um nach der zweiten Zeile von var (Wege) abzuleiten
% i: Laufvariable für die generalisierten Koordinaten
% als letztes der Differentiationsschritt step

% in "Bcreate.m" werden dann die Ableitungen durch den Aufruf der
% "diff_1.m" Funktion ausgeführt
% Mit dieser Vorgehensweise lässt sich die Übergabe der Handles der Zwän-
ge
% in die Differentiationsfunktion automatisieren um das Programm univer-
sal
% zu halten
for i=1:n
    for mue=1:u
        Biu(i,mue)=Bcreate(mue,var,2,i,0.0001);
    end
end
Bl=Biu;

%%% B_0_mue wird erzeugt:

% erzeugen einer 0-Matrix welche später durch die Ableitungen bestückt
wird
Bou=zeros(u);

% Analoges Vorgehen wie für B:i:mue
% Unterschied: es wird nur nach der Zeit abgelitten (dritte Zeile von
var)
% somit keine Abhängigkeit von den Wegen und somit keine Schleife über i
for mue=1:u
        Bou(mue)=Bcreate(mue,var,3,1,0.0001);
end

% 3. zweite Ableitung der Zwänge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% B_i_mue_Punkt wird erzeugt:

% erzeugen einer 0-Matrix welche später durch die Ableitungen bestückt
wird
Biup=zeros(n,u);

% Die Ableitung von B_i_mue nach der Zeit wird gebildet.
% Zusätzlich erfolgt die Multiplikation mit den Geschwindigkeiten aus der
% ersten Zeile des "var"-Vektors und die Aufsumierung der Ergebnisse.
% Zur Differentiation wird an die Funktion "diff_B.m" die Ableitung aus
der
% in "Bcreate.m" gebildeten Funktion als Handle übergeben.
% "diff_B.m" gewährleistet nun die einzelnen Zwänge abzubearbeiten, da
die
% Laufkoordinate "mue" mit eingeht.
for i=1:n
    for mue=1:u
        for j=1:n
        Biup(i,mue)=Biup(i,mue)...
                    +diff_B(@Bcreate,mue,var,2,i,0.0001,2,j)*var(1,j);
        end
        Biup(i,mue)=Biup(i,mue)+diff_B(@Bcreate,mue,var,2,i,0.0001,3,1);
    end
end

%%% B_0_mue_Punkt wird erzeugt:

% erzeugen einer 0-Matrix welche später durch die Ableitungen bestückt
wird
Boup=zeros(u);

% Das Prozedere läuft nun analog wie bei Biup ab, mit dem Unterschied
% das nach der Zeit abgelitten wird.
for mue=1:u
    for j=1:n
        Boup(mue)=Boup(mue)...
                  +diff_B(@Bcreate,mue,var,3,1,0.0001,2,j)*var(1,j);
    end
        Boup(mue)=Boup(mue)+diff_B(@Bcreate,mue,var,3,1,0.0001,3,1);
end

% 4. Bau des Gleichungssystems + Lösung %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% erzeugen der rechten Seite des zu korrigierenden Grundgleichungssystems

Br=zeros(u,1);
for j=1:u
    for i=1:n
        Br(j)=Br(j)+Biup(i,j)*var(1,i);
    end
end
Bo=zeros(u,1);
for j=1:u
    Bo(j)=Boup(j);
end

Br=-Br-Bo;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Das "MatSort.m" File eliminiert schlecht konditionierte Gleichungen.
Dazu
% werden die Beträge der Summen aller Zeilen verglichen und die mit den
% geringsten Beträgen gestrichen. Übrig bleiben nur so viele Zeilen wie
% Anzahl der Zwänge

% Es hat sich heraus gestellt, das immer alle Zeilen benötigt werden, da
% die Anzahl der Zeilen immer der Anzahl der Zwänge entspricht.

% Die Unterfunktion ist so aufgebaut, das auch für den Fall, das keine
% Zeile gestrichen werden müssen, auch keine gestrichen werden.

% schlecht konditionierte Gleichungen werden gestrichen:
rv=MatSort(Biu',Br,u);

% korrigiertes GLS:

for j=1:u
    Bl(j)=Biu(rv(j));
    Br(j)=Br(rv(j));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Aufruf und Anpassung der Systembeherrschenden Matrix
C=Matbuild_n(t,y);
C1=C(1:n,2:2:2*n);
C2=C(1:n,1:2:2*n);

% Aufruf und Anpassung des Systembeherrschenden Vektors
a=zeros(n,1);
for i=1:n
    a(i,1)=diff_1(@Lagrange,2,i,var,0.0001);
end
C2=-C2*(var(1,1:n))'+a;

%%% Bau des SUPER-GLS

% Bau der Matrix
B=vertcat(C1,Bl');

D=zeros(n+u,u);
for j=1:u
    for i=1:n
        D(i,j)=Biu(i,j);
    end
end
B=horzcat(B,D);

%Bau des Vektors
b=vertcat(C2,Br);

% Der Übergabevektor enthält nun die qipp und die Lambdas
qlambda=B\b;
```

Bcreate.m

