Einfluss von nicht-iterativer Co-Simulation auf die numerische Lösung von Systemgleichungen

Stephanie Messner

Einfluss von nicht-iterativer Co-Simulation auf die numerische Lösung von Systemgleichungen

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Stephanie Messner

Institut für Regelungs- und Automatisierungstechnik
und
Kompetenzzentrum - Das Virtuelle Fahrzeug Forschungsgesellschaft mbH
A-8010 Graz

20.5.2014

Diese Arbeit ist in deutscher Sprache verfasst.

Begutachter:	Ao.Univ.-Prof. Dr. Anton Hofer
Betreuer im Unternehmen:	Dr. Martin Benedikt

An dieser Stelle möchte ich mich sehr herzlich bei Herrn Prof. Hofer bedanken, der mich während der Arbeit sehr unterstützte und stets ein offenes Ohr für meine Anliegen hatte.
Ein großes Dankeschön gilt auch Dr. Martin Benedikt und den Mitarbeitern des Kompetenzzentrums - Das virtuelle Fahrzeug mbh, die diese Arbeit überhaupt erst ermöglicht haben.
Ein großer Dank gilt meiner Familie und ganz besonders meiner Mutter, ohne die ich es nie soweit geschafft hätte.
Zu guter Letzt möchte ich mich bei allen Freunden und Studienkollegen bedanken, die meine Studienzeit unvergesslich machen.
Besonderer Dank gilt meinem Freund, der immer an mich glaubt und stets geduldig mit mir ist.

Auf zu neuen Ufern!

Abstract

The virtual development of complex technical systems is becoming increasingly important. In each application area, whether thermodynamics, mechanics, electrical engineering, etc., special simulation tools that have been optimized for a specific task. This saves time and allows the developer to make statements about the behavior of a model. Especially in the automotive industry it is important to couple these simulation tools to perform a full vehicle simulation. This can be implemented by an independent co-simulation platform.
The co-simulation platform is responsible for the data exchange between simulation tools and their underlying subsystems. By the non-iterative co-simulation and sequential coupling the subsystems are calculated in a defined order. At certain coupling times the generated data will be exchanged. Sometimes, due to a coupling loop, an extrapolation of the coupling signals is necessary. This extrapolation causes an estimation error, which depends on the order of the extrapolation.
In this work, the influence of discontinuities of the input signal that are caused by the zero-order-hold (ZOH) extrapolation on the numerical solution of the underlying system equations is considered. For this purpose, various numerical integration methods and their numerical errors are treated theoretically in the first part of the work. In the second part, the previously made statements are proven by different tests on special differential equations. Finally, experimental tests were carried out on a co-simulation example.

Kurzfassung

Die virtuelle Entwicklung dynamischer Systeme gewinnt zunehmend an Bedeutung. Für jedes Anwendungsgebiet, ob Thermodynamik, Mechanik, Elektrotechnik usw., nutzt ein bestimmtes Simulationswerkzeug, welches für eine Aufgabe optimiert wurden. Das spart Zeit und ermöglicht dem Entwickler, Aussagen über das Verhalten eines Modells zu treffen. Vor allem in der Automobilindustrie ist es wichtig, die verschiedenen Simulationswerkzeuge zu koppeln, um eine vollständige Fahrzeugsimulation durchführen zu können. Dies kann mit einem unabhängigen Co-Simulationsplattform implementiert werden.
Die Co-Simulationsplattform ist für den Datenaustausch zwischen den Simulationswerkzeugen und deren unterlagerten Teilsysteme verantwortlich. Durch die nicht-iterativen Co-Simulation und sequentielle Kopplung werden die Teilsysteme in einer definierten Reihenfolge berechnet und tauschen zu bestimmten Koppelzeitpunkten die generierten Daten (Koppelgrößen) aus. Durch Kopplungsschleifen ist eine Extrapolation der Kopplungsgrößen notwendig. Diese Extrapolation hat je nach Ordnung einen Schätzfehler der Koppelgrößen zur Folge.
In dieser Arbeit wird der Einfluss von Unstetigkeiten im Eingangssignal, die durch die Extrapolation nullter Ordnung (ZOH) entstehen, auf die numerische Lösung der unterlagerten Systemgleichungen behandelt. Zu diesem Zweck werden verschiedene numerische Integrationsverfahren und deren numerische Fehler im ersten Teil dieser Arbeit theoretisch behandelt. Im zweiten Abschnitt werden die vorher aufgestellten Aussagen mit verschiedenen Tests an definierten Differentialgleichungen verifiziert. Abschließend werden experimentelle Versuche an einem Co-Simulationsbeispiel durchgeführt und die Ergebnisse analysiert.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AWP	Anfangswertproblem
DAE	differenzial-algebraisches Gleichungssystem
DGL	Differenzialgleichung
EHK	Einheitskreis
ESV	Einschrittverfahren
EV	Eigenvektor
EW	Eigenwerte
ICOS	Independent Co-Simulation
LMSV	Lineares Mehrschrittverfahren
LTI	lineare, zeitinvariante Systeme
NB	Nebenbedingung
NEPCE	Nearly Energy-Preserving Coupling Element
NS	Nullstelle
ODE	gewöhnliche Differenzialgleichung
OM	OpenModelica
RL	Ruhelage
z.B.	zum Beispiel

Kapitel 1

Einleitung

In der Automobilindustrie wird je nach Anwendungsgebiet, z.B. Elektrotechnik, Thermodynamik, Funktionale Sicherheit usw., zur virtuellen Entwicklung ein bestimmtes Simulationswerkzeug verwendet, welches für eine bestimmte Aufgabe optimiert ist. Das spart Zeit und erlaubt dem Entwickler Aussagen bezüglich des Verhaltens eines Modells zu treffen. Das Ziel der Co-Simulation ist es, Modelle verschiedener Simulationswerkzeuge, welche in Wechselwirkung zueinander und mit der Fahrzeugumwelt stehen, zu koppeln, um z.B. eine Gesamtfahrzeugsimulation durchzuführen, siehe Abbildung 1.1 [3], [8].

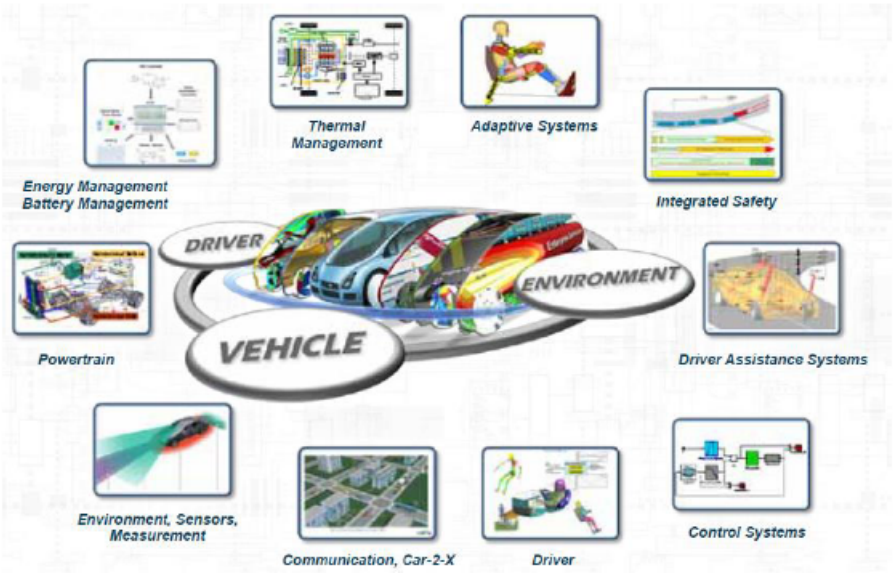


Abbildung 1.1: Verschiedene Fachbereiche und Simulationswerkzeuge, die durch Co-Simulation gekoppelt werden können [4]

Der Datenaustausch zwischen den Simulationswerkzeugen (Koppelgrößen) passiert zu bestimmten Koppelzeitpunkten. Innerhalb dieses zeitlichen Intervalles werden die numerischen Berechnungen mit den, in den Entwicklungsumgebungen integrierten, Lösungsalgorithmen unabhängig durchgeführt. Eine Synchronisation der Teilsysteme ist zwingend notwendig. Hier gibt es zwei

Methoden: Iterative und nicht-iterative Verfahren [3], [14].
Da die iterative Kopplung ein Rücksetzen der Teilsysteme erfordert, was softwaretechnisch oft aufwendig sein kann, wird als allgemeiner Zugang die nicht-iterative Kopplung in der Co-Simulation häufig eingesetzt. Hierbei wird jedes Teilsystem im Zeitintervall zwischen zwei Koppelzeitpunkten genau einmal simuliert. Dieser Ansatz erfordert eine Extrapolation der Eingangsgrößen, wenn sich durch die Kopplung der unterlagerten Teilmodelle eine interne Schleife ergibt [4].
Die Extrapolation verursacht einen Schätzfehler der Koppelgrößen. Die einfachste Extrapolation ist jene nullter Ordnung, welche der Funktionalität eines Abtast-Haltegliedes (ZOH) gleichzusetzen ist. In [3] beschäftigte man sich damit, den Effekt des Schätzfehlers, der durch die Extrapolation entsteht, zu reduzieren. Es wurden Methoden entwickelt, um diesen Fehler auszugleichen. Auf dieser Basis können Glättungsfilter eingesetzt werden, um einen glatten Signalverlauf zu erzeugen.
Durch die Extrapolation entstehen im Allgemeinen an den Koppelzeitpunkten hochfrequente Signalanteile. Diese Unstetigkeiten wirken sich nachteilig auf die numerische Lösung aus [2]. Jedoch ist nicht genau bekannt, in wie fern die Algorithmen und die berechnete Lösung beeinflusst werden.

Im Rahmen dieser Arbeit sollen die Effekte von Unstetigkeiten auf die numerische Lösung untersucht werden. Dabei ist von Interesse wie sich die Unstetigkeiten auf die verschiedenen Lösungsalgorithmen und auf die Systemklassen auswirken. Der Einfluss soll so weit wie möglich allgemein dargestellt werden und anschließend durch numerische Simulation gezeigt werden. Ziel ist eine Studie, die belegt, dass Unstetigkeiten in der Koppelgröße einen Einfluss auf die numerische Lösung von Systemgleichungen besitzen. Es sollte ersichtlich sein, für welche Kombination von Lösungsalgorithmus und Systemklasse Gegenmaßnahmen, wie zum Beispiel (z.B.) Glättung in der Kopplung, notwendig ist. Abschließend ist dies anhand einer Co-Simulation zu validieren.
Die theoretische Behandlung der Auswirkungen von Unstetigkeiten in der Eingangsgröße von Systemgleichungen auf die numerische Lösungsverfahren wird in Kapitel 4 behandelt. Auf Grundlage dieser Betrachtungen wird in Kapitel 5, mit Hilfe von Testsystemen, durch Simulation die Aussagen zu bestätigen und die Ergebnisse entsprechend zu bewerten. Schlussendlich werden in Kapitel 6 abschließende Untersuchungen durch Aufsetzen eines Co-Simulationsbeispieles durchgeführt.

Kapitel 2

Co-Simulation

Eine Co-Simulationsplattform soll es ermöglichen, verschiedene Teilmodelle, modelliert und simuliert in unterschiedlichen Simulationswerkzeugen, geeignet zu koppeln und dabei die komplexen Wechselwirkungen zwischen den Modellen zu berücksichtigen. Idealerweise soll durch Co-Simulation kein Einfluss auf den jeweiligen Lösungsalgorithmus und dessen Schrittweiten genommen werden. Ein Datenaustausch zur Synchronisation der Teilsysteme findet lediglich zu definierten Koppelzeitpunkten statt. In Abbildung 2.1 ist die Kopplung von verschiedenen Simulationswerkzeugen über eine Co-Simulationsplattform dargestellt [4].

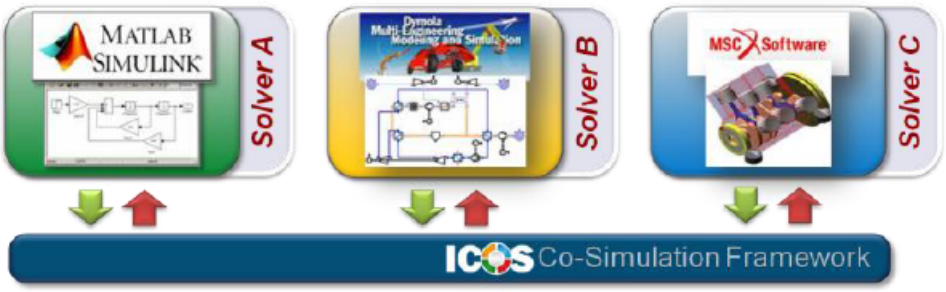


Abbildung 2.1: Kopplung von verschiedenen Simulationswerkzeugen über eine Co-Simulations-Plattform

Das *Kompetenzzentrum - Das virtuelle Fahrzeug Forschungsgesellschaft mbH* beschäftigt sich mit der Entwicklung einer unabhängigen Co-Simulationsplattform ICOS [10]. Diese Co-Simulationsplattform ermöglicht es, eine Reihe von Simulationswerkzeugen zu koppeln, wie z.B.: Abaqus, MSC Adams, AVL CRUISE /BOOST, ECS Kuli, Flowmaster, LS-Dyna, LabVIEW, Ricardo Wave, Phyton, Dymola/Modelica oder MATLAB. Somit wird mit ICOS eine Gesamtfahrzeugsimulation ermöglicht [4].

2.1 Kopplungsmethoden

Eine Kopplung der Teilsysteme entspricht einer Synchronisation der Teilsysteme. Diese Teilmodelle werden für bestimmte Zeitintervalle simuliert und zu bestimmten Zeitpunkten findet der Datenaustausch zur Synchronisation statt. Dieses Zeitintervall bezeichnet man als Makroschrittweite ΔT. Innerhalb dieses Intervalles werden die unterlagerten Teilsysteme unabhängig von den individuell festgelegten Lösungsalgorithmen und fester oder variabler Schrittweite eigenständig gelöst. Die vom Lösungsverfahren verwendete Schrittweite wird als Mikroschrittweite δT bezeichnet [3]. Eine Darstellung der Schrittweiten und ihre Bedeutung sind aus Grafik 2.2 zu entnehmen. Durch die Verwendung unterschiedlicher Mikro- so wie auch Makroschrittweiten spricht man im Allgemeinen von einer Multiratensimulation.

Abbildung 2.2: Datenaustausch mit ICOS

Bei der nicht-iterativen Co-Simulation werden Teilsysteme genau einmal über die Makroschrittweite simuliert. Zur zeitlichen Abfolge der Simulation stehen zwei Möglichkeiten zur Wahl: Die sequentielle und parallele Ausführungsreihenfolge. Bei der *sequentiellen* Ausführungsreihenfolge werden die Teilmodelle zeitlich hintereinander für jeden Makroschritt gerechnet und stellen dem nachfolgenden Teilsystem Daten zur Verfügung. Bei *paralleler* Ausführung werden die Modelle über ein Zeitintervall simultan gelöst.
Man betrachte zwei Teilsysteme die über ICOS zu den Zeitpunkten ΔT gekoppelt werden. Die berechneten Ausgänge bilden jeweils die Eingänge des anderen Teilmodelles (Koppelgrößen), somit ergibt sich eine geschlossene Schleife, siehe Grafik 2.3. Keines der beiden Modelle ist aufgrund der unbekannten Eingangsgrößen für sich lösbar.

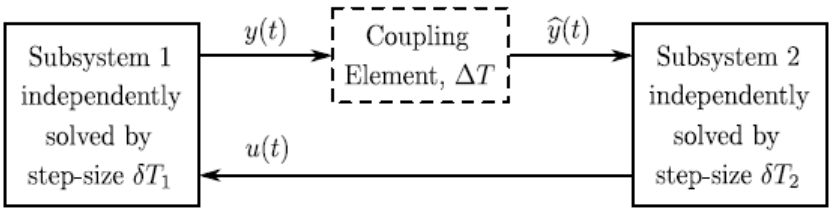


Abbildung 2.3: Schleife mit zwei Teilsystemen [5]

Am Ende jedes Zeitintervalles ΔT passiert ein Datenaustausch. Da zwischen den Koppelzeit-

punkten keine Informationen über die Koppelgrößen vorhanden sind, muss die unbekannte Koppelgröße extrapoliert bzw. geschätzt werden.

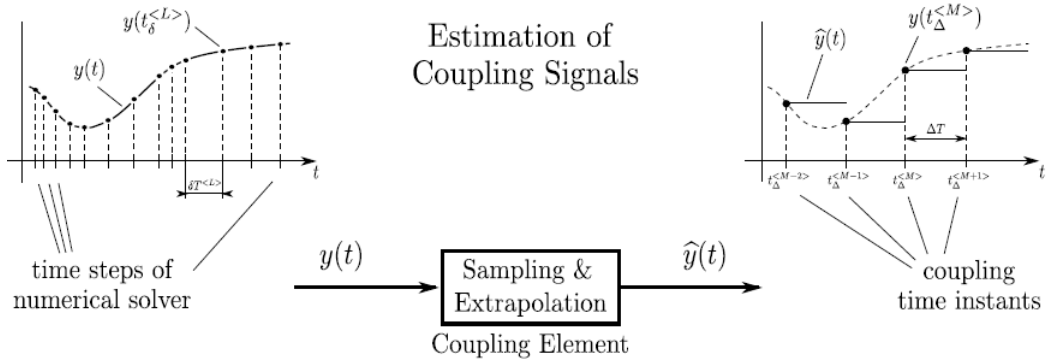


Abbildung 2.4: Extrapolation nullter Ordnung des Koppelsignales [5]

In ICOS gibt es verschiedene Extrapolationsmethoden: Extrapolation nullter Ordnung, erster Ordnung und zweiter Ordnung. Die Extrapolation nullter Ordnung tastet das Signal ab und hält diesen Wert für die Dauer der Makroschrittweite konstant. In dieser Arbeit wird die Methode nullter Ordnung betrachtet und ist in Abbildung 2.4 schematisch dargestellt. Betrachtet man die Schleife aus Abbildung 2.3 ist zu erkennen, dass durch eine sequentielle Kopplung nur eine Koppelgröße extrapoliert wird,wo hingegen die parallele Kopplung es erfordert, dass alle Koppelgrößen geschätzt werden. Bezüglich der Genauigkeit ist somit die sequentielle Methode günstiger [3], [4].
In Darstellung 2.4 sind im extrapolierten Verlauf der Koppelgröße auch die Unstetigkeiten zu den Koppelzeitpunkten zu erkennen. Diese können unter Umständen vorhandene schnelle Dynamiken im nachfolgenden Teilsystem anregen oder die numerische Genauigkeit, wie in dieser Arbeit belegt, beeinflussen.

Kapitel 3

Grundlagen

Um mit der Anaylse beginnen zu können, müssen einige Definitionen eingeführt werden. In diesem Kapitel werden die in dieser Arbeit betrachteten Systemklassen betrachtet. Diese Systemklassen können mit verschiedenen angeführten numerischen Integrationsverfahren gelöst werden, welche Eigenschaften besitzen, die in den folgenden Abschnitten erläutert werden, wie zum Beispiel Schrittweite, numerische Stabilität und Konvergenz.
Weiteres wurden zwei Fehlermaße eingeführt, um den Einfluss des Eingangssignales auf die Lösungsverfahren qualitativ bewerten zu können.

3.1 Systemklassen

3.1.1 ODE-System

Eine gewöhnliche Differentialgleichung m-ter Ordnung stellt eine Gleichung dar, deren zu suchende Variable m-mal nach der unabhängigen Variable abgeleitet wird. In der Systemtheorie wird die zu suchende Variable als eine Zeifunktion definiert [9]. Somit stellt der Zeitparameter t die unabhänige Variable dar. In der klassischen Regelungstechnik werden gewöhnliche Differentialgleichungen erster Ordnung betrachtet. Ein System von Differentialgleichungen kann in Form eines Zustandsraumodells $\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{u})$ dargestellt werden. Die zu suchenden Variablen $x_1 ... x_n$ werden als Zustandsvariablen und die Eingangsfunktionen $u_1 ... u_r$ als Systemeingänge bezeichnet. Dabei wird die Anzahl der Zustandsvariablen die Systemordnung genannt. In weiterer Folge wird die Systemordnung als Ordnung bezeichnet.
Folgend ist ein solches Zustandsraummodell oder ODE-System dargestellt.

$$\begin{aligned}\frac{dx_1}{dt} &= f_1(t, x_1, ...x_n, u_1, ...u_r)\\ &\vdots\\ \frac{dx_n}{dt} &= f_n(t, x_1, ...x_n, u_1, ...u_r)\end{aligned}$$

In Vektorschreibweise:

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

mit:

$$\begin{aligned}\mathbf{x} &= [x_1, x_2, \dots x_n]^T\\ \mathbf{f} &= [f_1, f_2, \dots f_n]^T\\ \mathbf{u} &= [u_1, u_2, \dots u_r]^T\end{aligned}$$

Mit folgenden Anfangswerten: $x_1(t_0) = x_{1,0}, \dots, x_n(t_0) = x_{n,0}$ und einem vorgegebenen Eingangsverlauf $\mathbf{u}(t)$ für $t_0 \leqslant t \leqslant t_e$.

3.1.2 DAE-System

Als DAE Systeme werden DGL Systeme mit algebraischen Nebenbedingungen bezeichnet. Dafür gibt es 2 verschiedene Darstellungsformen. Diese können wie folgt angegeben werden: [11]

- Implizites DAE-System

$$\mathbf{0} = \mathbf{f}(t, \dot{\mathbf{x}}, \mathbf{x}, \mathbf{z}, \mathbf{u}) \tag{3.1}$$

- Semi-explizites differenzial-algebraisches Gleichungssystem (DAE)-System

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \mathbf{z}, \mathbf{u}) &(3.2)\\ \mathbf{0} &= \mathbf{g}(t, \mathbf{x}, \mathbf{z}) &(3.3)\end{aligned}$$

Bezeichnungen:
$\mathbf{x}(t) \dots n$ Zustandsvariablen
$\mathbf{z}(t) \dots m$ algebraische Variablen
$\mathbf{u}(t) \dots r$ Eingänge

Differenzieller Index

> *“ Unter dem differenziellen Index versteht man die minimale Anzahl von Differenziationen, die auf die Gleichungen des DAE-Systems anzuwenden sind, um zu einem gewöhnlichen expliziten Differenzialgleichungssystem zu gelangen. ”*
>
> [[11]]

> *“ Der Index einer DAE ist ein Maß für die Anzahl von Differentiationen, die zur Lösung der DAE notwendig sind. ”*
>
> [[29]]

Der Index besagt, wie oft die Nebenbedingung 3.3, unter Einbeziehung von Gleichung 3.2, nach der Zeit t abgeleitet werden muss, um ein ODE-System zu erhalten.

Ein ODE-System hat den Index $i = 0$ und ein DAE-Systeme den Index $i > 0$. Ein physikalisches System, dessen mathematisches Modell durch z.B. den Lagrange Formalismus auf ein DAE-System gebracht wurde, besitzt meistens den Index $i = 3$.

Der differenzielle Index kann mit einer sogenannnten Indexprüfung ermittelt werden.

Um ein DAE-System, wie in Gleichung 3.2 und 3.3 dargestellt, mit Index $i > 0$ mit einem expliziten numerischen Integrationsverfahren lösen zu können, muss dieses DAE-System index-reduziert werden, damit schlussendlich ein System gewöhnlicher Differenzialgleichung (DGL) vorliegt. Bei $i = k$ bedeutet das laut [11] :

$$\frac{d^k}{dt^k}(\mathbf{g}(t, \mathbf{x}, \mathbf{z})) = 0$$

Die Anfangswerte $\mathbf{x}(0), \mathbf{z}(0)$ müssen durch das Gleichungssystem immer erfüllt sein.
In dieser Arbeit werden DAE-Systeme mit differenziellem Index $i = 1$ und einer algebraischen Variable $z(t)$ behandelt.

3.1.3 Steife Systeme

Ausgehend vom linearen Fall $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, bezeichnet [7] jene DGL-Systeme als steife Systeme, wenn die Systemmatrix $\mathbf{A}$ zwei Eigenwerte λ_{min} und λ_{max} mit folgender Eigenschaft besitzt:

$$\mid \frac{Re\{\lambda_{max}\}}{Re\{\lambda_{min}\}} \mid > 10^3 \tag{3.4}$$

3.2 Einführung numerische Lösungsverfahren

Zur numerischen Lösung gewöhnlicher DGL-Systeme können Ein- oder Mehrschrittverfahren, welche wiederum explizit oder implizit sein können, verwendet werden. In diesem Abschnitt werden die grundlegenden Begriffe erläutert.
Zur weiteren Betrachtung wird aus Gründen der Übersichtlichkeit eine einzige gewöhnliche DGL 1. Ordnung herangezogen

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

mit

$$x(t_0) = x_0.$$

Man möchte den Verlauf $x(t)$ in einem abgeschlossenen Intervall $[t_0, t_e]$ numerisch ermitteln. Es ist anzunehmen, dass $u(t)$ im zu untersuchenden Intervall $[t_0, t_e]$ bekannt ist und somit ist laut [11] eine weitere Vereinfachung möglich:

$$\dot{x} = f(t, x) \tag{3.5}$$

Laut [7] werden folgende Bezeichnungen eingeführt:

- Exakte Lösung: $x(t_i)$
- Numerische Approximation: x_i
- Exakte Ableitung: $f(t_i, x(t_i))$
- Numerische Approximation der Ableitung: $f(t_i, x_i)$

Des weiteren wird folgender Satz vorausgesetzt [7]:

„... Die Lösung x(t) der Differentialgleichung ... existiert in einem Gebiet G

$$G = \{(t, x) || t - t_0| \leq \alpha, |x_i - x_{i,0}| \leq \beta, i = 1, \ldots, n\}\, \alpha, \beta \ldots const.$$

und ist eindeutig, wenn gilt:

1. $f(t, x)$ *ist stetig und*
2. $f(t, x)$ *erfüllt für alle* q, v *die Bedingung,*

$$\|f(t, q) - f(t, v)\| \leq L\|q - v\|$$

mit einer (endlichen) Konstanten L (Lipschitz-Konstante).“

3.2.1 Schrittweite

Numerische Lösungsverfahren von DGL werden laut [28] auch Diskretisierungsverfahren genannt, da ein Zeitintervall von $[t_0, t_e]$ in Schritte t_i mit $i = 0, \ldots, N$ unterteilt wird, mit $N \in \mathbb{N}$. Der Abstand $h_i = |t_{i+1} - t_i|$ wird als Schrittweite bezeichnet.
Die Schrittweite h kann als konstant oder variabel gewählt werden [7]:

- konstantes h:

$$h = \frac{t_e - t_0}{N}$$
$$t_i = t_0 + i \cdot h$$

- variables h_i:

$$t_{i+1} = t_i + h_i$$

Man unterscheidet folgende Diskretisierungsverfahren:

3.2.2 Einschrittverfahren

Ein Einschrittverfahren (ESV) basiert auf einer Taylorreihenentwicklung um den Punkt t_i. Dabei werden t_i, x_i, h_i für die Berechnung von x_{i+1} verwendet:

$$x_{i+1} = x_i + h_i \Phi(t_i, x_i, h_i) \tag{3.6}$$

In dieser Vorschrift wird Φ Verfahrensfunktion genannt, sie ergibt sich aus der Taylorreihenentwicklung.

3.2.3 Lineare Mehrschrittverfahren

Ein Lineares Mehrschrittverfahren (LMSV) basiert auf einer Polynomapproximation, da eine bestimmte Anzahl von Werten aus der Vergangenheit, t_k, x_k mit $k = i + 1 - m, \ldots, i + 1$, für die Berechnung von x_{i+1} verwendet werden. Zur Vereinfachung der Darstellung wird eine konstante Schrittweite h gewählt.

$$\sum_{l=0}^{m} \alpha_l x_{i+l} = h \sum_{l=0}^{m} \beta_l f(t_{i+l}, x_{i+l}) \tag{3.7}$$

Aus dieser Darstellung lassen sich auch alle ESV ableiten.

3.2.4 Verfahrenstypen

Ist ein Verfahren unabhängig von den zukünftigen Werten $t_{i+1}, x_{i+1}, h_{i+1}$ so wird es **explizit** genannt. Werden die Werte $t_{i+1}, x_{i+1}, h_{i+1}$ benötigt, so ist das Verfahren **implizit**.

3.2.5 Fehler der numerischen Lösung

[7] unterscheidet den lokalen und globalen Fehler wie folgt:

- *lokaler Fehler*: Ist der Fehler der innerhalb eines Iterationsschrittes von x_i auf x_{i+1} entsteht, unter der Voraussetzung, dass die Iterationsschritte für x_i fehlerfrei durchgeführt wurden.
- *globaler Fehler*: Ist der akkumulierte Fehler für x_{i+1} aus den lokalen Fehlern.

Weiteres ergibt sich der *totale Fehler* ε_t aus:

$$\varepsilon_t(t_{i+1}) = \varepsilon_{d,global} + \varepsilon_r \tag{3.8}$$

Der Fehler ε_r ist der Rundungsfehler, der durch die begrenzte Zahlendarstellung im Computer auftritt. Die Variable $\varepsilon_{d,global}$ ist hier der *globale Diskretisierungsfehler* an der Stelle t_{i+1}, den [11] wie folgt berechnet:

$$\varepsilon_{d,global}(t_{i+1}) = x(t_{i+1}) - x_{i+1} \tag{3.9}$$

Der *lokale Diskretisierungsfehler* $\varepsilon_{d,lokal}(t_i)$ ergibt sich laut [11] aus der Differenz der exakten Lösung mit der Iterationsvorschrift, die ebenfalls mit der exakten Lösung berechnet wurde. Hier beispielhaft für das explizite Eulerverfahren:

$$\varepsilon_{d,lokal}(t_{i+1}) = \frac{x(t_{i+1}) - (x(t_i) + h \cdot f(t_i, x(t_i)))}{h} \tag{3.10}$$

Der Diskretisierungsfehler ist ein algorithmischer Fehler. Nimmt man an, dass es keinen Rundungsfehler gibt, so erkennt man, dass $\varepsilon_t = \varepsilon_{d,global}$.

3.3 Konsistenz, Konvergenz und numerische Stabilität

Für die Konsistenz von ESV und LMSV ist der lokale Diskretisierungsfehler $\varepsilon_{d,lokal}$ (in einem Schritt), für die Konvergenz der globale Diskretisierungsfehler $\varepsilon_{d,global}$ (in i Schritten) von Bedeutung. Auf diese Begriffe wird in diesem Abschnitt besonders eingegangen.
Die in diesem Kapitel verwendeten Definitionen wurden aus [13] entnommen.

3.3.1 Konsistenz

Ein **ESV** oder **LMSV** ist konsistent, wenn gilt:

$$\lim_{h \to 0} \varepsilon_{d,lokal}(t_i) = 0$$

Die Konsistenzordung $q \in \mathbb{N}$ ergibt sich mit:

$$\|\varepsilon_{d,lokal}(t_i)\| = \mathcal{O}(h^q)$$

Erfolgt die Abtastung eines numerischen Verfahrens mit unendlich kleiner Schrittweite, ist ein Algorithmus konsistent, wenn lokal die numerische Lösung der exakten Lösung entspricht. Somit ist z.B. für ein ESV $\varepsilon_{d,lokal}$ der vernachlässigte Rest der Taylorreihenentwicklung (4.2) dividiert durch die Schrittweite h. Das ESV besitzt Konsistenzordung q. In der Literatur wird der lokale Diskretisierungsfehler auch oft als *Abschneidefehler* bezeichnet.

3.3.2 Numerische Stabilität

Da ein ESV durch die allgemeine Form von LMSV aus Gleichung 3.7 dargestellt werden kann, beziehen sich weitere Aussagen zur Stabilität auf diese Form. Stabilität garantiert, dass die richtige Lösung angenähert wird. Die weiteren Aussagen wurden an [30] angelehnt.

Nullstabilität

Für die Definition der Nullstabilität wird folgende Testgleichung verwendet:

$$\dot{x} = 0, x(0) = 1$$

Diese Testgleichung wird in Formel (3.7) eingesetzt und somit die rechte Seite der Formel Null gesetzt. Die Nullstellen dieser Gleichung können wie folgt berechnet werden:

$$\rho(r) = \sum_{l=0}^{m} \alpha_l r^l \tag{3.11}$$

Somit nennt sich (3.11) charakteristisches Polynom oder eines von zwei erzeugenden Polynomen.

Ein LMSV heißt null-stabil, wenn keine Nullstelle (NS) des charakteristischen Polynoms einen Betrag größer als 1 hat, und wenn alle NS mit Betrag 1 einfach sind.[25]

Durch Analysieren der Nullstellen $r_i \in \mathbb{C}$ des charakteristischen Polynoms 3.11 erhält man folgende Stabilitätsaussagen:

- Das Verfahren ist nullstabil, wenn alle Nullstellen $|r_i| \leq 1$ und $|r_i| = 1$ einfach sind.
- Sind $m - 1$ Nullstellen $|r_i| < 1$, ist das Verfahren stark stabil.
- Es ist schwach stabil, wenn mehr als eine Nullstelle auf dem Einheitskreis (EHK) liegen.

Die Nullstabilität sagt demnach aus, was passiert wenn $h \to 0$.

Absolute Stabilität

Die Nullstabilität alleine garantiert nicht, dass die numerische Approximation für Schrittweiten $h > 0$ zur wahren Lösung konvergiert, die absolute Stabilität schon.

Für die Definition des Begriffes der absoluten Stabilität wird folgende Testgleichung verwendet:

$$\dot{x} = \lambda x, x(0) = x_0 \tag{3.12}$$

Diese wird in Formel (3.7) eingesetzt.

> *" Ein Algorithmus wird absolut stabil genannt, wenn die numerische Lösung des Testproblems abklingendes Verhalten besitzt, sofern auch die exakte Lösung abklingt. Als Gebiet der absoluten Stabilität ... wird jenes Gebiet in der komplexen $h\lambda$-Ebene bezeichnet, in welchem der Algorithmus absolut stabil ist. "*
>
> [[7]]

Somit ergibt sich folgendes zweites erzeugendes Polynom:

$$\sigma(r) = \sum_{l=0}^{m} (\alpha_l - h\beta_l\lambda)\, r^l \tag{3.13}$$

Mit $Re\{\lambda\} < 0$. Die Nullstellen des zweiten charakteristischen Polynoms definieren ein Gebiet der absoluten Stabilität mit einer fixen Schrittweite h, es wird eine komplexe $h\lambda$-Ebene aufgespannt. Das Verfahren ist absolut stabil für $h\lambda$, wenn alle Nullstellen von 3.13 $\mid r_i \mid < 1$ [15].
Für das explizite Eulerverfahren gilt folgendes Gebiet der absoluten Stabilität [12]:

$$|1 + h\lambda| < 1 \tag{3.14}$$

Somit gilt für die Schrittweite h:

$$h < |\frac{-2}{\lambda}| \tag{3.15}$$

Das ist in Abbildung 3.1 dargestellt.