```
% Funktion zur Koordination der Zwang-Handles
% Hier werden über die Laufvariable mue die Zwänge mit der "diff_1.m"
% Funktion abgelitten.
% Dazu werden die relevanten Informationen übergeben
```

```
function B =Bcreate(mue,var,k,l,step)

% Die Handles der Zwänge werden geladen
global handles;

% Die Differentiation wird ausgeführt
  B=diff_1(handles{mue},k,l,var,step);
```

Zwang_1.m

```
% Zwangfunktion
% bei mehreren Zwängen muss die Funktion entsprechend gespeichert werden:
% Zwang_1.m, Zwang_2.m, Zwang_3.m, ... usw.
function f1 =Zwang_1(in)
% Platzhalter fuer Zwang
% Eingabewerte bei Zwang sind Wege, Zeit da holonome Zwänge nur davon
% abhängen dürfen

q=in(2,:);      % holonome Zwänge niemals von Geschwindigkeiten abhängig
t=in(3,1);

% Konstanten festlegen/ die in Zwangsgleichung vorkommen.
Fc=5;
c=5;

% Zwang formulieren
% Bsp:
f1=Fc-c*q(2);
```

diff_B.m

```
function [returnval] = diff_B(func,mue,arg,j,i,step,number1,number2)

% Funktionswert an der Stelle/ mue ist die Laufvariable für den Zwang
res1=func(mue,arg,j,i,step);

% Funktionswert ein wenig weiter
temp=arg(number1,number2)+step;
arg(number1,number2)=temp;

% Funktionswert rechnen
res2=func(mue,arg,j,i,step);

% Differenzenquotient zurück
returnval=(res2-res1)/step;
```

MatSort.m

```
% Funktion ermittelt die relevanten Zeilen des Gleichungssystems
% Eingabe der Matrix, des Vektors und der Anzahl der releveanten Zeilen
% die Anzahl der relevanten Zeilen entspricht dabei der Anzahl der Zwänge
% Die Matrizen müssen dabei Zeilenweise eingegeben werden
% Der Vektor muss als Spaltenvektor eingegeben werden
function rv =MatSort(A,a,m)

% Berechnung der Ausdehnung der Matrix
s=size(A);
% bilden der Beträge der Matrix und des Vektors
A=abs(A);
```

```
a=abs(a);

% Summierung der Zeilen + dem Vektor
A=A';
S=(sum(A))'+a;

g=(1:s(1))';

V=horzcat(g,S);

B=sortrows(V,2);

r=B(:,1);

rv=sortrows(r((s(1)-m)+1:s(1)));
```

Anlagen, Teil 4

Quelltexte der Anwendung am Rennwagen:

Jene Quelltexte, die sich im Vergleich zum Grundprogramm geändert haben, oder neu dazugekommen sind.

ODEsolver.m

```
clear
clc

% festlegen der Anzahl der Freiheitsgrade
global n;
n=10;
% festlegen der Anzahl der Zwänge
global u;
u=0;
% Position des Schwerpunktes des Rumpfes
global SP;
SP=[825;310:0];
% festgelegen des Lenkwinkels in °
global lw;
lw=0;
% festlegen der Lenkgeschwindigkeit
global lwp;
lwp=0;

%%%% Anfangswerte %%%%
% Abwechselnd Wege und Geschwindigkeiten

y(1)=160;       %   Länge der Feder vorn links
y(2)=0;       %   Anfangsgeschwindigkeit der Feder vorn links
y(3)=188.66;       %   Länge der Feder hinten links
y(4)=0;       %   Anfangsgeschwindigkeit der Feder hinten links
y(5)=188.66;       %   Länge der Feder vorn rechts
y(6)=0;       %   Anfangsgeschwindigkeit der Feder vorn rechts
y(7)=188.66;       %   Länge der Feder hinten rechts
y(8)=0;       %   Anfangsgeschwindigkeit der Feder hinten rechts
y(9)=0;       %   Translation in X-Richtung
y(10)=0;      %   Translationsgeschwindigkeit in X-Richtung
y(11)=0;      %   Translation in Y-Richtung
y(12)=0;      %   Translationsgeschwindigkeit in Y-Richtung
y(13)=0;      %   Translation in Z-Richtung
y(14)=0;      %   Translationsgeschwindigkeit in Z-Richtung
y(15)=0;      %   Rotationswinkel um X-Achse
y(16)=0;      %   Winkelgeschwindigkeit um X-Achse
y(17)=0;      %   Rotationswinkel um Y-Achse
y(18)=0;      %   Winkelgeschwindigkeit um Y-Achse
y(19)=0;      %   Rotationswinkel um Z-Achse
y(20)=0;      %   Winkelgeschwindigkeit um Z-Achse
```

```
%y=Anfang(y);

% Solver
[T,Y] = ode45(@Handlefun,[0 2],y);