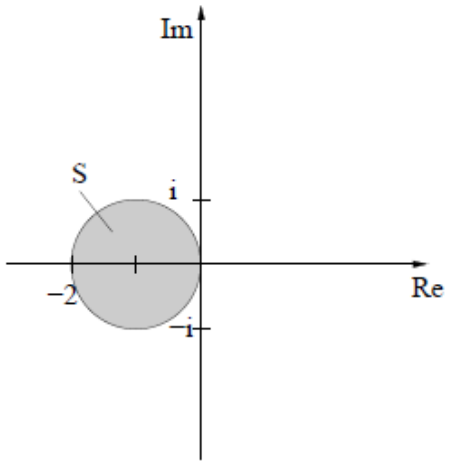


Abbildung 3.1: Gebiet der absoluten Stabilität explizites Eulerverfahren [28]

Das Gebiet der absoluten Stabilität für das implizite Eulerverfahren:

$$|\frac{1}{1-h\lambda}| < 1 \tag{3.16}$$

Das numerische Integrationsverfahren ist für $h > 0$ stabil. Somit beinhaltete das Stabilitätsgebiet die gesamte linke komplexe Halbebene, siehe Abbildung 3.2, dies nennt man auch $A - stabil$.[21]

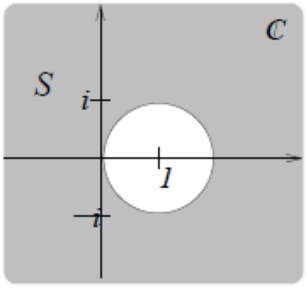


Abbildung 3.2: Gebiet der absoluten Stabilität implizites Eulerverfahren [28]

3.3.3 Konvergenz

Konvergenz liegt dann vor, wenn:

$$\lim_{i\to\infty} \varepsilon_{d,global}(t_i) = 0$$

Besitzt das ESV Konsistenzordung q, somit ist auch gültig:

$$\|\varepsilon_{d,global}(t_i)\| = \mathcal{O}(h^q)$$

Wird das numerische Verfahren unendlich fein diskretisiert und die numerische Lösung nähert sich an die exakte Lösung an, so spricht man von einem konvergenten Verfahren.
Man kann davon ausgehen, wenn ein numerisches Verfahren *konsistent* und *nullstabil* ist, dass es auch *konvergent* ist. Das ist in Abbildung 3.3 grafisch festgehalten.

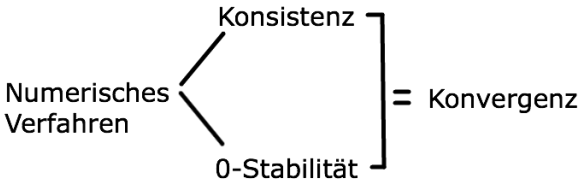


Abbildung 3.3: Zusammenhang Konsistenz, Stabilität und Konvergenz

3.4 Fehlermaße

Damit die numerischen Ergebnisse verschiedener Lösungsverfahren quantitativ vergleichbar sind, wurden zwei bestimmte Fehlermaße eingeführt, der durchschnittliche quadratische und der maximale Fehler. Je nach Verwendungszweck, Sichtweise oder System können verschiedene Fehlermaße verwendet werden.

Der durchschnittliche quadratische Fehler ist definiert durch:

$$E_{sqmean} = \frac{1}{N}\sum_{i=1}^{N}(w_i - f_i)^2 \tag{3.17}$$

Die Variable N bezeichnet die Anzahl der Abtast- oder Berechnungszeitpunkte, w_i ist der wahre Wert und f_i der fehlerhafte Wert zum i-ten Abtastzeitpunkt.
Der maximale Fehler ist durch die Supremumsnorm definiert:

$$E_{max} = \max_i |w_i - f_i| \tag{3.18}$$

Kapitel 4

Einschrittverfahren und lineare Mehrschrittverfahen

In Anlehnung an [21] wird die integrale Formulierung für ESV wie folgt dargestellt:

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(s, x(s))ds \tag{4.1}$$

Ausgehend von dieser Darstellung kann die Lösung x(t) des Anfangswertproblem (AWP) laut [7] durch eine Taylorreichenentwicklung um den Punkt t_i approximiert werden. Alle darauf basierenden Algorithmen nennt man ESV.

$$\begin{aligned} x(t_i + h) &= x(t_i) + x'(t_i)h + x''(t_i)\frac{h^2}{2!} + \cdots + x^{(k)}(t_i)\frac{h^k}{k!} + O(h^{k+1}) \\ x(t_i + h) &= x(t_i) + f(t_i, x_i)h + f'(t_i, x_i)\frac{h^2}{2!} + \cdots \\ &\quad + f^{(k-1)}(t_i, x_i)\frac{h^k}{k!} + \mathcal{O}(h^{k+1}) \end{aligned} \tag{4.2}$$

$\mathcal{O}(h^{k+1})$ ist der vernachlässigte Term der Ordnung $k + 1$.
Allgemein kann ein (explizites) Einschrittverfahren mit konstanter Schrittweite h laut [28] wie in Gleichung 3.6 dargestellt werden.
Auch für LMSV wird in [21] folgende Darstellung

$$x(t_{i+1}) = x(t_{i+1-\sigma}) + \int_{t_{i+1-\sigma}}^{t_{i+1}} f(s, x(s))ds \tag{4.3}$$

verwendet, wobei $\sigma \in \mathbb{N}, \sigma <= m$ die Anzahl der vorhergehenden Werte ist.
Die Formulierung für ein solches Vefahren ist in 3.7 dargestellt. Meist wird $\alpha_0 = 1$ gewählt. Falls $\beta_0 \neq 0$ ist das Verfahren implizit, anderenfalls explizit. Sind alle $\alpha_l = 0$, außer α_0, so entspricht das einem ESV. Erforderliche Startwerte werden für LMSV oft durch ESV berechnet.

4.1 Einschrittverfahren und die Auswirkungen des Einganges auf die numerische Lösung

Hier werden nur ein paar, vorwiegend einfache, explizite, sowie implizite Verfahren aufgelistet, die auch später noch behandelt werden. Beginnend mit dem einfachsten, dem Eulerverfahren. All diese Verfahren sind konsistent und konvergent. Für die Anwendung der Algorithmen ist wichtig, geeignete (konsistente) Anfangswerte zu verwenden.
Die Eingangsfunktion $u(t)$ wirkt auf die Systemgleichungen und somit auf die Diskretisierungsverfahren ein. Dies hat eine Abtastung des Einganges u_i zu diskreten Zeitpunkten t_i zur Folge $u_i = u(t_i)$.

4.1.1 Explizites Eulerverfahren

Nach dem zweiten Glied der Taylorreihenentwicklung 4.2 wird hier abgebrochen.

$$x_{i+1} = x_i + hf(t_i, x_i, u_i) \tag{4.4}$$

Auswirkungen des Fehlers

Der vernachlässigte Rest $\mathcal{O}(h^2)$ entspricht somit dem lokalen Diskretisierungsfehlers an der Stelle t_i. Der quadratische Anteil des vernachlässigten Terms fließt am meisten in $\varepsilon_{d,lokal}$ ein. Die Berechnung wurde anlehnend an [3] gezeigt:

$$\begin{aligned} x''(t_i)\frac{h^2}{2!} &= \left.\frac{d^2x(t)}{dt^2}\frac{h^2}{2!}\right|_{t=t_i} \\ &= \left.\frac{df(t,x)}{dt}\frac{h^2}{2!}\right|_{t=t_i} \\ &= \left.\left(\frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial t}\right)\frac{h^2}{2!}\right|_{t=t_i} \\ &= \bar{\varepsilon}_{d,lokal} \end{aligned} \tag{4.5}$$

Die Gleichung ist, wie in 3.2 erwähnt, auch vom Eingang $u(t)$ abhängig, deswegen wird u nach der Zeit t abgeleitet.
Wie man aus Gleichung 4.5 erkennt, wirkt sich die Ableitung des Einganges u nach der Zeit t auf den lokalen Diskretisierungsfehler aus. Bei starken Änderungen der Eingangfunktion $u(t)$ kann dieser einen großen Wert annehmen.
Hängt f nicht explizit von t ab, wird sich $\dfrac{\partial f}{\partial t}$ zu null ergeben.
Liegt das DGL-System der Form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}u$ vor, ist zu erkennen, dass sich durch $\dfrac{\partial f}{\partial u}\dfrac{du}{dt} = \mathbf{b}\dot{u}$ ergibt. Somit nimmt der Eingangsvektor $\mathbf{b}$ Einfluss auf den lokalen Diskretisierungsfehler.

4.1.2 Explizites Runge-Kuttaverfahren 2.Ordnung

Dieses Verfahren besitzt folgenden Algorithmus:

$$\begin{aligned} x_{i+1} &= x_i + h\left[a \cdot k_1 + b \cdot k_2\right] \\ k_1 &= f\left(t_i, x_i\right) \\ k_2 &= f\left(t_i + p \cdot h, x_i + p \cdot h \cdot k_1\right) \end{aligned} \tag{4.6}$$

[7] zeigt, dass dies einer Taylorreihenentwicklung gleichzusetzen ist, die nach dem dritten Glied abgebrochen wird. Folgend ergibt sich:

Auswirkungen des Fehlers

$$x(t_i + h) = x(t_i) + h\left[f(t_i, x(t_i)) + \frac{h}{2}\left(\frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial t}\right)\Big|_{t=t_i}\right] \tag{4.7}$$

Daraus ist zu erkennen, dass in die Berechnung für x_{i+1} schon $\dot{u}$ einfließt und somit ein genaueres Ergebnis zu erwarten ist.
Der lokale Fehler ist somit der vernachlässigte Rest $\mathcal{O}(h^3)$:

$$\begin{aligned} x'''(t_i)\frac{h^3}{3!} &= \frac{d^3x(t)}{dt^3}\frac{h^3}{3!}\Big|_{t=t_i} \\ &= \frac{d^2f(t,x)}{dt^2}\frac{h^3}{3!}\Big|_{t=t_i} \\ &= \frac{d}{dt}\left(\frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial t}\right)\frac{h^3}{3!}\Big|_{t=t_i} \\ &= \bar{\varepsilon}_{d,lokal}(t_i) \end{aligned}$$

Berechnet man die Ableitung nach der Zeit t, ergibt sich für:

$$\begin{aligned} \bar{\varepsilon}_{d,lokal}(t_{i+1}) &= \left(\frac{\partial^2 f}{\partial x^2}\dot{x}^2 + \frac{\partial f}{\partial x}\ddot{x} + \frac{\partial^2 f}{\partial x \partial u}\dot{x}\dot{u} + \frac{\partial f}{\partial x}\ddot{x}\right. \\ &+ \frac{\partial^2 f}{\partial u \partial x}\dot{u}\dot{x} + \frac{\partial f}{\partial u}\ddot{u} + \frac{\partial^2 f}{\partial u^2}\dot{u}^2 + \frac{\partial f}{\partial u}\ddot{u} \\ &+ \frac{\partial^2 f}{\partial x \partial t}\dot{x} + \frac{\partial f}{\partial x}\ddot{x} + \frac{\partial^2 f}{\partial u \partial t}\dot{u} + \frac{\partial f}{\partial u}\ddot{u} \\ &+ \left.\frac{\partial^2 f}{\partial t \partial x}\dot{x} + \frac{\partial^2 f}{\partial t \partial u}\dot{u} + \frac{\partial^2 f}{\partial t^2}\right)\frac{h^3}{3!}\Big|_{t=t_i} \end{aligned} \tag{4.8}$$

Es ist ersichtlich, dass $\dot{u}$ und $\ddot{u}$ in den Fehler eingehen.
Aufgrund des genaueren Algorithmus als bei Eulerverfahren ist ein kleinerer Diskretisierungsfehler zu erwarten.

4.1.3 Implizites Eulerverahren

$$x_{i+1} = x_i + hf(t_{i+1}, x_{i+1}) \tag{4.9}$$

Für die Lösung von x_{i+1} wird ein Verfahren zur Lösung nichtlinearer Gleichungen, wie z.B. das *Newton Verfahren* verwendet, wie in [33] beschrieben. Solche Verfahren nennt man iterative Nullstellensuche. In Abbildung 4.1 ist das Verfahren schematisch dargestellt. Das Verfahren berechnet für $\varphi(x^*) = 0$ die Nullstelle x^*. Diese Nullstellensuche nähert sich iterativ an die exakte Lösung x^* an.

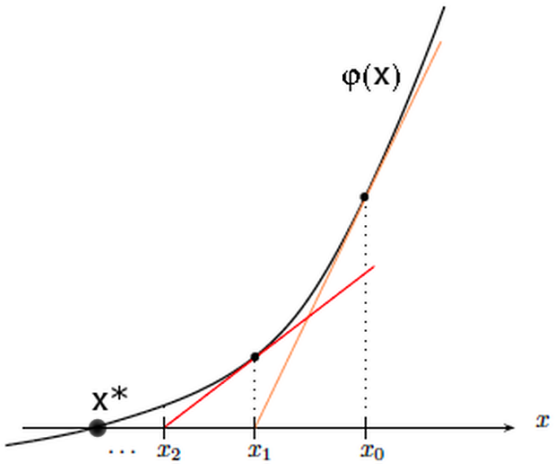


Abbildung 4.1: Darstellung Newton-Verfahren [33]

Die iterativ berechneten Lösungen x_k für Probleme der Form $\varphi(x_k) = 0$ werden für jeden Iterationsschritt folgend berechnet:

$$x_{k+1} = x_k - \frac{\varphi(x_k)}{\varphi'(x_k)} \tag{4.10}$$

Der Index k ist hierbei der Iterationsschritt. Die Funktion $\varphi(x_k)$ wird nach der zu iterierenden Variable x_k abgeleitet. Die Iteration wird so lange wiederholt, bis eine Fehlerschranke für $\mid x_{k+1} - x_k \mid$ unterschritten wird.
Angelehnt an [11], indem zur Lösung eines DAE System die Zustandsgleichungen und Nebenbedingung (NB) auf die Form $\varphi(x_k, z_k) = 0$ gebracht wurde, kann ein ODE System 2. Ordnung in folgende Form gebracht und somit von einem impliziten Lösungsverfahren numerisch berechnet werden:

$$\begin{aligned} \mathbf{p}_{i+1} &= (x_{1,i+1}, x_{2,i+1})^T \\ \boldsymbol{\varphi}(\mathbf{p}_{i+1}) &= (\varphi_1(x_{1,i+1}), \varphi_2(x_{2,i+1}))^T = \mathbf{0} \qquad (4.11) \\ \mathbf{p}_{i+1,k+1} &= \mathbf{p}_{i+1,k} - \mathbf{J}(\mathbf{p}_{i+1,k})^{-1} \boldsymbol{\varphi}(\mathbf{p}_{i+1,k}) \qquad (4.12) \end{aligned}$$

Für jeden Diskretisierungsschritt t_{i+1} wird das Gleichungssystem (4.12) gelöst.
Es wird solange iteriert bis $|\mathbf{p}_{i+1,k+1} - \mathbf{p}_{i+1,k}| \leq \varepsilon$.

Auswirkungen des Fehlers

Zu betrachten ist nun, in welchem Ausmaß die Eingangsfunktion $u(t)$ das Ergebnis einer iterativen Nullstellensuche beeinflusst.
Dazu wird die Jacobi-Matrix eingehend analysiert [11]:

$$\mathbf{J}(\mathbf{p}_{i+1,k}) = \begin{pmatrix} \frac{\partial \varphi_1}{\partial x_{1,i+1,k}} & \frac{\partial \varphi_2}{\partial x_{1,i+1,k}} \\ \frac{\partial \varphi_1}{\partial x_{2,i+1,k}} & \frac{\partial \varphi_2}{\partial x_{2,i+1,k}} \end{pmatrix} \tag{4.13}$$

Bei allen Lösungsverfahren wie dem Newton Verfahren, oder der Fixpunktiteration wird nach den Zustandsvariablen abgeleitet x_{i+1} und nicht nach der Zeit t. Somit ergibt sich, dass der Eingang und dessen Unstetigkeiten diese Verfahren nicht beeinflussen, da $\dot{u}$ nicht vorkommt. Der einzige Fehler ergibt sich hier aus der Ungenauigkeit ϵ.
Auch bei impliziten Verfahren gibt es einen Abschneidefehler, welcher gleich motiviert werden kann wie in Gleichung 4.2, nur entwickelt man hier um t_{i+1}, wie [31], [28] und [26]:

$$x(t_{i+1} - h) = x(t_{i+1}) - f(t_{i+1}, x_{i+1})h - \mathcal{O}(h^2) \tag{4.14}$$

Somit ergibt sich der Hauptteil des lokalen Diskretisierungsfehlers genau wie beim expliziten Eulerverfahren aus Gleichung 4.5 plus dem Fehler ε aus dem Newton Verfahren.

$$\bar{\varepsilon}_{d,lokal} = \left(\frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial t} \right) \frac{h^2}{2!}\bigg|_{t=t_{i+1}} + \varepsilon \tag{4.15}$$

Auch hier fließt $\dot{u}$ in den Fehler mit ein.