% Plot
subplot((2*n+1),1,1); plot(T,Y(:,1),'-'),title('Vorspannweg der Feder
vorn links');
subplot((2*n+1),1,2); plot(T,Y(:,2),'-'),title('Anfangsgeschwindigkeit
der Feder vorn links');
subplot((2*n+1),1,3); plot(T,Y(:,3),'-'),title('Vorspannweg der Feder
hinten links');
subplot((2*n+1),1,4); plot(T,Y(:,4),'-'),title('Anfangsgeschwindigkeit
der Feder hinten links');
subplot((2*n+1),1,5); plot(T,Y(:,5),'-'),title('Vorspannweg der Feder
vorn rechts');
subplot((2*n+1),1,6); plot(T,Y(:,6),'-'),title('Anfangsgeschwindigkeit
der Feder vorn rechts');
subplot((2*n+1),1,7); plot(T,Y(:,7),'-'),title('Vorspannweg der Feder
hinten rechts');
subplot((2*n+1),1,8); plot(T,Y(:,8),'-'),title('Anfangsgeschwindigkeit
der Feder hinten rechts');
subplot((2*n+1),1,9); plot(T,Y(:,9),'-'),title('Translation in X-
Richtung');
subplot((2*n+1),1,10); plot(T,Y(:,10),'-
'),title('Translationsgeschwindigkeit in X-Richtung');
subplot((2*n+1),1,11); plot(T,Y(:,11),'-'),title('Translation in Y-
Richtung');
subplot((2*n+1),1,12); plot(T,Y(:,12),'-
'),title('Translationsgeschwindigkeit in Y-Richtung');
subplot((2*n+1),1,13); plot(T,Y(:,13),'-'),title('Translation in Z-
Richtung');
subplot((2*n+1),1,14); plot(T,Y(:,14),'-
'),title('Translationsgeschwindigkeit in Z-Richtung');
subplot((2*n+1),1,15); plot(T,Y(:,15),'-'),title('Rotationswinkel um X-
Achse');
subplot((2*n+1),1,16); plot(T,Y(:,16),'-'),title('Winkelgeschwindigkeit
um X-Achse');
subplot((2*n+1),1,17); plot(T,Y(:,17),'-'),title('Rotationswinkel um Y-
Achse');
subplot((2*n+1),1,18); plot(T,Y(:,18),'-'),title('Winkelgeschwindigkeit
um Y-Achse');
subplot((2*n+1),1,19); plot(T,Y(:,19),'-'),title('Rotationswinkel um Z-
Achse');
subplot((2*n+1),1,20); plot(T,Y(:,20),'-'),title('Winkelgeschwindigkeit
um Z-Achse');

% Ausgabe ausgewählter Ergebnisse in Tabellenform
M1=horzcat(T,Y(:,1));
M2=horzcat(T,Y(:,3));
M3=horzcat(T,Y(:,5));
M4=horzcat(T,Y(:,7));
M5=horzcat(T,Y(:,9));
M6=horzcat(T,Y(:,11));
M7=horzcat(T,Y(:,13));
M8=horzcat(T,Y(:,15));
M9=horzcat(T,Y(:,17));
M10=horzcat(T,Y(:,19));
csvwrite('ZMS1.dat',M1)
csvwrite('ZMS2.dat',M2)
csvwrite('ZMS3.dat',M3)
csvwrite('ZMS4.dat',M4)
```

```
csvwrite('ZMS5.dat',M5)
csvwrite('ZMS6.dat',M6)
csvwrite('ZMS7.dat',M7)
csvwrite('ZMS8.dat',M8)
csvwrite('ZMS9.dat',M9)
csvwrite('ZMS10.dat',M10)
```

Lagrange.m

```
function[Lag]=Lagrange(in)
% Platzhalter fuer Lagrange-Funktion
% Eingabewerte bei Lagrange-Funktion sind Geschwindigkeiten, Wege, Zeit

qp=in(1,:);
q=in(2,:);
t=in(3,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Konstanten

m=15;    % kg      Masse/Rad
M=240;   % kg      Masse Fahrzeugrumpf + Fahrer
c=25;    % N/mm    Steifigkeit der Federn
% Trägheitstensor Rad
j=[892518 0 0;0 496135 0;0 0 496135]; % in kg*mm^2
% Bezogen auf Schwerpunkt
% lokale x-Achse = Rotationsachse wenn sich Rad dreht

% Trägheitstensor Rumpf
J=[40762581 12553794 -602;12553794 168655866 -445;-602 -445 179441266];
% in kg*mm^2

% Federlänge in Ausgangslage:
f=188.66;    % in mm

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%global Omm;
Omm=Ommega(in);
% Berechnung der Ommegavektoren/Rad, Schwerpunktvektoren/Rad, Ge-
schwvek/Rad
% bezogen auf das globale System
w=Omm(:,1:4);
SPv=Omm(:,5:8);

% Transformierung der Ommegavektoren in lokale Rad-Koordinatensysteme
w=w_Trans(w);

% Berechnung des Ommegavektors des Rumpfes bezogen auf den Schwerpunkt
W=[qp(8);qp(9);qp(10)];

% Transformierung von W in lokale Rumpf-Koordinaten
W=WTrans(W);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Lagrange-Funktion
```

```
% kinetische Energie:

% kinetische Energien der Räder
Ekr1=0.5*m*(SPv(1,1))^2+0.5*m*(SPv(2,1))^2+0.5*m*(SPv(3,1))^2+...
     0.5*(w(:,1))'*j*(w(:,1));
Ekr2=0.5*m*(SPv(1,2))^2+0.5*m*(SPv(2,2))^2+0.5*m*(SPv(3,2))^2+...
     0.5*(w(:,2))'*j*(w(:,2));
Ekr3=0.5*m*(SPv(1,3))^2+0.5*m*(SPv(2,3))^2+0.5*m*(SPv(3,3))^2+...
     0.5*(w(:,3))'*j*(w(:,3));
Ekr4=0.5*m*(SPv(1,4))^2+0.5*m*(SPv(2,4))^2+0.5*m*(SPv(3,4))^2+...
     0.5*(w(:,4))'*j*(w(:,4));
% kinetische Energie des Rumpfes
Ek=0.5*M*qp(5)^2+0.5*M*qp(6)^2+0.5*M*qp(7)^2+0.5*W'*J*W;

% Summe der kinetischen Energien
Ekin=Ekr1+Ekr2+Ekr3+Ekr4+Ek;

% potentielle Energie:
% Änderung der Federlänge zur Ausgangslage
Epot=(c/2)*(norm((q(1))-f)^2+norm((q(2))-f)^2+norm((q(3))-f)^2+...
     norm((q(4))-f)^2);

% endgültige Lagrange Funktion:
Lag=Ekin-Epot;
```

Ommega.m

```
% Berechnung der Ommegavektoren der Räder
% Ortsvektoren der Radschwerpunkte
% Geschwindigkeitsvektoren der Radschwerpunkte
% Abhängig von Eingangsvariablen und Lenkkennwerten
function[Omm]=Ommega(in)

% Laden der Anzahl der Freiheitsgrade
global n;
global lw;
global lwp;

% Erweiterung der Variablenmatrix um eine Spalte um Lenkeinschlag und
% Lenkgeschwindigkeit als festgelegten Faktor mit einzubeziehen
m=n+1;
in(1,m)=lwp;
in(2,m)=lw;

% Geschwindigkeiten werden neu deklariert
qp=in(1,:);

% Die Ortsvektoren der Punkte werden bestimmt
global K;
K=Kinematik(in);
r=K(:,9:20);
r=r';
% anlegen der Matrix der Geschwndigkeiten
v=zeros(12,4);

% Berechnung der Geschwindigkeitsvektoren in Matrixform
for i=1:m
    v=v+(diff_1(@Radvektoren,2,i,in,0.01))*qp(i);
end
```

```
% Bildung der Differenzvektoren der Ortsvektoren
rd(1:12,1:3)=[r(1:12,2)-r(1:12,1),r(1:12,3)-r(1:12,2),r(1:12,3)-
r(1:12,1)];

% Bildung der Differenzvektoren der Geschwindigkeitsvektoren
vd(1:12,1:3)=[v(1:12,2)-v(1:12,1),v(1:12,3)-v(1:12,2),v(1:12,3)-
v(1:12,1)];


% Auswahl der relevanten Vektoren
vm=VMax(rd,vd);

% Preparierung der relevanten Vektoren VL
rvl=rd(1:3,1:3);
r1=rvl(:,vm(1));
vvl=vd(1:3,1:3);
v1=vvl(:,vm(1));

% Preparierung der relevanten Vektoren HL
rhl=rd(4:6,1:3);
r2=rhl(:,vm(2));
vhl=vd(4:6,1:3);
v2=vhl(:,vm(2));

% Preparierung der relevanten Vektoren VR
rvr=rd(7:9,1:3);
r3=rvr(:,vm(3));
vvr=vd(7:9,1:3);
v3=vvr(:,vm(3));

% Preparierung der relevanten Vektoren HR
rhr=rd(10:12,1:3);
r4=rhr(:,vm(4));
vhr=vd(10:12,1:3);
v4=vhr(:,vm(4));

% resultierenden Ommegas für jedes Rad
w1=(cross(r1,v1))/(norm(r1))^2;
w2=(cross(r2,v2))/(norm(r2))^2;
w3=(cross(r3,v3))/(norm(r3))^2;
w4=(cross(r4,v4))/(norm(r4))^2;

% Ausgabe der Ommegas in Matrixform
% jede Spalte steht für ein Rad
w=horzcat(w1,w2,w3,w4);

% Ortsvektoren der Schwerpunkte
% jede Spalte steht für ein Rad
%SPr=horzcat(r(1:3,4),r(4:6,4),r(7:9,4),r(10:12,4));

% Geschwindigkeitsvektoren der Schwerpunkte
% jede Spalte steht für ein Rad
SPv=horzcat(v(1:3,4),v(4:6,4),v(7:9,4),v(10:12,4));