4.2 Lineare Mehrschrittverfahren und die Auswirkungen des Einganges auf die numerische Lösung

Es wird allgemein der lokale Diskretisierungsfehler von LMSV angegeben. Aus der allgemeinen Form lässt sich für jedes Verfahren der Form 3.7 der lokale Diskretisierungsfehler $\varepsilon_{d,lokal}$ herleiten.
Der lokale Diskretisierungsfehler für LMSV ergibt sich zu:

$$\varepsilon_{d,lokal} = \frac{1}{h}\left(\sum_{l=0}^{m} \alpha_l x(t_{i+l}) - h \sum_{l=0}^{m} \beta_l f(t_{i+l}, x(t_{i+l})) \right) \tag{4.16}$$

Unter Zuhilfenahme der Taylorreihenentwicklungen

$$\begin{aligned} x(t_i + lh) &= x(t_i) + lhx'(t_i) + \frac{(lh)^2}{2}x''(t_i) + \cdots + \frac{(lh)^q}{q!}x^{(q)}(t_i) + \mathcal{O}(h^{q+1}) \\ x'(t_i + lh) &= x'(t_i) + lhx''(t_i) + \frac{(lh)^2}{2}x'''(t_i) + \cdots + \frac{(lh)^{q-1}}{(q-1)!}x^{(q-1)}(t_i) + \mathcal{O}(h^q) \end{aligned}$$

, die in Gleichung 4.16 eingesetzt werden, kann der lokale Diskretisierungsfehler bestimmt werden. Als Beispiel dienen impliziter und expliziter Euler:

- $\alpha_0 = -1, \alpha_1 = 1, \beta_0 = 0$:

$$\bar{\varepsilon}_{d,lokal} = -\frac{h}{2} f'(t_i, x_i) \tag{4.17}$$

- $\alpha_0 = -1, \alpha_1 = 1, \beta_1 = 0$:

$$\bar{\varepsilon}_{d,lokal} = \frac{h}{2} f'(t_i, x_i) \tag{4.18}$$

4.3 Differential-algebraische Lösungsverfahren

Um ein differential-algebraisches Gleichungssystem zu lösen, muss einerseits numerisch integriert werden und andererseits die NB eingehalten werden. In [11] werden für Index-1-Systeme zwei Algorithmen vorgestellt, eine explizite und eine implizite Vorgehensweisen.

4.3.1 Explizites numerisches Integrationsverfahren

Ausgehend von einem expliziten Eulerverfahren zur Integration, sieht der Algorithmus wie folgt aus:

$$\begin{aligned} k = i: \quad 0 &= g(x_i, z_i) \Rightarrow z_i \\ k = i+1: \quad x_{i+1} &= x_i + f(x_i, z_i) \Rightarrow x_{i+1} \end{aligned} \tag{4.19}$$

Die Anfangswerte x_0 müssen gegeben sein. Es werden die algebraische Gleichung und die Differenzengleichung sequentiell gelöst. Dies fordert einen zusätzlichen algebraischen Gleichungslöser im Algorithmus. Es ist jedoch mit einem geringen Rechenaufwand verbunden.

4.3.2 Implizites numerisches Integrationsverfahren

Bei diesem Verfahren werden implizite numerische Integrationsverfahren, wie z.B. das implizite Eulerverfahren, angewendet. Somit können die Differenzengleichung und die algebraische Gleichung gleichzeitig gelöst werden.

$$\begin{aligned} x_{i+1} &= x_i + f(x_{i+1}, z_{i+1}) \\ 0 &= g(x_{i+1}, z_{i+1}) \end{aligned} \tag{4.20}$$

Zur Lösung wird wie in Abschnitt 4.1.3 eine iterative Nullstellensuche verwendet. Passt man das Verfahren für die neuen Verhältnisse an, erhält man das nichtlineare Gleichungssystem:

$$\begin{aligned} \mathbf{p}_{i+1} &= (x_{1,i+1}, x_{2,i+1}, z_{i+1})^T \\ \boldsymbol{\varphi}(\mathbf{p}_{i+1}) &= (\varphi_1(x_{1,i+1}), \varphi_2(x_{2,i+1}, \varphi_3(z_{i+1})))^T = \mathbf{0} \\ \mathbf{p}_{i+1,k+1} &= \mathbf{p}_{i+1,k} - \mathbf{J}(\mathbf{p}_{i+1,k})^{-1} \boldsymbol{\varphi}(\mathbf{p}_{i+1,k}) \end{aligned} \tag{4.21}$$

Dieses Verfahren ist mit mehr Rechenaufwand verbunden, es ist jedoch für größere Schrittweiten numerisch stabil.

4.3.3 Einfluss der Eingangsgröße auf den lokalen Diskretisierungsfehlers

In Kapitel 4.1 wurde gezeigt, dass sich der lokale Diskretisierungsfehler zwischen implizitem und explizitem numerischen Integrationsverfahren nur im Vorzeichen unterscheidet. Somit ist es ausreichend , für ein DAE System, das mit einem Eulerverfahren gelöst wird, den Restterm der Taylorreihenentwicklung nur einmal anzugeben. Ergänzend zu der in 4.1 gezeigten Herleitung ist, dass man nach einer weiteren zeitabhängigen Variable $z(t)$ ableiten muss.

$$\bar{\varepsilon}_{d,lokal} = \left(\frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial z}\frac{dz}{dt} + \frac{\partial f}{\partial t} \right) \frac{h^2}{2!}\bigg|_{t=t_{i+1}} + \varepsilon \tag{4.22}$$

Die Variable ε kennzeichnet den Fehler, der durch den algebraischen Gleichungslöser verursacht wird. Aus Formel (4.22) ist erkennbar, dass sich $\dot{u}(t)$ direkt auf den Fehler auswirkt. Indirekt könnte sich auch durch $\dot{z}(t)$ eine zeitliche Ableitung der Eingangsfunktion $u(t)$ ergeben, die sich zusätzlich auf den Fehler auswirkt. Dies wird in Abschnitt 5.2.1 näher behandelt.
Für das implizite Verfahren aus Gleichung (4.21) wird die Jacobi-Matrix analysiert:

$$\mathbf{J}(\mathbf{p}_{i+1,k}) = \begin{pmatrix} \frac{\partial \varphi_1}{\partial x_{1,i+1,k}} & \frac{\partial \varphi_2}{\partial x_{1,i+1,k}} & \frac{\partial \varphi_3}{\partial x_{1,i+1,k}} \\ \frac{\partial \varphi_1}{\partial x_{2,i+1,k}} & \frac{\partial \varphi_2}{\partial x_{2,i+1,k}} & \frac{\partial \varphi_3}{\partial x_{2,i+1,k}} \\ \frac{\partial \varphi_1}{\partial z_{i+1,k}} & \frac{\partial \varphi_2}{\partial z_{i+1,k}} & \frac{\partial \varphi_3}{\partial z_{i+1,k}} \end{pmatrix} \tag{4.23}$$

Auch hier ist erkennbar, dass die zeitliche Ableitung $\dot{u}(t)$ nicht in die iterative Nullstellensuche einfließt.

Kapitel 5

Vergleich von numerischen Lösungsverfahren anhand eines Testsystems

Im vorhergehenden Kapitel wurde die Auswirkung des Eingangssignales auf den lokalen Diskretisierungsfehlers für verschiedene Lösungsverfahren theoretische behandelt. Um diese Effekte auch praktische beweisen zu können, werden in diesem Kapitel verschiedene Testsysteme eingeführt. Diese Systeme sind jeweils zwei DGL 1. Ordnung. Folgende Testsysteme werden behandelt: Ein einfaches lineares, ein steifes und zwei DAE-Systeme. Bei einem DAE-System wirkt das Eingangssignal $u(t)$ auf die algebraische NB, beim anderen nicht. Diesen Testsystemen wird ein bestimmtes Eingangssignal $u(t)$ aufgeschaltet. Mit diesem Eingangssignal ist es möglich die Ruhelage der Systeme im gesamten Zustandsraum zu wählen und man kann somit einen beliebigen zeitlichen Verlauf der Zustandsgrößen (Trajektorienverlauf) vorgeben.
Um die eingeführten Fehlermaße anwenden zu können, werden die analytische Lösungen berechnet. Jeweils für das kontinuierliche und das abgetastete Eingangssignal. Es ist zu erwähnen, dass die DAE-Systeme so gewählt wurden, dass beide den selben Lösungsverlauf wie beim ODE-System besitzen.
An diesen Testsystemen und den beiden Eingangsignalen werden verschiedene Versuche in MATLAB durchgeführt. Es werden verschiedene numerische Integrationsverfahren mit fester und variabler Schrittweite angewendet und die Simulationsergebnisse mit den analytischen Lösungen verglichen und anschließend diskutiert.

5.1 Definition der Testsysteme

Für die erste Auswertung der Simulationsergebnisse wurden vier einfache Testsysteme 2. Ordnung gewählt: Ein lineares gewöhnliche Differenzialgleichung (ODE)-System, ein steifes System und zwei DAE-Systeme, eines weist $u(t)$ in der NB auf, das andere nicht. Jedes dieser linearen, zeitinvariante (LTI) Systeme wurde jeweils für ein kontinuierliches und ein abgetastetes Eingangssignal berechnet.

Weiteres wurde ein instabiles System definiert, welches mit Hilfe einer eingepassten Schaltgeraden und einem der oben genannten stabilen Systeme kombiniert wurde. Durch die Wahl einer geeigneten Eingangsfunktion $u(t)$ resultiert ein spezieller Verlauf der Zustandsgrößen. Trifft diese Trajektorie auf die Schaltgerade, so wird das instabile System wirksam und sie läuft ins Unendliche. Dies dient zur Veranschaulichung, wie sich die Eingangsfunktion $u(t)$ auf die Stabilität auswirkt.

5.1.1 Lineares ODE-System mit Schaltgerade

Das stabile lineare, zeitinvariante Systeme (LTI) System 2. Ordnung wurde wie folgt definiert:

$$\dot{\boldsymbol{x}} = \mathbf{A}_1\mathbf{x} + \mathbf{b}_1 u = \begin{pmatrix} -0.5 & 0.5 \\ 0.125 & -0.5 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} -0.5 \\ 1 \end{pmatrix} u =: \mathbf{f}_1(\boldsymbol{x}, u) \tag{5.1}$$

Der Eingang $u(t)$ wird beliebig vorgegeben. Dieses System weist folgende EV und stabile Eigenwerte (EW) auf:

$$\begin{aligned} s_1 &= -0.25 \\ s_2 &= -0.75 \\ \boldsymbol{v_1} &= \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} \\ \boldsymbol{v_2} &= \begin{pmatrix} -1 \\ 0.5 \end{pmatrix} \end{aligned} \tag{5.2}$$

In Abbildung 5.1 sind das Vektorfeld des Systems (5.1) und dessen EV für die RL $x_R = 0$ und $u_R = 0$ dargestellt.

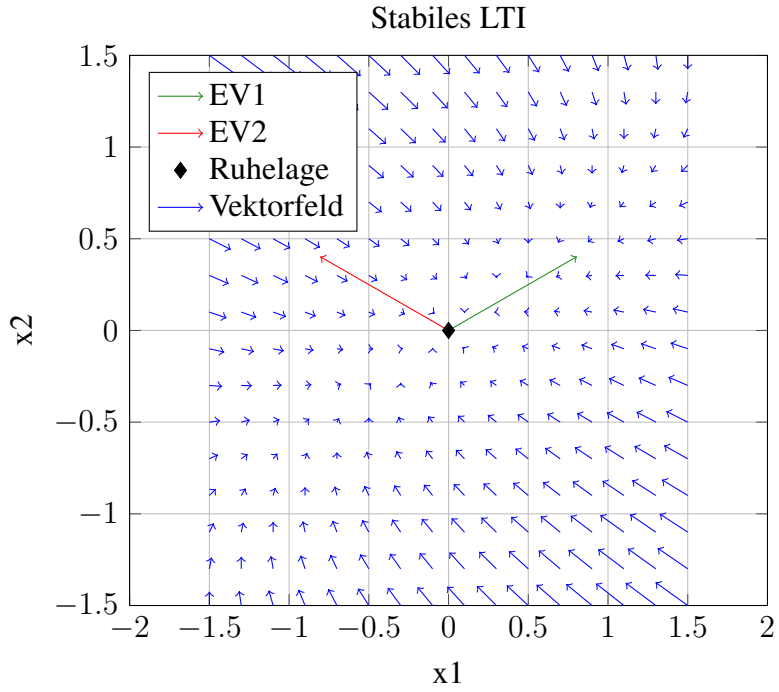


Abbildung 5.1: Stabiles System mit RL und EV

Für das kombinierte System wird folgende Systemgleichung für das instabile, von u unabhängiges, Teilsystem definiert:

$$\dot{\boldsymbol{x}} = \mathbf{A}_2\mathbf{x} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \boldsymbol{x} \tag{5.3}$$

Die zugehörigen EW und EV lauten:

$$\begin{aligned} s_1 = s_2 &= 2 \\ \boldsymbol{v_1} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \boldsymbol{v_2} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

In Abbildung 5.2 ist das Vektorfeld des Systems (5.3) und dessen EV mit zugehöriger RL dargestellt.

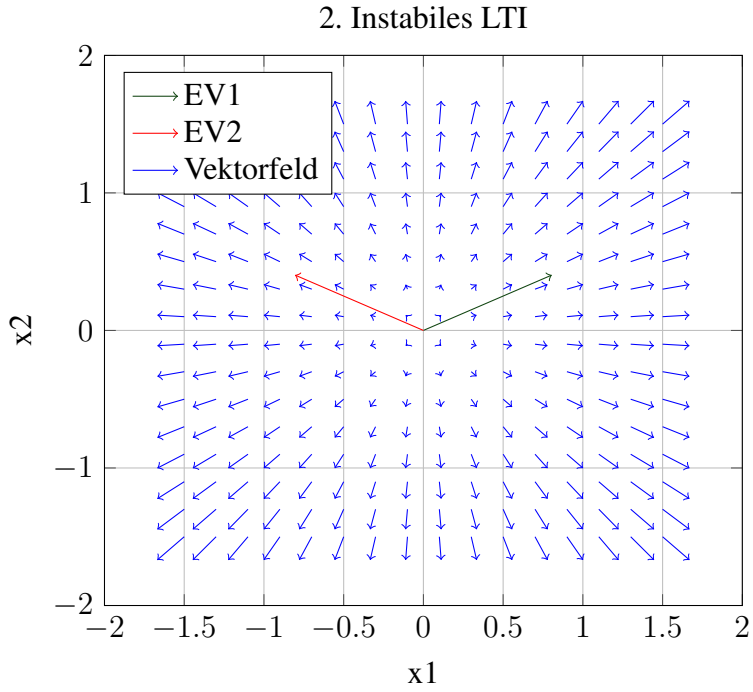


Abbildung 5.2: Instabiles System mit EV

Die Vereinigung von instabilem und stabilem System, welche durch eine Schaltgerade G getrennt werden, ergibt das kombinierte Testsystem das in weiterer Folge definiert wird:

$$G : x_2 = -10x_1 - 48 \tag{5.4}$$

Das resultierende System kann als Zustandraummodell folgend angeschrieben werden:

$$\dot{\mathbf{x}} = \begin{cases} \mathbf{A}_1\mathbf{x} + \mathbf{b}_1 u & \text{für } x_2 \geqslant -10x_1 - 48 \\ \mathbf{A}_2\mathbf{x} & \text{sonst} \end{cases} \tag{5.5}$$

Das finale System ist in Grafik 5.3 abgebildet.

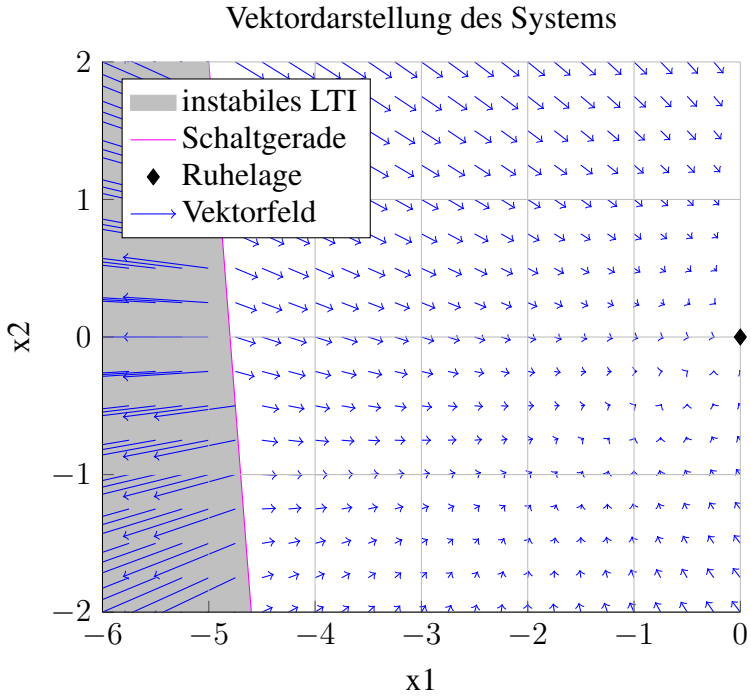


Abbildung 5.3: System mit Schaltgerade. Linksseitig: instabiles System, rechtsseitig: stabiles System

Wird G durch die Vorgabe einer beliebigen Ruhelage überschritten, wird ein anderes Teilsystem wirksam.

5.1.2 Steifes, lineares ODE-System mit Schaltgerade

Das Teilsystem für den steifen, stabilen Fall lautet:

$$\dot{\boldsymbol{x}} = \mathbf{A}_3\mathbf{x} + \mathbf{b}_1 u = \begin{pmatrix} -150.125 & 299.75 \\ 74.9375 & -150.125 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} -0.5 \\ 1 \end{pmatrix} u =: \mathbf{f}_2(\boldsymbol{x}, u) \tag{5.6}$$

Es besitzt folgende EW:

$$\begin{aligned} s_1 &= -0.25 \\ s_2 &= -300 \end{aligned} \tag{5.7}$$

Die EV entsprechen $\boldsymbol{v_1}$ und $\boldsymbol{v_2}$ aus Formel (5.2). Das Vektorfeld des Systems aus Gleichung (5.6) mit der RL für $x_R = 0$ und $u_R = 0$ ist in Abbildung 5.4 dargestellt.