Omm=horzcat(w,SPv);
```

Radursprünge.m

```
% Hauptprogramm zur vollständigen Bestimmtheit des Fahrzeugs im Raum
% Input: Länge der Federn: l_VL, l_HL, l_VR, l_HR [mm] (Max 200 /Min 130)
%        Winkel Lenkeinschlag: winkel [°] (Max 120 /Min -120)
%        Translation des Rahmens in X, Y, Z
%        Rotationswinkel alpha1 (um x-Achse), alpha2 (um y), alpha3 (um
z)
% Output: 3 Radvektoren / Rad
% integrierte Programme: "solve_VL.m", "solve_HL.m", "solve_VR.m",
%                        "solve_HR.m", "Kootrans.m", "InputKoord.m"
function[lK]=Radurspruenge(in)

laenge_VL=in(2,1);
laenge_HL=in(2,2);
laenge_VR=in(2,3);
laenge_HR=in(2,4);
X=in(2,5);
Y=in(2,6);
Z=in(2,7);
alpha1=in(2,8);
alpha2=in(2,9);
alpha3=in(2,10);
winkel=in(2,11);

% Bildung des Schwerpunktvektors:
global SP;
% Bezug der Fahrwerkspunkte in Ausgangslage:
InputKoord;
%   Übergabe der Eingabewerte in Unterfunktionen
%   Punkte werden so verschoben, das SP auf Urschprung liegt
%   Alle Punkte werden berechnet:
vl_1=solve_VL(laenge_VL,winkel);
hl_1=solve_HL(laenge_HL);
vr_1=solve_VR(laenge_VR,winkel);
hr_1=solve_HR(laenge_HR);
%   Koordinatentransformation-Rotation um alle 3 Achsen:
vl_2=KooTrans(vl_1,alpha1,alpha2,alpha3);
hl_2=KooTrans(hl_1,alpha1,alpha2,alpha3);
vr_2=KooTrans(vr_1,alpha1,alpha2,alpha3);
hr_2=KooTrans(hr_1,alpha1,alpha2,alpha3);
%   Zurückverschiebung der Punkte:
Z=SP+[X;Y;Z];

% Berechnung der enzelnen Punkte:
VL=zeros(3,16);
HL=zeros(3,16);
VR=zeros(3,16);
HR=zeros(3,16);

for i=1:16
    VL(:,i)=(vl_2(:,i))+Z;
    HL(:,i)=(hl_2(:,i))+Z;
    VR(:,i)=(vr_2(:,i))+Z;
    HR(:,i)=(hr_2(:,i))+Z;
end

% Berechnung der Radursprünge

lK=zeros(3,8);
```

```
lK(:,1)=VL(:,15)-VL(:,14); % ovl
lK(:,2)=VL(:,16)-VL(:,14); % pvl

lK(:,3)=HL(:,15)-HL(:,14); % ohl
lK(:,4)=HL(:,16)-HL(:,14); % phl

lK(:,5)=VR(:,15)-VR(:,14); % ovl
lK(:,6)=VR(:,16)-VR(:,14); % pvl

lK(:,7)=HR(:,15)-HR(:,14); % ovl
lK(:,8)=HR(:,16)-HR(:,14); % pvl
```

Radvektoren.m

```
% Hauptprogramm zur vollständigen Bestimmtheit des Fahrzeugs im Raum
% Input: Länge der Federn: l_VL, l_HL, l_VR, l_HR [mm] (Max 200 /Min 130)
%        Winkel Lenkeinschlag: winkel [°] (Max 120 /Min -120)
%        Translation des Rahmens in X, Y, Z
%        Rotationswinkel alpha1 (um x-Achse), alpha2 (um y), alpha3 (um
z)
% Output: 3 Radvektoren / Rad
% integrierte Programme: "solve_VL.m", "solve_HL.m", "solve_VR.m",
%                        "solve_HR.m", "Kootrans.m", "InputKoord.m"
function[r]=Radvektoren(in)

laenge_VL=in(2,1);
laenge_HL=in(2,2);
laenge_VR=in(2,3);
laenge_HR=in(2,4);
X=in(2,5);
Y=in(2,6);
Z=in(2,7);
alpha1=in(2,8);
alpha2=in(2,9);
alpha3=in(2,10);
winkel=in(2,11);

% Bildung des Schwerpunktvektors:
global SP;
% Bezug der Fahrwerkspunkte in Ausgangslage:
InputKoord;
%   Übergabe der Eingabewerte in Unterfunktionen
%   Punkte werden so verschoben, das SP auf Urschprung liegt
%   Alle Punkte werden berechnet:
vl_1=solve_VL(laenge_VL,winkel);
hl_1=solve_HL(laenge_HL);
vr_1=solve_VR(laenge_VR,winkel);
hr_1=solve_HR(laenge_HR);
%   Koordinatentransformation-Rotation um alle 3 Achsen:
vl_2=KooTrans(vl_1,alpha1,alpha2,alpha3);
hl_2=KooTrans(hl_1,alpha1,alpha2,alpha3);
vr_2=KooTrans(vr_1,alpha1,alpha2,alpha3);
hr_2=KooTrans(hr_1,alpha1,alpha2,alpha3);
%   Zurückverschiebung der Punkte:
Z=SP+[X;Y;Z];

% Berechnung der enzelnen Punkte:
VL=zeros(3,16);
```

```
HL=zeros(3,16);
VR=zeros(3,16);
HR=zeros(3,16);

for i=1:16
    VL(:,i)=(vl_2(:,i))+Z;
    HL(:,i)=(hl_2(:,i))+Z;
    VR(:,i)=(vr_2(:,i))+Z;
    HR(:,i)=(hr_2(:,i))+Z;
end

% Berechnung der globalen Ortsvektoren von den Radträgerpunkten/SP:
r1_vl=VL(:,3);
r2_vl=VL(:,6);
r3_vl=VL(:,8);
r4_vl=VL(:,14); % SP Rad
rvl=[r1_vl r2_vl r3_vl r4_vl];

r1_hl=HL(:,3);
r2_hl=HL(:,6);
r3_hl=HL(:,8);
r4_hl=HL(:,14); % SP Rad
rhl=[r1_hl r2_hl r3_hl r4_hl];

r1_vr=VR(:,3);
r2_vr=VR(:,6);
r3_vr=VR(:,8);
r4_vr=VR(:,14); % SP Rad
rvr=[r1_vr r2_vr r3_vr r4_vr];

r1_hr=HR(:,3);
r2_hr=HR(:,6);
r3_hr=HR(:,8);
r4_hr=HR(:,14); % SP Rad
rhr=[r1_hr r2_hr r3_hr r4_hr];

r=vertcat(rvl,rhl,rvr,rhr);
```

Rumpfursprung.m

```
% Hauptprogramm zur vollständigen Bestimmtheit des Fahrzeugs im Raum
% Input: Länge der Federn: l_VL, l_HL, l_VR, l_HR [mm] (Max 200 /Min 130)
%        Winkel Lenkeinschlag: winkel [°] (Max 120 /Min -120)
%        Translation des Rahmens in X, Y, Z
%        Rotationswinkel alpha1 (um x-Achse), alpha2 (um y), alpha3 (um
z)
% Output: 3 Radvektoren / Rad
% integrierte Programme: "solve_VL.m", "solve_HL.m", "solve_VR.m",
%                        "solve_HR.m", "Kootrans.m", "InputKoord.m"
function[r]=Rumpfursprung(in)

laenge_VL=in(2,1);
laenge_HL=in(2,2);
laenge_VR=in(2,3);
laenge_HR=in(2,4);
X=in(2,5);
Y=in(2,6);
Z=in(2,7);
alpha1=in(2,8);
```

```
alpha2=in(2,9);
alpha3=in(2,10);
winkel=in(2,11);

% Bildung des Schwerpunktvektors:
global SP;
% Bezug der Fahrwerkspunkte in Ausgangslage:
InputKoord;
%   Übergabe der Eingabewerte in Unterfunktionen
%   Punkte werden so verschoben, das SP auf Urschprung liegt
%   Alle Punkte werden berechnet:
vl_1=solve_VL(laenge_VL,winkel);
hl_1=solve_HL(laenge_HL);
vr_1=solve_VR(laenge_VR,winkel);
hr_1=solve_HR(laenge_HR);
%   Koordinatentransformation-Rotation um alle 3 Achsen:
vl_2=KooTrans(vl_1,alpha1,alpha2,alpha3);
hl_2=KooTrans(hl_1,alpha1,alpha2,alpha3);
vr_2=KooTrans(vr_1,alpha1,alpha2,alpha3);
hr_2=KooTrans(hr_1,alpha1,alpha2,alpha3);
%   Zurückverschiebung der Punkte:
Z=SP+[X;Y;Z];

% Berechnung der enzelnen Punkte:
VL=zeros(3,16);
HL=zeros(3,16);
VR=zeros(3,16);
HR=zeros(3,16);

for i=1:16
    VL(:,i)=(vl_2(:,i))+Z;
    HL(:,i)=(hl_2(:,i))+Z;
    VR(:,i)=(vr_2(:,i))+Z;
    HR(:,i)=(hr_2(:,i))+Z;
end

% Berechnung der Richtungsvektoren zur Bildung der Einheitsvektoren
ex=VL(:,1)-VL(:,2);
ey=VL(:,2)-VL(:,4);

r=horzcat(ex,ey);
```

VMax.m

```
% Programm bestimmt den größte Wert von | rd x vd | aus 3 Möglichkeiten
% eingegeben werden alle
% Der Rückgabevektor enthält die Nummer der Zeile des relavanten
% Geschwindigkeits-/zugehörigen Ortsvektors
% Zeilen 1 bis 4 stehen dabei für das jeweilige Rad
function[vm]=Vmax(rd,vd)

% Ausgabevektor wird erzeugt:
vm=zeros(4,1);

% Berechnung von | rd x vd | für jedes Rad um
vl=[norm(cross(vd(1:3,1),rd(1:3,1)));norm(cross(vd(1:3,2),rd(1:3,2)));...
    norm(cross(vd(1:3,3),rd(1:3,3)))];
hl=[norm(cross(vd(4:6,1),rd(4:6,1)));norm(cross(vd(4:6,2),rd(4:6,2)));...
    norm(cross(vd(4:6,3),rd(4:6,3)))];
vr=[norm(cross(vd(7:9,1),rd(7:9,1)));norm(cross(vd(7:9,2),rd(7:9,2)));...
    norm(cross(vd(7:9,3),rd(7:9,3)))];
```