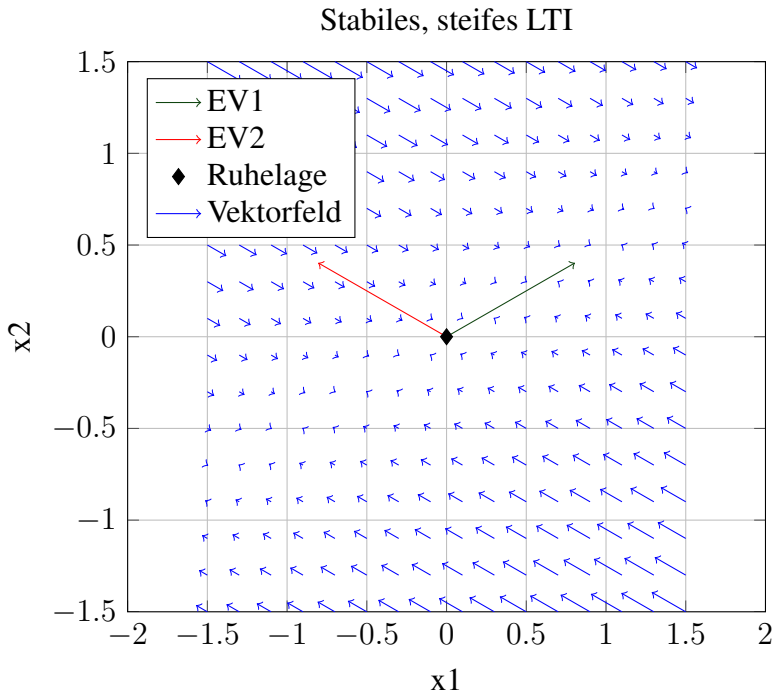


Abbildung 5.4: Stabiles, steifes System mit RL und EV

Kombiniert mit dem instabilen System 5.3 und der Schaltgeraden G (5.4) ergibt sich folgendes Testsystem:

$$\dot{\mathbf{x}} = \begin{cases} \mathbf{A}_3\mathbf{x} + \mathbf{b}_1 u & \text{für } x_2 \geqslant -10x_1 - 48 \\ \mathbf{A}_2\mathbf{x} & \text{sonst} \end{cases} \tag{5.8}$$

In Abbildung 5.5 ist das vereinigte System grafisch dargestellt.

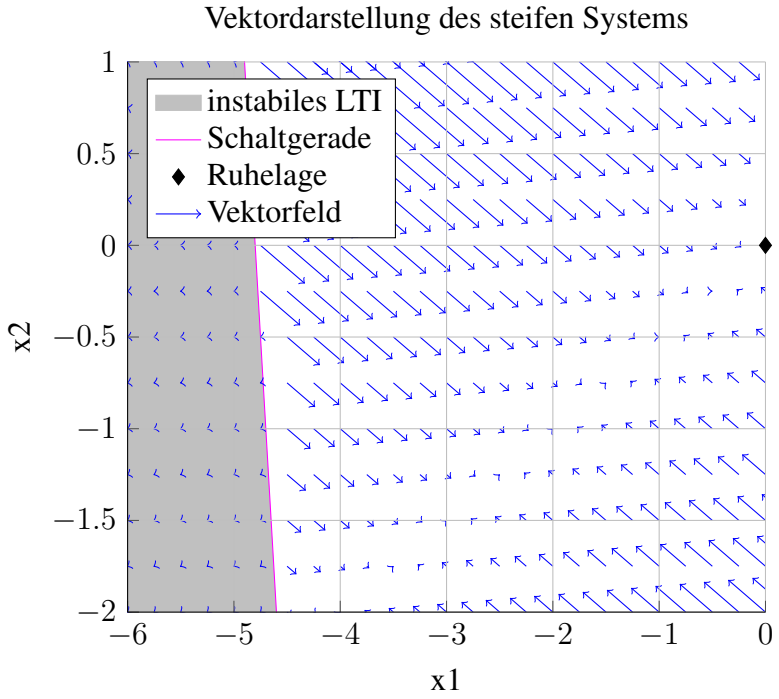


Abbildung 5.5: System mit Schaltgerade. Linksseitig: instabiles System, rechtsseitig: stabiles, steifes System

5.1.3 Lineares DAE System mit Schaltgerade

Für stabile DAE Systeme 2. Ordnung vom Index 1 wurden zwei Systeme gewählt. Folgendes stabiles DAE System weist $u(t)$ in der NB auf:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{A}_4\mathbf{x} + \mathbf{b}_2 u = \begin{pmatrix} -0.25 & 0.5 \\ 0.125 & -0.5 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} z + \begin{pmatrix} 1.5 \\ 2 \end{pmatrix} u := \boldsymbol{f}_3(\boldsymbol{x}, z, u) \\ 0 &= 0.25x_1 + 2u + z_1 =: g_1(\boldsymbol{x}, z) \end{aligned} \tag{5.9}$$

Das zweite stabile DAE System ist in der NB unabhängig von der Eingangsfunktion $u(t)$:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{A}_4\mathbf{x} + \mathbf{b}_1 u = \begin{pmatrix} -0.25 & 0.5 \\ 0.125 & -0.5 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} z + \begin{pmatrix} -0.5 \\ 1 \end{pmatrix} u := \boldsymbol{f}_4(\boldsymbol{x}, z, u) \\ 0 &= 0.25x_1 + z_1 =: g_2(\boldsymbol{x}, z) \end{aligned} \tag{5.10}$$

Kombiniert mit dem instabilen System 5.3 und der Schaltgeraden G (5.4) ergibt sich für das erste Testsystem:

$$\dot{\mathbf{x}} = \begin{cases} \mathbf{A}_4\mathbf{x} + \mathbf{b}_2 u & \text{für } x_2 \geqslant -10x_1 - 48 \\ \mathbf{A}_2\mathbf{x} & \text{sonst} \end{cases} \tag{5.11}$$

, sowie für das zweite Testsystem: Kombiniert mit dem instabilen System 5.3 und der Schaltgeraden G ((5.4)) ergibt sich folgendes Testsystem:

$$\dot{\mathbf{x}} = \begin{cases} \mathbf{A}_4\mathbf{x} + \mathbf{b}_2 u & \text{für } x_2 \geqslant -10x_1 - 48 \\ \mathbf{A}_2\mathbf{x} & \text{sonst} \end{cases} \tag{5.12}$$

Für beide Systeme gelten die selben EV und EW für das instabile und stabile System, sowie die Schaltgerade wie in Abschnitt 5.1.1. Somit ergibt sich schlussendlich der selbe Trajektorienverlauf bei selbem Eingang $u(t)$.

5.1.4 Analytische Lösungen

Für die Auswertung der Simulationsergebnisse ist es notwendig, die analytische Lösung der stabilen Testsysteme zu ermitteln. Hier spielt das Eingangssignal $u(t)$ eine wichtige Rolle.

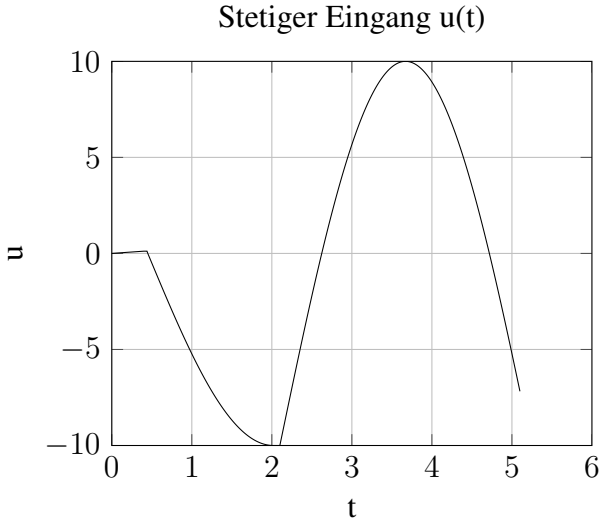


Abbildung 5.6: Stetiges Eingangssignal $u(t)$

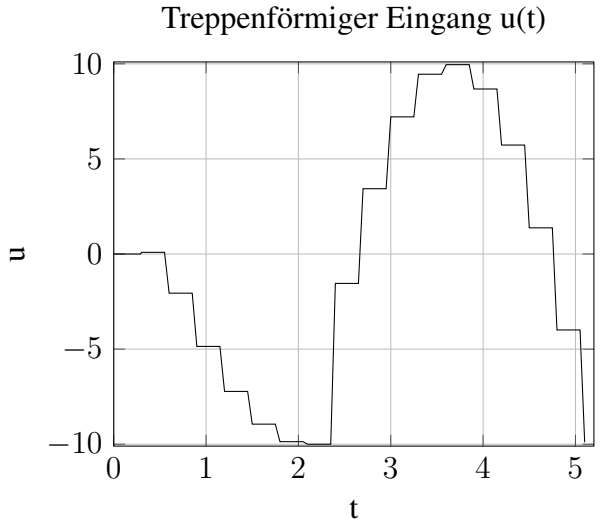


Abbildung 5.7: Abgetastetes Eingangsignal $u_{tr}(t)$

In Abbildung 5.6 ist das stetige Eingangssignal für das einfache LTI System und die DAE Systeme dargestellt. Dieses wurde alle $\Delta T = 0.3s$ abgetastet und ist in Grafik 5.7 abgebildet. Das stetige und abgetastete Eingangssignal für das steife LTI System ist in Abbildung 5.8 und 5.9 aufgezeichnet.

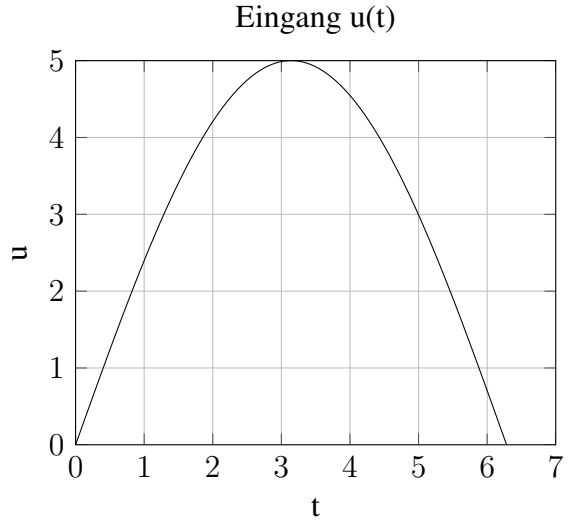


Abbildung 5.8: Stetiges Eingangssignal $u(t)$

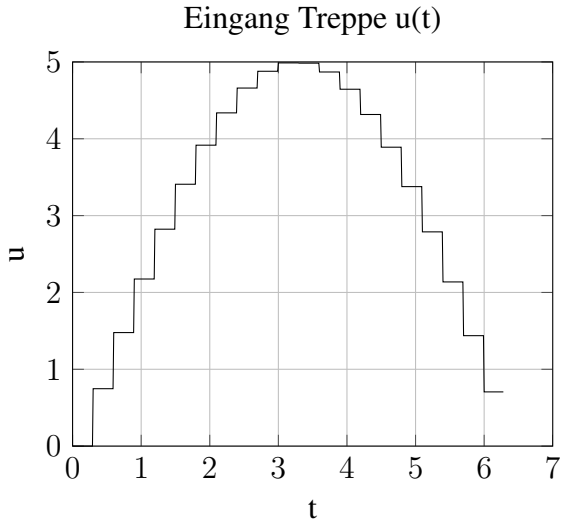


Abbildung 5.9: Abgetastetes Eingangsignal $u_{tr}(t)$

Das Integral für die Lösung eines LTI Systeme nach $\mathbf{x}(t)$ ist folgend aus [9] gegeben:

$$\mathbf{x}(t) = \boldsymbol{\phi}(t)\mathbf{x}_0 + \int_0^t \boldsymbol{\phi}(t-\tau)\mathbf{b}u(\tau)d\tau \tag{5.13}$$

Die Variable $\boldsymbol{\phi}(t)$ stellt die Transitionsmatrix, $\mathbf{x}_0$ die Startwerte und $\mathbf{b}$ den Eingangsvektor des LTI Systems dar.
Die Transitionsmatrix $\boldsymbol{\phi}(t)$ wurde zuerst im Laplace-Bereich berechnet und nachträglich in den Zeitbereich zurück transformiert.

$$\begin{aligned} \boldsymbol{\phi}(s) &= (s\mathbf{E} - \mathbf{A})^{-1} \\ \boldsymbol{\phi}(t) &\circ\!\!-\!\!\bullet\ \boldsymbol{\phi}(s) \end{aligned} \tag{5.14}$$

Da die Eingangsfunktionen, sowohl die stetige als auch treppenförmige, in mehrere Intervalle unterteilt ist, muss auch das Integral aufgeteilt werden.
Alle Berechnung wurde mit der Anwendung $Maxima$ [19] durchgeführt. Dies ist ein symbolisches Rechenprogramm.

Einfaches lineares-zeitinvariantes und differential-algebraisches Systeme

Für das System aus dem vorangegangen Abschnitt 5.1.1 errechnet sich folgende Transitionsmatrix $\boldsymbol{\phi}(t)$:

$$\boldsymbol{\phi}(t) = \begin{pmatrix} \frac{1}{2e^{t/4}} + \frac{1}{2e^{3t/4}} & \frac{1}{e^{t/4}} - \frac{1}{e^{3t/4}} \\ \frac{1}{4e^{t/4}} - \frac{1}{4e^{3t/4}} & \frac{1}{2e^{t/4}} + \frac{1}{2e^{3t/4}} \end{pmatrix} \tag{5.15}$$

Die stetige Eingangsfunktion $u(t)$ ist in 5 Intervalle unterteilt, wie Abbildung 5.6 zu entnehmen ist. Folgend sind die Intervalle aufgelistet:

$$\begin{aligned} u_0(0 \leq t < 0.4) &:= 0.3sin(t) \\ u_1(0.4 \leq t < 0.45) &:= 0.3sin(0.4) \\ u_2(0.45 \leq t < \frac{\pi}{2} + 0.45) &:= -10sin(t - 0.45) \\ u_3(\frac{\pi}{2} + 0.45 \leq t < 2.1) &:= -10 \\ u_4(2.1 \leq t < 5.1) &:= 20sin(t - 2.1) - 10 \end{aligned}$$

Somit ergibt sich das resultierende Integral mit den Anfangswerten $\mathbf{x}_0^T = \begin{pmatrix} 3 & -3 \end{pmatrix}$ zu:

$$\begin{aligned} \mathbf{x}(t) &= \boldsymbol{\phi}(t)\mathbf{x}_0 + \int_0^{0.4} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_0(\tau)d\tau + \int_{0.4}^{0.45} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_1(\tau)d\tau \\ &+ \int_{0.45}^{\frac{\pi}{2}+0.45} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_2(\tau)d\tau + \int_{\frac{\pi}{2}+0.45}^{2.1} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_3(\tau)d\tau \\ &+ \int_{2.1}^{5.1} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_4(\tau)d\tau \end{aligned} \tag{5.16}$$

Aufgrund des abgetasteten Eingangssignals $u_{tr}(t)$ muss das Integral für jeden Abtastzeitpunkt aufgeteilt werden. Dies sieht wie folgt aus:

$$\begin{aligned} \mathbf{x}_{tr}(t) &= \boldsymbol{\phi}(t)\mathbf{x}_0 + \int_0^{\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_0(0)d\tau + \int_{\Delta T}^{2\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_0(\Delta T)d\tau \\ &+ \int_{2\Delta T}^{3\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_2(2\Delta T)d\tau + \ldots + \int_{16\Delta T}^{17\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u_4(16\Delta T)d\tau \end{aligned} \tag{5.17}$$

Die DAE Systeme wie in Kapitel 5.1.3 definiert, ergeben für beide Eingangssignale $u(t)$ und $u_{tr}(t)$ die analytischen Lösungen aus Gleichung (5.16) und (5.17). Die Lösungstrajektorien für den glatten und abgetasteten Eingang ist in Grafik 5.10 dargestellt.