```
hr=[norm(cross(vd(10:12,1),rd(10:12,1)));norm(cross(vd(10:12,2),...
    rd(10:12,2)));norm(cross(vd(10:12,3),rd(10:12,3)))];

% Einführung eines Zählvektors
a=[1;2;3];

% Zusammenfügen der Vektoren mit dem Zählvektor
vl=horzcat(a,vl);
hl=horzcat(a,hl);
vr=horzcat(a,vr);
hr=horzcat(a,hr);

% Sortieren der Zeilen nach der 2. Spalte
vl=sortrows(vl,2);
hl=sortrows(hl,2);
vr=sortrows(vr,2);
hr=sortrows(hr,2);

% Rückgabe der Komponente mit dem größten Element
vm(1)=vl(3,1);
vm(2)=hl(3,1);
vm(3)=vr(3,1);
vm(4)=hr(3,1);
```

w_Trans.m

```
function w =w_Trans(w)

% erzeugen der Radursprünge werden berechnet
global K
op=K(1:3,1:8);

% Einheitsvektoren an den lokalen Ursprüngen in globaler Form werden
% gebildet
ex1=op(:,1)/norm(op(:,1));
ex2=op(:,3)/norm(op(:,3));
ex3=op(:,5)/norm(op(:,5));
ex4=op(:,7)/norm(op(:,7));

ey1=op(:,2)/norm(op(:,2));
ey2=op(:,4)/norm(op(:,4));
ey3=op(:,6)/norm(op(:,6));
ey4=op(:,8)/norm(op(:,8));

ez1=cross(ex1,ey1)/norm(cross(ex1,ey1));
ez2=cross(ex2,ey2)/norm(cross(ex2,ey2));
ez3=cross(ex3,ey3)/norm(cross(ex3,ey3));
ez4=cross(ex4,ey4)/norm(cross(ex4,ey4));

e1=[ex1 ey1 ez1];
e2=[ex2 ey2 ez2];
e3=[ex3 ey3 ez3];
e4=[ex4 ey4 ez4];

e=eye(3);

% Bilden der Transformationsmatrizen
M1=[dot(e(:,1),e1(:,1)) dot(e(:,2),e1(:,1)) dot(e(:,3),e1(:,1));...
    dot(e(:,1),e1(:,2)) dot(e(:,2),e1(:,2)) dot(e(:,3),e1(:,2));...
```

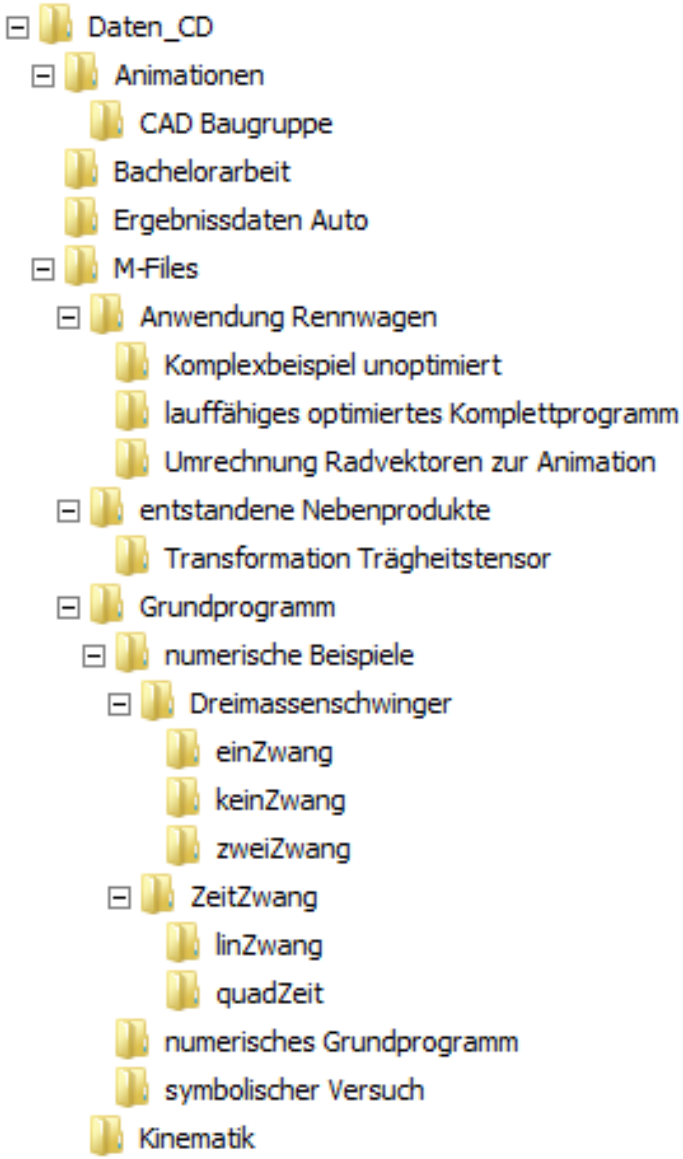
```
    dot(e(:,1),e1(:,3)) dot(e(:,2),e1(:,3)) dot(e(:,3),e1(:,3))];

M2=[dot(e(:,1),e2(:,1)) dot(e(:,2),e2(:,1)) dot(e(:,3),e2(:,1));...
    dot(e(:,1),e2(:,2)) dot(e(:,2),e2(:,2)) dot(e(:,3),e2(:,2));...
    dot(e(:,1),e2(:,3)) dot(e(:,2),e2(:,3)) dot(e(:,3),e2(:,3))];

M3=[dot(e(:,1),e3(:,1)) dot(e(:,2),e3(:,1)) dot(e(:,3),e3(:,1));...
    dot(e(:,1),e3(:,2)) dot(e(:,2),e3(:,2)) dot(e(:,3),e3(:,2));...
    dot(e(:,1),e3(:,3)) dot(e(:,2),e3(:,3)) dot(e(:,3),e3(:,3))];

M4=[dot(e(:,1),e4(:,1)) dot(e(:,2),e4(:,1)) dot(e(:,3),e4(:,1));...
    dot(e(:,1),e4(:,2)) dot(e(:,2),e4(:,2)) dot(e(:,3),e4(:,2));...
    dot(e(:,1),e4(:,3)) dot(e(:,2),e4(:,3)) dot(e(:,3),e4(:,3))];

% ausführen der Transformation
w1=M1*w(:,1);
w2=M2*w(:,2);
w3=M3*w(:,3);
w4=M4*w(:,4);

% Transformierte Ommega-Vektoren
w=horzcat(w1,w2,w3,w4);
```

WTrans.m

```
function Wt =WTrans(W)

% Berechnen der Einheitsvektoren des Rumpfes
global K;
ev=K(1:3,21:22);

% Anpassung der Einheitsvektoren
ex=ev(:,1)/norm(ev(:,1));
ey=ev(:,2)/norm(ev(:,2));
ez=cross(ex,ey)/norm(cross(ex,ey));

% Bilden der Einheitsmatrix
e=eye(3);

% Bilden der Einheitsvektormatrix
el=[ex ey ez];

% Bilden der Transformationsmatrizen
M=[dot(e(:,1),el(:,1)) dot(e(:,2),el(:,1)) dot(e(:,3),el(:,1));...
    dot(e(:,1),el(:,2)) dot(e(:,2),el(:,2)) dot(e(:,3),el(:,2));...
    dot(e(:,1),el(:,3)) dot(e(:,2),el(:,3)) dot(e(:,3),el(:,3))];

% Transformation des Ommega Vektors
Wt=M*W;
```

Anlagen, Teil 5

Inhalt der beiliegenden Daten CD:



Literatur

[Schl2007] Schlecht, Berthold: Maschinenelemente 1, München, Pearson Studium, 2007

[Piet2006] Pietruszka, Wolf Dieter: MATLAB und Simulink in der Ingenieurpraxis, Wiesbaden, B.T. Teubner Verlag, 2006

[Bei/Gro1997] Beitz, Wolfgang; Grote, Karl-Heinrich: DUBBEL - Taschenbuch für den Maschinenbau, Berlin, Springer- Verlag, 1997

[Zimm2010] Zimmermann, Martin: Vorlesung Maschinendynamik an der Hochschule Mittweida, Mittweida, 2010, Vorlesung 1/ Folie 4

[Wiki2011] http://de.wikipedia.org/wiki/Bewegungsgleichung, verfügbar am 05.08.2011, 15:00 Uhr

[Brunk2002] http://mechanik.tu-berlin.de/brunk/index.htm, verfügbar am 10.08.2011, 14:00 Uhr

[MATLAB®] MATLAB® Hilfe in der Version 7.10.0.499 (R2010a)

[Wiki2011] http://de.wikipedia.org/w/index.php?title=Datei:Afgeleide.svg&filetimestamp=20110410115333, verfügbar am 04.09.2011, 17:00 Uhr

[Zimm2011] Gespräche mit Professor Zimmermann während des Bachelor Projektes im Jahr 2011

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 13.09.2011

Andreas Petzold

Danksagung

Meine Bachelorarbeit möchte ich zum Anlass nehmen, mich zu bedanken,

bei Herrn Professor Martin Zimmermann, der mich während der Anfertigung meiner Bachelorarbeit begleitet und mich mit zahlreichen Hinweisen, Anregungen und vor allem Geduld unterstützt hat,

bei Herrn Professor Frank Weidermann, der sich bereit erklärt hat, meine Bachelorarbeit als Zweitprüfer zu bewerten,

bei Sascha Saralajew, der mir wichtige Tipps im Umgang mit MATLAB® mit auf den Weg gegeben hat,

bei Sandra Schubert, die zum Korrekturlesen zur Stelle war,

bei Sibylle Junghans, die mir das kurzfristige Binden der Arbeit ermöglicht hat,

bei Maria Philipp und unserer ungeborenen Tochter, die mir wertvolle moralische Stützen waren und mich immer motiviert haben,

und nicht zuletzt bei meinen Eltern, die mein Studium überhaupt erst ermöglicht haben.