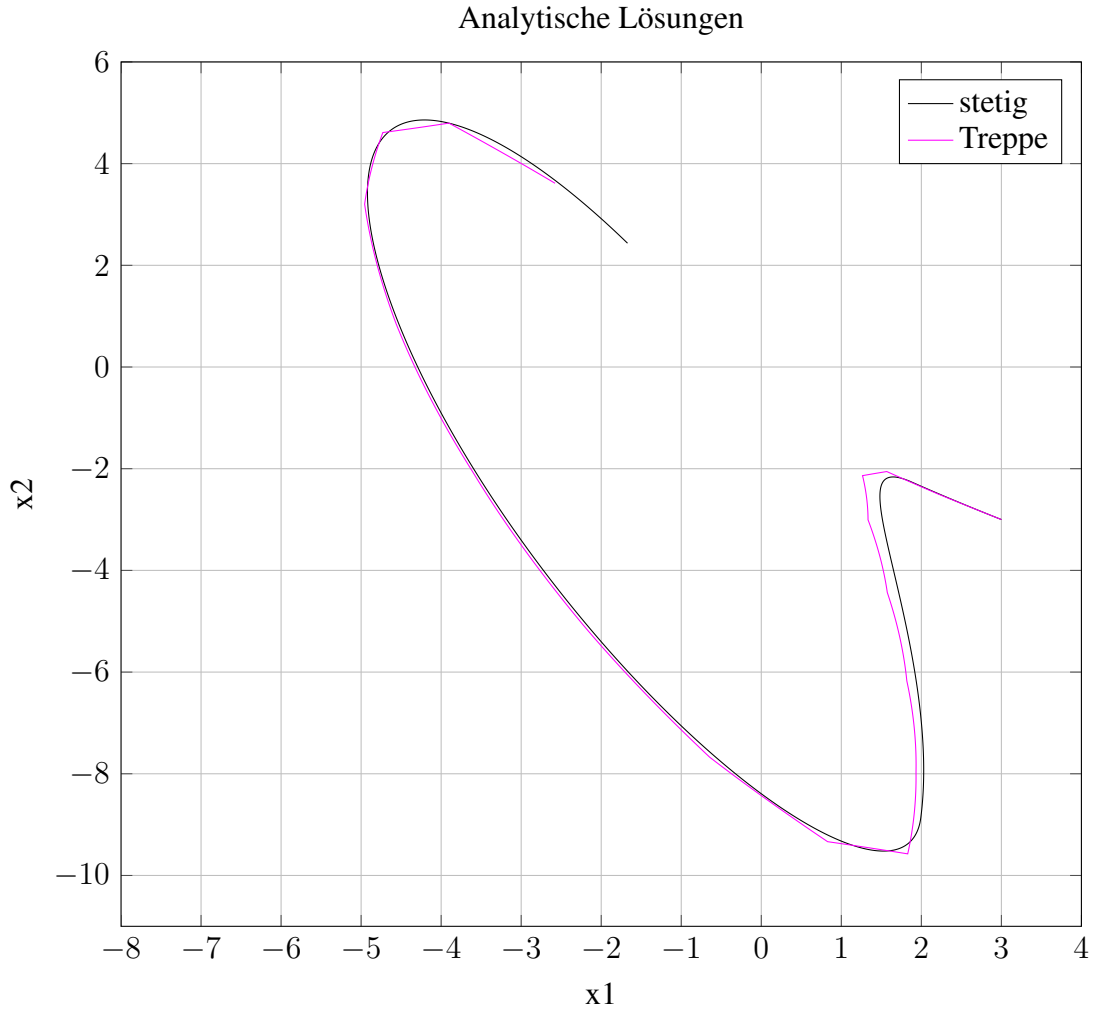


Abbildung 5.10: Trajektorien des ODE-Systems mit glattem und treppenförmigen Eingang

Steifes lineares-zeitinvariantes System

Für das steife System aus dem vorangegangen Abschnitt 3.1.3 errechnet sich folgende Transitionsmatrix $\boldsymbol{\phi}(t)$:

$$\boldsymbol{\phi}(t) = \begin{pmatrix} \frac{1}{2e^{t/4}} + \frac{1}{2e^{300t}} & \frac{1}{e^{t/4}} - \frac{1}{e^{300t}} \\ \frac{1}{4e^{t/4}} - \frac{1}{4e^{300t}} & \frac{1}{2e^{t/4}} + \frac{1}{2e^{300t}} \end{pmatrix} \tag{5.18}$$

Die stetige Eingangsfunktion ergibt sich zu $u(t) = 5sin(\frac{t}{2})$, siehe Abbildung 5.8. Somit kann $\mathbf{x}(t)$ mit den Anfangswerten $\mathbf{x}_0^T = \begin{pmatrix} 3 & -3 \end{pmatrix}$ gegeben werden:

$$\mathbf{x}(t) = \boldsymbol{\phi}(t)\mathbf{x}_0 + \int_0^{\frac{\pi}{2}} \boldsymbol{\phi}(t-\tau)\mathbf{b}u(\tau)d\tau \tag{5.19}$$

Die Lösung $\mathbf{x}(t)$ für das abgetastete Eingangssignal $u_{tr}(t)$ wird wie in Formel (5.20) berechnet:

$$\begin{aligned}\mathbf{x}_{tr}(t) &= \boldsymbol{\phi}(t)\mathbf{x}_0 + \int_0^{\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u(0)d\tau + \int_{\Delta T}^{2\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u(\Delta T)d\tau + \dots \\ &+ \int_{19\Delta T}^{20\Delta T} \boldsymbol{\phi}(t-\tau)\mathbf{b}u(19\Delta T)d\tau \end{aligned} \tag{5.20}$$

Die Lösungstrajektorien für den glatten und angetasteten Eingang ist in Grafik 5.11 und genauer in Grafik 5.12 dargestellt, um den Unterschied besser erkennen zu können.

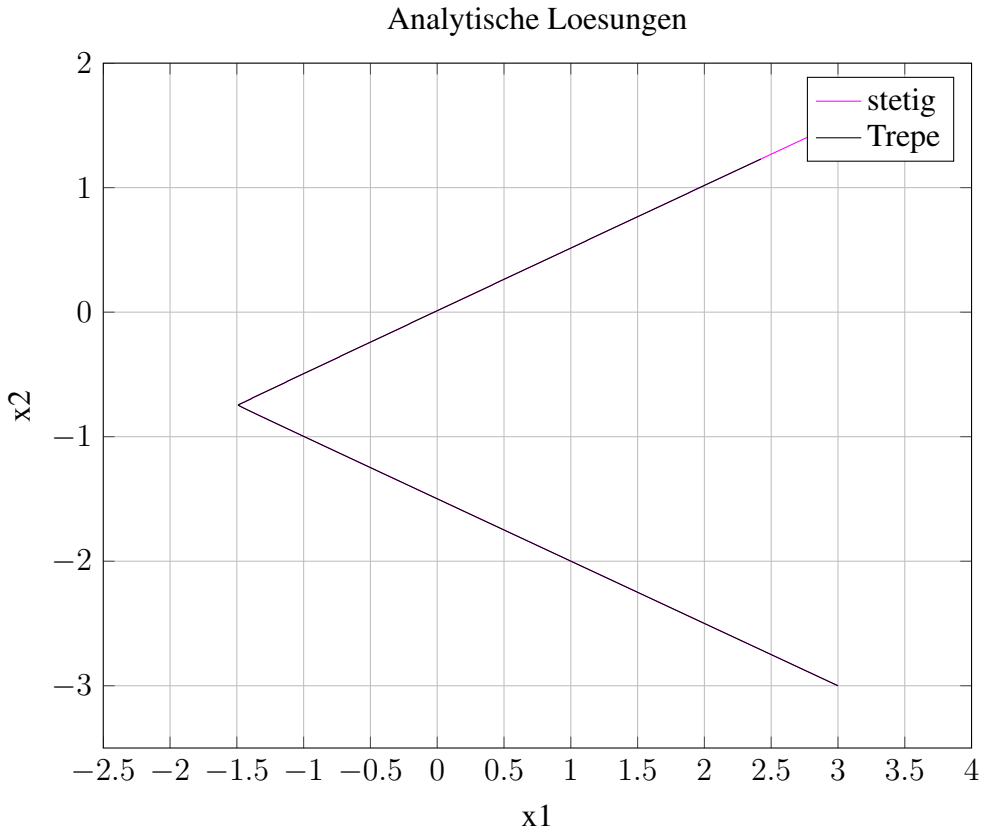


Abbildung 5.11: Trajektorien des steifen ODE-Systems mit glattem und treppenförmigen Eingang

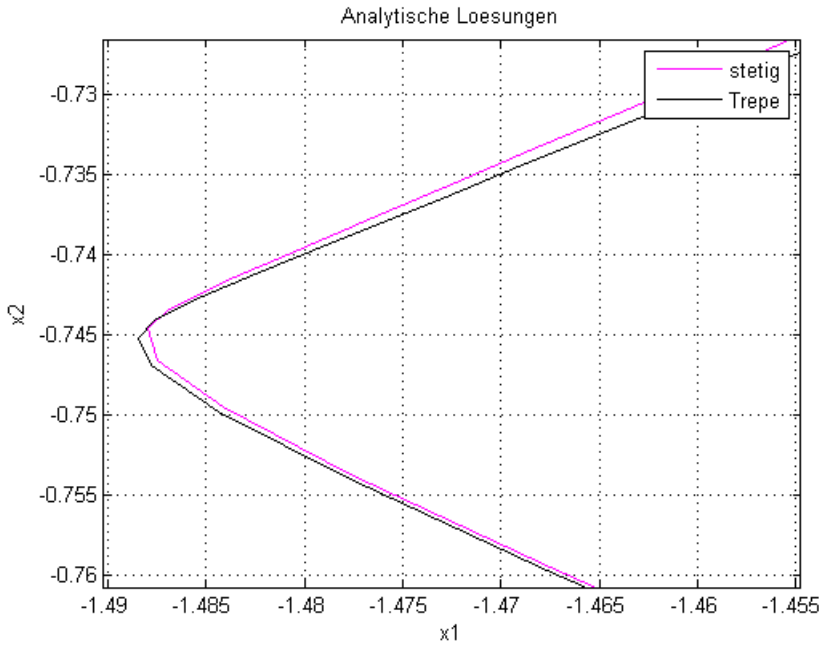


Abbildung 5.12: Trajektorien des steifen ODE-Systems, vergrößerte Darstellung

5.2 Auswirkungen einer treppenförmigen Eingangsfunktion auf das numerische Ergebnis in MATLAB

Für die verschiedenen Testfälle wurde den Systemen unterschiedliche Eingangsfunktionen aufgeschaltet: eine kontinuierliche (stetige) Kurve und deren treppenförmige (unstetige) Eingangsfunktion, wie in Grafik 5.6 und Grafik 5.7 abgebildet.
Aus den Herleitungen der numerischen Fehler aus Kapitel 4 ist erkennbar, dass sich Unstetigkeiten in der Eingangsfunktion negativ auf die numerische Lösung auswirken. Dies ist aufgrund der Ableitung der Eingangsgröße nach der Zeit im lokalen Diskretisierungsfehler erklärbar, da im Bereich der Unstetigkeiten die Steigung gegen unendlich strebt.
Durch Versuche an den Testsystemen aus Abschnitt 5 mit einfachen numerischen Lösungsverfahren wird dieser Effekt dargestellt und mit der analytischen Lösung für die glatte und abgetastete Eingangsfunktion aus Abschnitt 5.1.4 verglichen. Für den Vergleich war die Einführung von geeigneten Fehlermaßen wichtig. Zwei Fehlerkriterien sind hierbei von Vorteil. Der maximal Fehler E_{max}, der die maximale Abweichung von der idealen Lösung angibt und der durchschnittlich quadratische Fehler E_{sqmean}, der den durchschnittlichen Fehler zu allen Berechnungszeitpunkten angibt.
In Abbildung 5.13 sind die Ergebnisse der numerischen Integration mit dem explizten Eulerverfahren mit fester Schrittweite des ODE Testsystems mit kontinuierlichem Eingangssignal dargestellt.

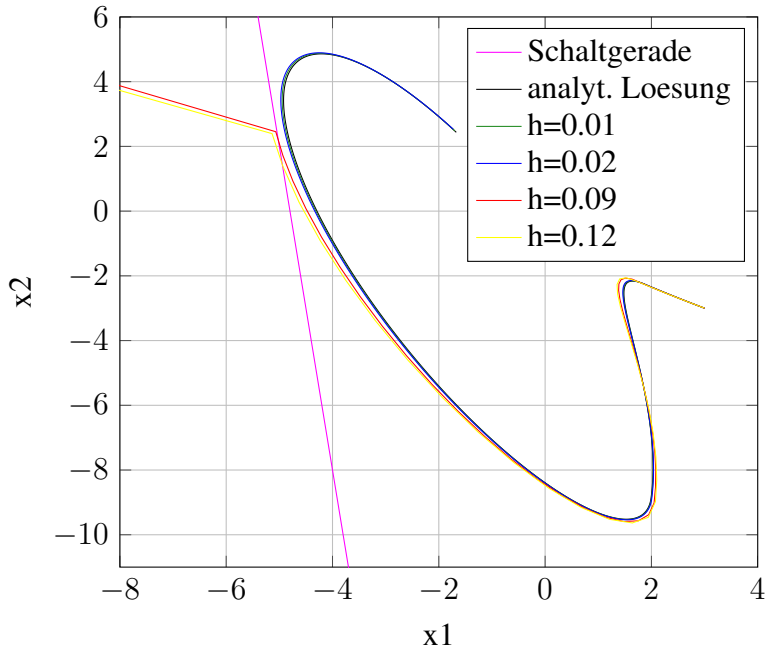


Abbildung 5.13: Ergebnisse der numerischen Integration des ODE-Systems mit expliziten Eulerverfahren und kontinuierlichen Eingang $u(t)$

Für kleine Schrittweiten h folgt die numerische Lösung der analytischen Lösungen gut. Je größer h wird, desto ungenauer kann die numerische Integration werden. Dies erkennt man für $h = 0.09$ und $h = 0.12$. Wählt man h zu groß, weicht die Lösungstrajektorie so weit von der idealen Lösung ab, dass sie in das instabile System läuft und gegen unendlich strebt.

In Abbildung 5.14 sind die Ergebnisse mit dem abgetasteten Eingangssignal dargestellt.

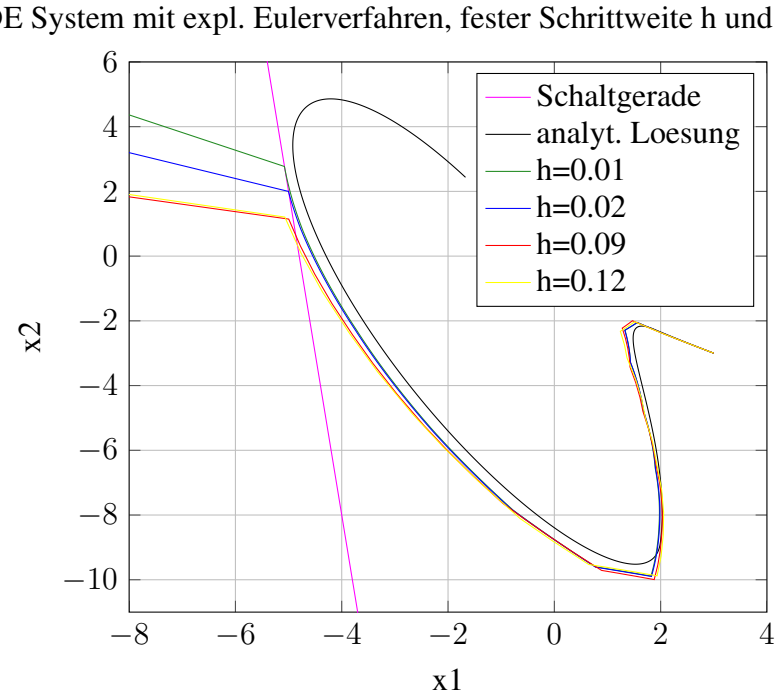


Abbildung 5.14: Ergebnisse der numerischen Integration des ODE-Systems mit expliziten Eulerverfahren und Treppen-Eingang $u(t)$

Hier bestätigen sich die vorangegangenen Überlegungen bezüglich des numerischen Fehlers bei Unstetigkeiten in der Eingangsgröße. Somit kann man einen Unterschied zwischen Grafik 5.13 und 5.14 deutlich erkennen. Durch das abgetastet Eingangssignal weichen die Lösungen, die mit verschiedenen Schrittweiten h berechnet wurden, weiter von der analytisch Lösungen ab, sodass keine numerische Lösung konvergiert.
Weiteres wird in diesem Kapitel der Fehler, der durch treppenförmige Eingangsfunktionen verursacht wird, an den verschiedenen Systemen genauer dargestellten.
Schlussendlich werden diese Überlegungen durch Versuche an den verschiedenen Testsystemen mit verschiedenen in Matlab implementierten Lösern gezeigt. Dabei ist zu erwähnen, dass die Löser mit einer Schrittweitensteuerung ausgestattet sind. Es wird der Fehler geschätzt und darauf die Schrittweite angepasst. Treten in den DGL schnelle Änderungen oder Unstetigkeiten in den Gleichungen auf, wird die Schrittweite verkleinert. Das wird auch in [1] behandelt. Das bedeutet, dass sich die Anzahl der Schritte für ein System mit treppenförmigen Eingangssignal deutlich höher sein muss, als bei einem System mit kontinuierlichem Eingangssignal.

Aus Testzwecken wurden spezielle Lösungsverfahren zusätzlich in Matlab implementiert. Dies sind bestehende Verfahren bezogen von [17]. Mit fester und variabler Schrittweite wurden das implizite und explizite Eulerverfahren für ODE und DAE-Systeme ergänzt. Die expliziten Runge-Kutta-Verfahren dritter ($ode3$) und fünfter ($ode5$) Ordnung wurden mit fester Schrittweite eingefügt. Hierbei ist Vorsicht bei der Wahl der Schrittweite geboten. Sie darf nicht zu groß gewählt werden, damit das Verfahren numerisch stabil bleibt.

5.2.1 Theoretische Fehlerauswertung mit einer sprunghaften Eingangsgröße

In den vorangegangen Kapiteln 4.1 und 4.2 wurde der lokale Diskretisierungsfehler für verschiedene Lösungsverfahren und Systeme mathematisch dargestellt. Daraus wurde ersichtlich, dass durch Unstetigkeiten in der Eingangsgröße sich ein größerer numerischer Fehler ergibt. Die Überlegungen zum lokalen Diskretisierungsfehler werden hier visualisiert. Man bediente sich hier an dem ODE Testsystem und den DAE Testsystemen. An diese Systeme mit den Startwerten $\mathbf{x}_{0,ode} = \begin{bmatrix} 3 & 0 \end{bmatrix}^T$, $\mathbf{x}_{0,dae} = \begin{bmatrix} 3 & 0 & 0 \end{bmatrix}^T$ wurde zum Zeitpunkt $t_0 = 0s$ ein Eingangssprung von $u = 100$ angelegt. Dies wurde numerisch mit einem expliziten Eulerverfahren fester Schrittweite h gelöst. Der lokale Diskretisierungsfehler, der der für den ersten Berechnungsschritt durch den Eingangssprung verursacht wurde, wurde berechnet und zusätzlich zu dessen Komponenten über die Schrittweite abgebildet.

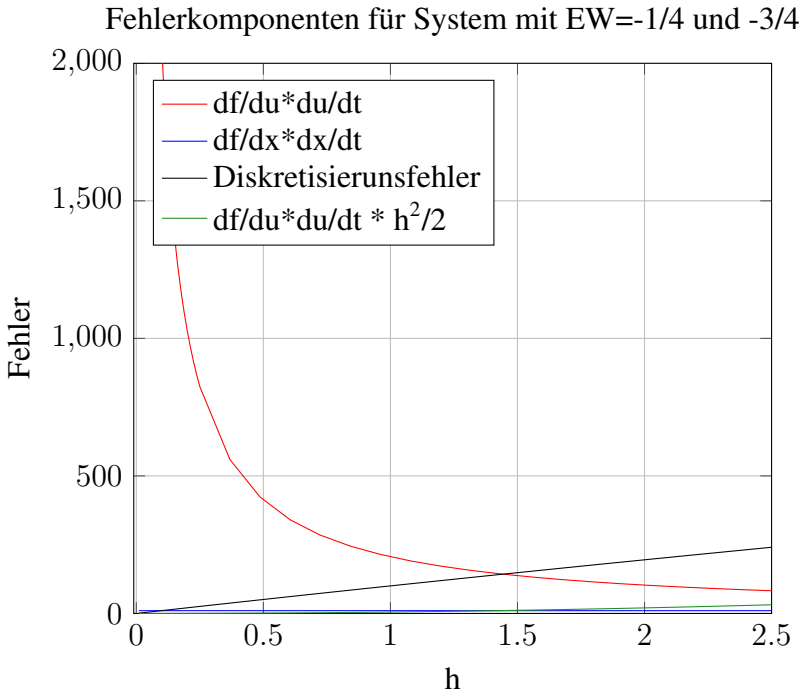


Abbildung 5.15: Fehlerauswertung für ODE System

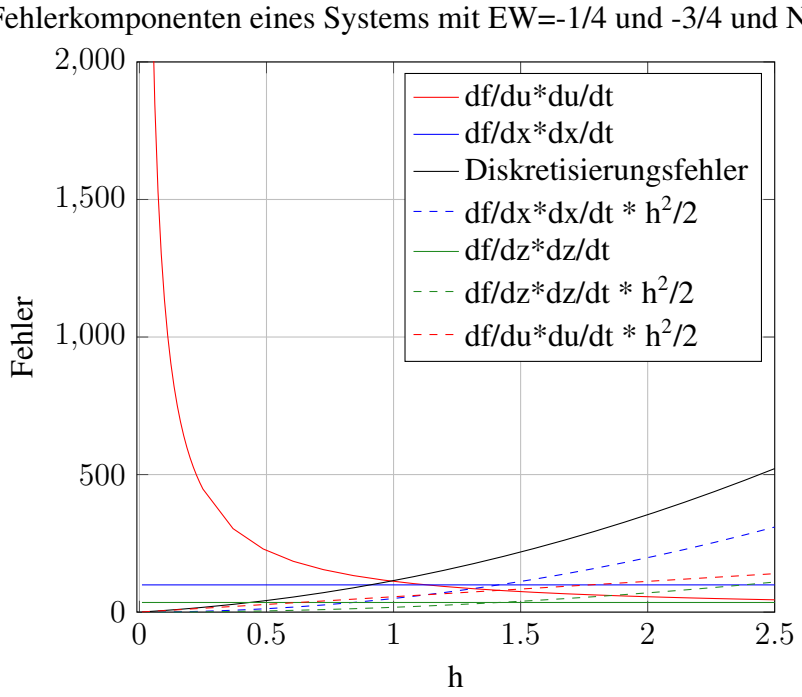


Abbildung 5.16: Fehlerauswertung für DAE System ohne u(t) in NB

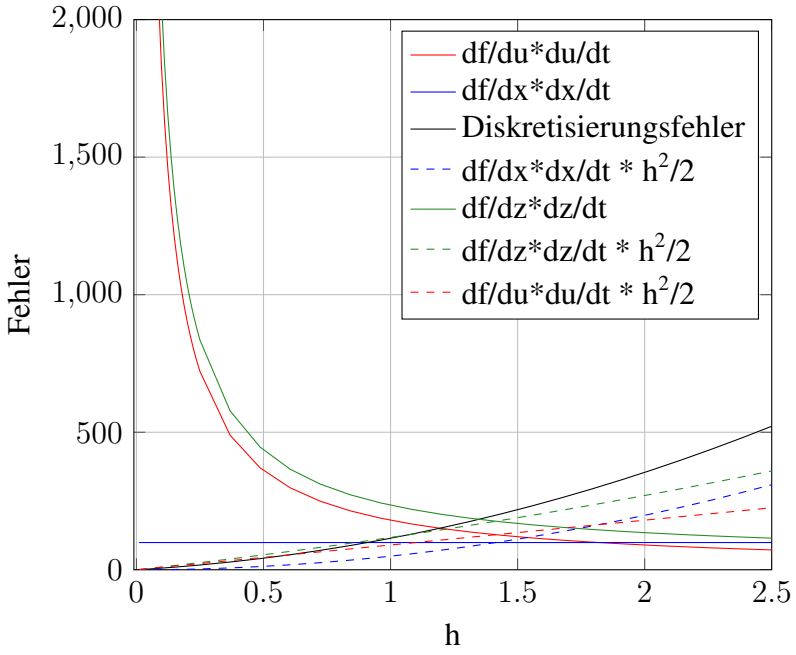


Abbildung 5.17: Fehlerauswertung für ODE System mit u(t) in NB

Der Fehler in Grafik 5.15 ist geringer als der in der Darstellung 5.16 und 5.17. Dies entspricht den Überlegungen aus Abschnitt 4.1 und 4.2.

Vergleicht man die Ergebnisse der DAE Systeme, unterscheiden sich die Gesamtfehler nicht voneinander, sie sind betragsmäßig gleich. Das würde bedeuten, dass der Eingang in der NB das numerische Ergebnis nicht weiter beeinflusst. Um die Darstellung zu bestätigen, wurde der mathematische Beweis im Anhang A erbracht. Da die algebraische Variable z nur auf x_1 wirkt, bleibt der Fehler der sich auf x_2 auswirkt in beiden DAE Systemen gleich, daher wird der Fehler x_1 betreffend angeführt. Wie bewiesen wurde, hat es keine Auswirkungen auf den numerischen Fehler, wenn die Eingangsgröße in der NB vorkommt.

5.2.2 Vorüberlegungen für Testfälle

Ein einfaches LTI System kann mit jedem in MATLAB mitgelieferten DGL-Lösungsverfahren berechnet werden. Diese Algorithmen wurden als Verfahren mit adaptiver Schrittweite implementiert. Ziel ist es eine Schrittweite h_i geeignet zu wählen, damit der lokale Fehler des akutellen Schrittes der Näherung x_{i+1} eine vorgegebene Fehlertoleranz unterschreitet. Aufgrund der Unbekanntheit der exakten Lösung $x(t_{i+1)}$ muss diese durch eine Approximation $\hat{x}_{i+1}$ ersetzt werden. Diese Vorgehensweise wird als Fehlerschätzung bezeichnet. Betrachtet man Runge-Kutta-Verfahren, wird die Approximation durch ein eingebettetes Verfahren berechnet. Es wird ein Verfahren der Ordnung p, für die Näherung x_{i+1}, mit einem Verfahren der Ordnung q, für die Schätzung $\hat{x}_{i+1}$, kombiniert, wobei $q = p + 1$ ist. Diese Verfahren werden als Runge-Kutta-Fehlberg-Verfahren bezeichnet. Der geschätzte Fehler ist die Differenz der beiden Lösungen. Dieser Ansatz gilt als überholt. Aktuelle eingebettete Verfahren verwenden zur Berechnung der Näherung das Verfahren höherer Ordnung p und für die Approximation das Verfahren niedrigerer Ordnung $q = p - 1$. Diese eingebetteten Runge-Kutta-Verfahren werden auch als $RKp(q)$ bezeichnet. [28], [32], [16]
Sämtliche Löser sind als $odep(q)$ angegeben, wobei q die Ordnung des Berechnungsverfahren und p die Ordnung der Fehlerschätzung darstellt [16]. Standardmäßig wird in MATLAB der Algorithmus $ode45$, oder auch Dormand-Prince (DOPRI) Verfahren, verwendet [18]. Es ist ein eingebettetes, explizites Runge-Kutta-Verfahren mit einer integrierten Schrittweitensteuerung. Dabei wird der nächste Wert x_{i+1} mit dem Verfahren 5. Ordnung berechnet und mit dem 4. Ordnung geschätzt $\hat{x}_{i+1}$. Die Differenz der beiden bezogen auf den Wertebereich von x ergibt den Fehler err. Ist dieser Fehler größer, als eine vorgegebene Fehlertoleranz $RelTol$ wird die Schrittweite verkleinert, solange bis die Fehlerschranke unterschritten wird. Ist err jedoch kleiner als $RelTol$ wird die Schrittweite erhöht. Damit kann angenommen werden, dass je kleiner $RelTol$ gewählt wird, umso genauer das Ergebnis wird und desto mehr Rechenschritte durchgeführt werden. Laut der MATLAB-Hilfe ist noch ein weiterer Parameter einzustellen, die absolute Toleranz $AbsTol$. Diese bestimmt die Genauigkeit, wenn die Lösung gegen Null geht.
Für $ode45$ ist der Codeausschnitt der Fehlerberechnung aus der Implementierung in MATLAB [18] gegeben.

```
err=absh*norm((f*E)./max(max(abs(y),abs(ynew)),threshold),inf);
```

Der Unterschied von x_{i+1} und $\hat{x}_{i+1}$ wird mit $absh * (f * E)$ berechnet. Die Skalierung wird durch die Supremumsnorm generiert, wobei y der alte Wert, $ynew$ der neue Wert und $threshold$

das Ergebnis der Division $\frac{AbsTol}{RelTol}$ ist. Somit ist Vorsicht geboten wie $AbsTol$ wählt wird. Ist $threshold$ größer als y und $ynew$, bezieht sich der Fehler auf diesen Wertebereich, so wird der Fehler immer klein gehalten, es ist also nicht garantiert, dass der Lösungsverlauf von Richtigkeit ist. Wie in [18] ist zu testen, welche Kombination von $RelTol$ und $AbsTol$ das beste Ergebnis liefert. Grundsätzlich ist es ein guter Ansatz $AbsTol$ kleiner als $RelTol$ zu wählen.
Geht man von einem einfachen LTI System mit kontinuierlichem Eingangssignal aus, wird ein Integrationsalgorithmus mit Schrittweitensteuerung und höherer Ordnung weniger Rechenschritte als ein Verfahren niedriger Ordnung brauchen. Die ist aufgrund der Fehlerschätzung zu argumentieren. Durch eine genauere Schätzung des Fehlers und der dadurch gewählten Schrittweite ist dies möglich.

Des weiteren ist durch Unstetigkeiten in der DGL eine Erhöhung der Anzahl der Berechnungsschritte zu erwarten [18]. Die Schrittweite wird an der Stelle der Unstetigkeit verkleinert bis die Fehlerschranke unterschritten wird. Im Gegensatz zu einem glatten Eingangssignal wird sich hier zeigen, dass durch eine genauere Fehlerschätzung im Falle von Unstetigkeiten mehr Rechenschritte erforderlich sind.
Schnelle Änderungen, wie bei einem steifen System haben dieselben Auswirkungen. Mehr Schritte erhöhen aber auch den Rundungsfehler. Steife Systeme können jedoch auch mit größerer Schrittweite gelöst werden [28]. Dafür sind A-stabile Integrationsverfahren geeignet. Implizite Verfahren sind A-stabil.

5.2.3 Testergebnisse des ODE-Systems

Das ODE Testsystem wurde mit verschiedenen Integrationsverfahren gelöst. Diese varieren in der Berechnungsordnung und der Wahl der Schrittweite. Es wurden Tests für Verfahren fester und zusätzlich mit variabler Schrittweite durchgeführt. Die Ergebnisse wurden aufbereitet und in Tabellenform dargestellt.

Numerische Integration mit fester Schrittweite

Angeführte Löser mit fester Schrittweite wurden verwendet: explizite Runge-Kutta Verfahren ode3 und ode5, explizites und implizites Eulerverfahren.
Folgende Schrittweiten wurden für die Tests eingeführt: $\mathbf{h}^T = (0.02, 0.04, 0.08, 0.12)$. Die Ergebnisse sind in Tabelle 5.1 aufgelistet.

Wie erwartet wächst mit steigender Schrittweite auch E_{sqmean}, sowie E_{max}. Zudem ist aus Tabelle 5.1 erkennbar, dass bei treppenförmigem Eingangssignal die Fehler größer sind als bei kontinuierlichem Eingangssignal. Weiteres ist gezeigt, dass ein Verfahren höherer Ordnung genauere numerische Werte liefert.

Tabelle 5.1: Ergebnisse der Simulation des ODE Testsystems für feste Schrittweiten

Löser	Schrittweite	E_{sqmean}	E_{max}	Eingang
ode3	0,02	6,5432e-08	6,3935e-04	stetig
	0,04	1,0448e-07	6,5643e-04	
	0,08	1,4952e-06	0,0034	
	0,12	1,0940e-05	0,0086	
	0,02	0,0465	0,3749	Treppe
	0,04	0,0512	0,4102	
	0,08	0,0512	0,4413	
	0,12	0,0643	0,4836	
ode5	0,02	5,1163e-10	5,6426e-05	stetig
	0,04	5,7249e-08	6,4424e-04	
	0,08	6,4998e-07	0,0016	
	0,12	1,1006e-06	0,0026	
	0,02	0,0484	0,3932	Treppe
	0,04	0,0465	0,3897	
	0,08	0,0513	0,4357	
	0,12	0,0483	0,4197	
expl. Euler	0,02	0,0027	0,1115	stetig
	0,04	0,0109	0,2223	
	0,08	0,0436	0,4400	
	0,12	0,1002	0,6674	
	0,02	0,0557	0,3936	Treppe
	0,04	0,0521	0,3573	
	0,08	0,0526	0,4230	
	0,12	0,0928	0,5580	
impl. Euler	0,02	0,0027	0,1105	stetig
	0,04	0,0106	0,2178	
	0,08	0,0408	0,4232	
	0,12	0,0918	0,6282	
	0,02	0,0635	0,4955	Treppe
	0,04	0,0749	0,5513	
	0,08	0,1102	0,6709	
	0,12	0,2523	0,8887	

Betrachtet man die Ergebnisse der beiden Eulerverfahren fällt auf, dass die Fehlerwerte nicht identisch sind, wie jedoch aus der Herleitung in 4.1.3 zu erwarten wäre. Dazu seien die Simulationsergebnisse beider Verfahren für das kontinuierliche System grafisch dargestellt, siehe MATLAB-Plots.

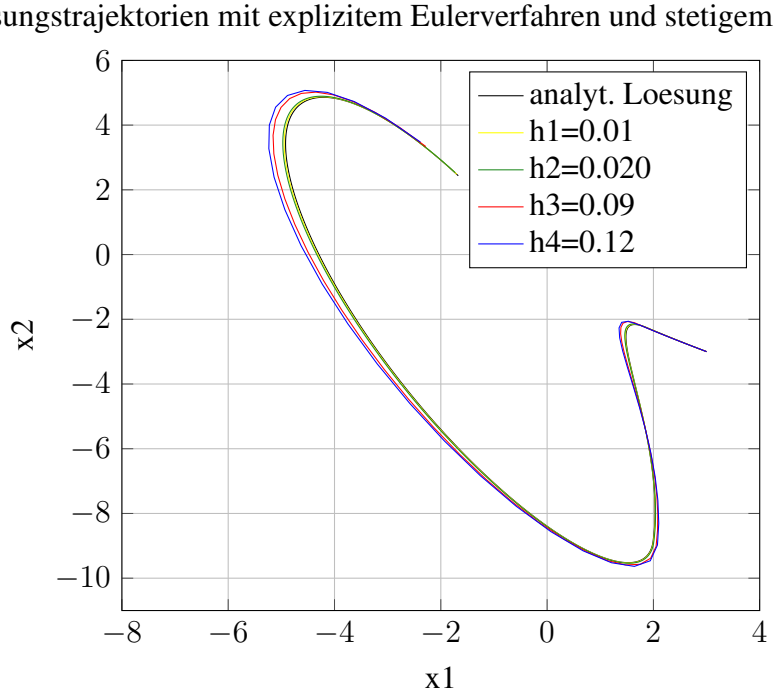


Abbildung 5.18: ODE System mit glattem $u(t)$ und explizitem Eulerverfahren fester Schrittweite

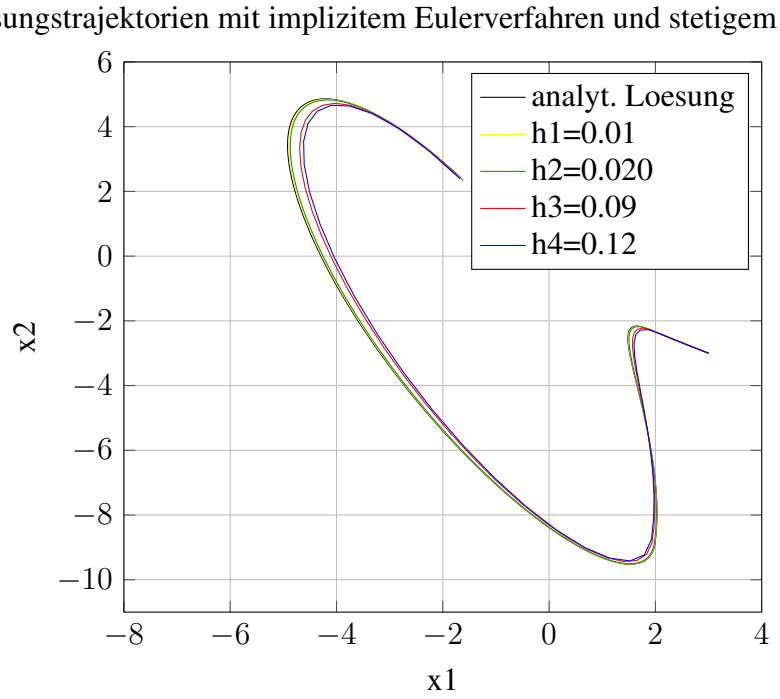


Abbildung 5.19: ODE System mit glattem $u(t)$ und implizitem Eulerverfahren fester Schrittweite

Aus den Abbildung 5.18 und 5.19 ist erkennbar, dass das Vorzeichen der Fehler gegensätzlich ist, das auch in Kapitel 4.1 angesprochen wird. Die Differenz im Betrag lässt sich einmal durch den Lösungsverlauf und durch die Implementierung des impliziten Verfahrens begründen, da zusätzlich ein algebraischer Löser verwendet werden muss, der ebenfalls nicht exakte Werte liefert.

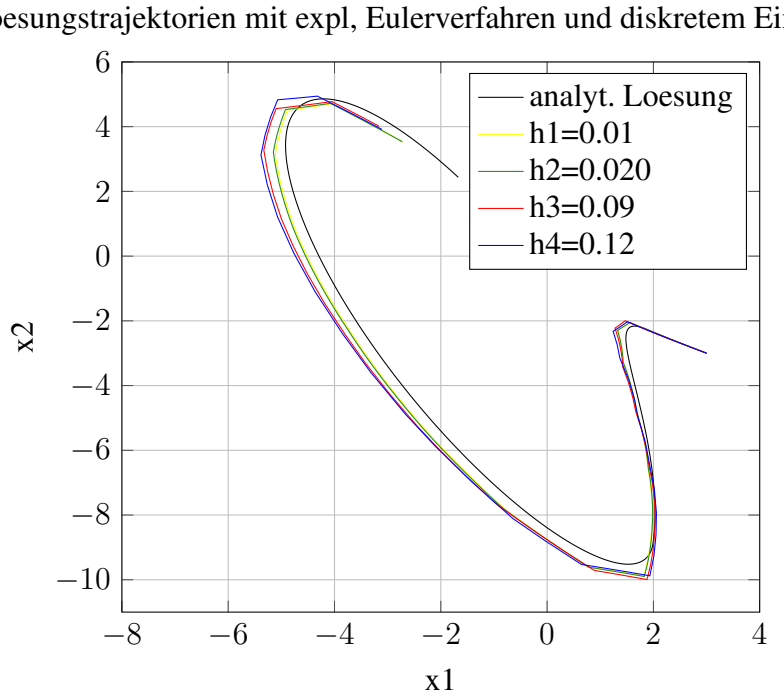


Abbildung 5.20: ODE System mit $u_{tr}(t)$ und explizitem Eulerverfahren fester Schrittweite

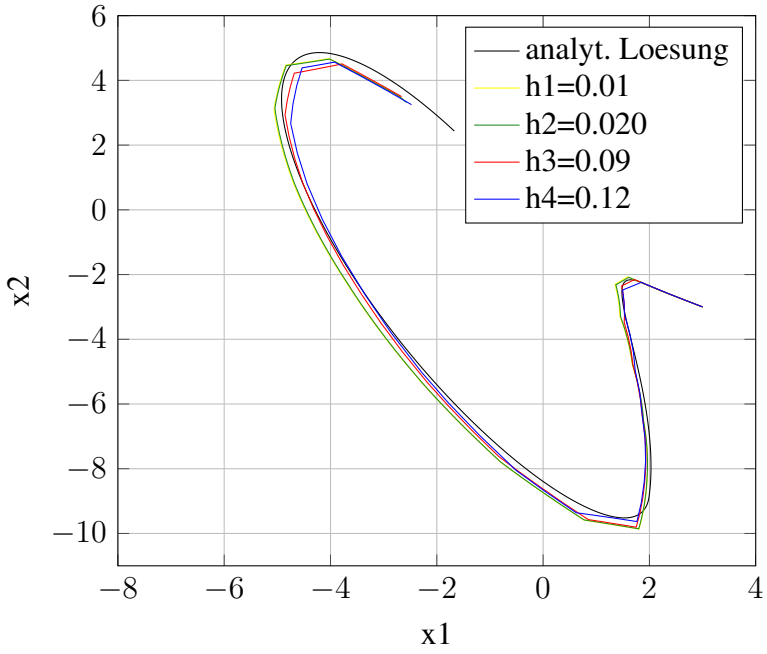


Abbildung 5.21: ODE System mit $u_{tr}(t)$ und implizitem Eulerverfahren fester Schrittweite

Ergänzend sind die Grafiken der Simulationsergebnisse der Eulerverfahren für eine treppen-

förmige Eingangsfunktion abgebildet, Plots 5.20 und 5.21.

Numerische Integration mit variabler Schrittweite

Angeführte Löser mit Schrittweitensteuerung wurden verwendet: ode23, ode45, explizites und implizites Eulerverfahren.
Dabei wurde die relative Toleranz $RelTol$ variiert: $RelTol = (1e-5, 1e-4, 1e-3, 1e-2)$.
Dabei ergaben sich die Simulationsergebnisse aus Tabelle 5.2.

Aus Tabelle 5.2 ist gut erkennbar, dass für kleine $RelTol$ die numerischen Ergebnisse besser sind und je größer die Toleranz gewählt wird, desto weniger Schritte werden berechnet. Die Verfahren höherer Ordnung liefern laut Fehlerbewertung schlechtere Ergebnisse. Dazu muss man die Ergebnisse genauer betrachten. Da die Verfahren höherer Ordnung eine bessere Schrittweitensteuerung besitzen, kann mit weniger Schritten ein sehr genaues Ergebnis erzielt werden. Weiteres wird gezeigt, dass durch Unstetigkeiten am Eingang sich die Schrittweite verkleinern muss, um unter der Fehlerschranke zu bleiben. Es wurde gezeigt, dass Verfahren höherer Ordnung im Falle von Unstetigkeiten im Eingangssignal mehr Berechnungsschritte brauchen als Verfahren niedriger Ordnung, da sie eine bessere Fehlerschätzung besitzen. Ausnahme ist hier das Eulerverfahren, da die Fehlerschätzung und das Verfahren nicht sehr genau sind, werden immer viele Berechnungsschritte benötigt. Natürlich braucht jedes Verfahren eine Mindestanzahl an Schritten, um eine richtige Lösung berechnen zu können.
Im Vergleich zu Algorithmen mit fester Schrittweite liegt der Vorteil von einem Verfahren mit Schrittweitensteuerung klar auf der Hand. Wählt man eine feste Schrittweite, so soll diese sehr klein sein, damit ein gutes Ergebnis erzielt wird. Mit variabler Schrittweite werden mit weniger Schritten bessere Ergebnisse erzielt. Besonders ersichtlich ist dies bei Unstetigkeiten in der Lösungstrajektorie, die ebenso durch Unstetigkeiten im Eingang entstehen können. Das adaptive Verfahren kann aber in speziellen Fällen abbrechen. Wählt man den Wert für $RelTol$ zu klein und der adaptive Algorithmus kann die Schrittweite nicht weiter verkleinern, da er das Minimum erreicht hat, kann er die Fehlerschranke nicht unterschreiten und bricht ab.

5.2.4 Testergebnisse des steifen Systems

Für das steife System wurden die selben Tests durchgeführt jedoch mit zum Teil anderen numerischen Integrationsverfahren. Für steife Systeme wurde in MATLAB der Standardlöser für steife Systeme $ode15s$ implementiert, welcher ein implizites Mehrschrittverfahren variabler Ordnung darstellt [18].

Tabelle 5.2: Ergebnisse der Simulation des ODE Testsystems für variable Schrittweiten

Löser	Schritte	Reltol	E_{sqmean}	E_{max}	Eingang
ode23	146	1e-05	4,4456e-10	3,7627e-05	stetig
	70	1e-04	3,4870e-08	4,4298e-04	
	33	1e-03	2,0600e-05	0,0045	
	15	1e-02	0,0018	0,1062	
	225	1e-05	0,0048	0,3749	Treppe
	139	1e-04	0,0481	0,3750	
	74	1e-04	0,0600	0,4580	
	24	1e-02	0,0665	0,4316	
ode45	92	1e-05	2,0274e-06	0,0055	stetig
	52	1e-04	1,1270e-05	0,0127	
	44	1e-03	2,8593e-04	0,0399	
	44	1e-02	4,8429e-04	0,0648	
	524	1e-05	0,0428	0,3749	Treppe
	276	1e-04	0,0370	0,3871	
	80	1e-04	0,0490	0,5115	
	44	1e-02	0,0743	0,6172	
expl. Euler	2437	1e-05	6,4735e-11	1,3040e-05	stetig
	790	1e-04	5,9897e-09	1,3573e-04	
	250	1e-03	6,0673e-07	0,0013	
	78	1e-02	5,3562e-05	0,0117	
	2207	1e-05	0,0499	0,3749	Treppe
	771	1e-04	0,0493	0,3748	
	277	1e-04	0,0493	0,3770	
	104	1e-02	0,0386	0,3520	
impl. Euler	2437	1e-05	3,5311e-10	1,3040e-05	stetig
	788	1e-04	2,9100e-09	1,3573e-04	
	249	1e-03	4,3201e-07	0,0013	
	80	1e-02	1,6366e-05	0,0117	
	2254	1e-05	0,0497	0,3749	Treppe
	802	1e-04	0,0492	0,3748	
	288	1e-04	0,0473	0,3770	
	110	1e-02	0,0385	0,3520	

Numerische Integration mit fester Schrittweite

Es wurden folgende numerische Integrationsverfahren mit fester Schrittweite $\mathbf{h}^T = (0.01, 0.002, 0.006, 0.0066)$ verwendet: explizites Runge-Kutta Verfahren $ode3$ und $ode4$, sowie explizites und implizites Eulerverfahren. Die Ergebnisse befinden sich in Tabelle 5.3.

Auch im Falle des steifen Systems sind ähnliche Beobachtungen, wie für das ODE System aus dem vorigen Abschnitt 5.2.3 gemacht worden. Das implizite Verfahren liefert genauere Werte. Dies lässt sich aufgrund der A-Stabilität impliziter Verfahren erklären. Aus Grafik 5.22 und 5.23 ist ersichtlich, dass ab einer gewissen Schrittweite die numerische Lösung des expliziten Lösungsalgorithmus zu oszillieren beginnt. Wird das implizite Verfahren mit der selben Schrittweite durchgeführt, wie in Darstellung 5.24 gezeigt, erhält man einen guten Lösungsverlauf. Aus diesem Grund eignen sich implizite Integrationsverfahren für steife Systeme besser als explizite, da aufgrund des A-stabilen Verhaltens impliziter Algorithmen größere Werte für die Schrittweite gewählt werden können.

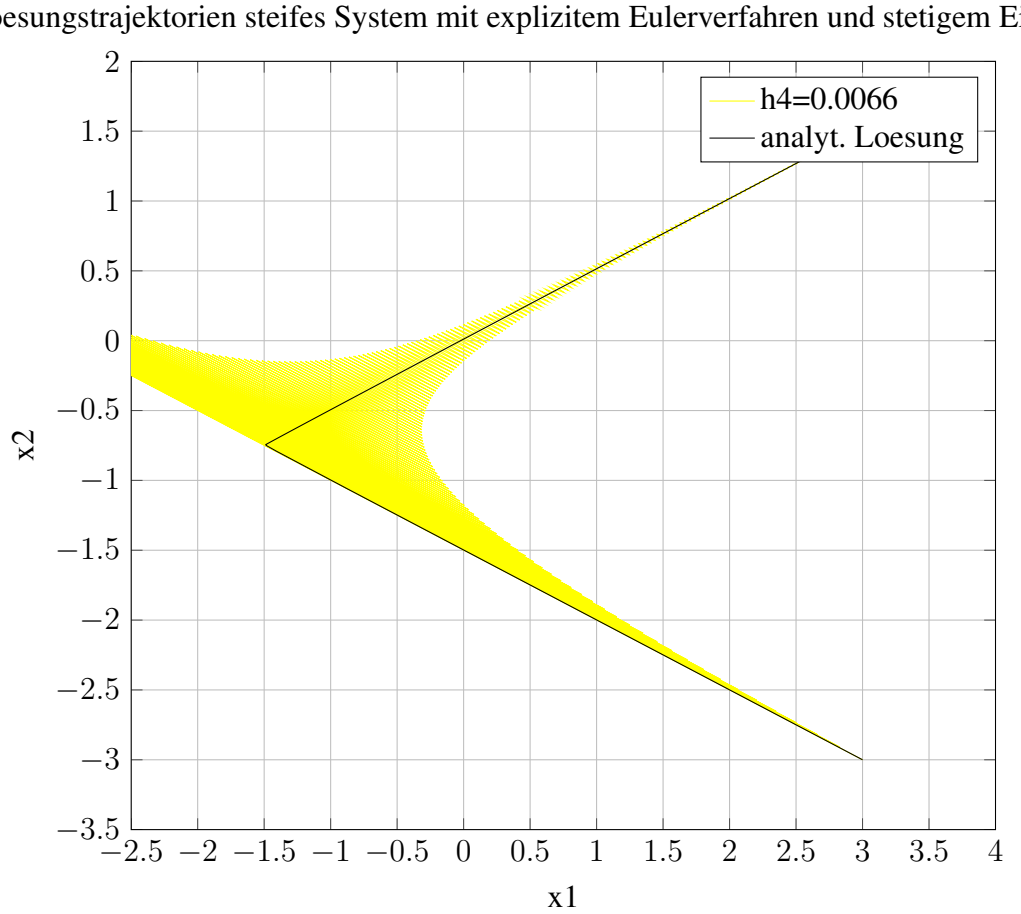


Abbildung 5.22: Steifes Testsystem mit glattem Eingang und explizitem Eulerverfahren fester Schrittweite

Tabelle 5.3: Ergebnisse der Simulation des steifen Testsystems für feste Schrittweiten

Löser	Schrittweite	E_{sqmean}	E_{max}	Eingang
ode3	0,0010	4,8077e-09	0,0019	stetig
	0,0050	9,3661e-05	0,1623	
	0,0060	0,0020	0,8686	
	0,0066	0,0069	1,5339	
	0,0010	2,3329e-08	0,0019	Treppe
	0,0050	9,4116e-05	0,1623	
	0,0060	0,0020	0,8686	
	0,0066	0,0069	1,5339	
ode5	0,0010	2,8388e-15	1,4473e-06	stetig
	0,0050	1,3273e-07	0,0067	
	0,0060	1,1522e-06	0,0182	
	0,0066	5,2952e-06	0,0307	
	0,0010	1,7276e-08	2,2028e-04	Treppe
	0,0050	5,6338e-07	0,0067	
	0,0060	3,2193e-06	0,0182	
	0,0066	6,2202e-06	0,0307	
expl. Euler	0,0010	6,4605e-05	0,2270	stetig
	0,0050	0,0134	2,4218	
	0,0060	0,0612	3,8543	
	0,0066	0,7754	4,7096	
	0,0010	6,4217e-05	0,2270	Treppe
	0,0050	0,0134	2,4218	
	0,0060	0,0612	3,8542	
	0,0066	0,7711	4,7095	
impl. Euler	0,0010	4,5545e-05	0,1751	stetig
	0,0050	3,7743e-04	0,3313	
	0,0060	2,8177e-04	0,2078	
	0,0066	2,2930e-04	0,2093	
	0,0010	4,6222e-05	0,1751	Treppe
	0,0050	3,9434e-04	0,3313	
	0,0060	3,5597e-04	0,2077	
	0,0066	2,3753e-04	0,2092	

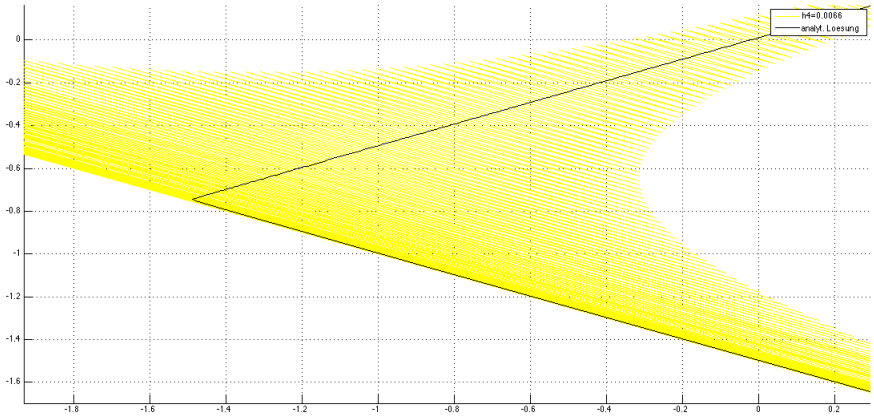


Abbildung 5.23: Ausschnitt steifes Testsystem mit glattem Eingang und explizites Eulerverfahren mit fester Schrittweite

Loesungstrajektorien steifes System mit implizitem Eulerverfahren und stetigem Eingang u(t)

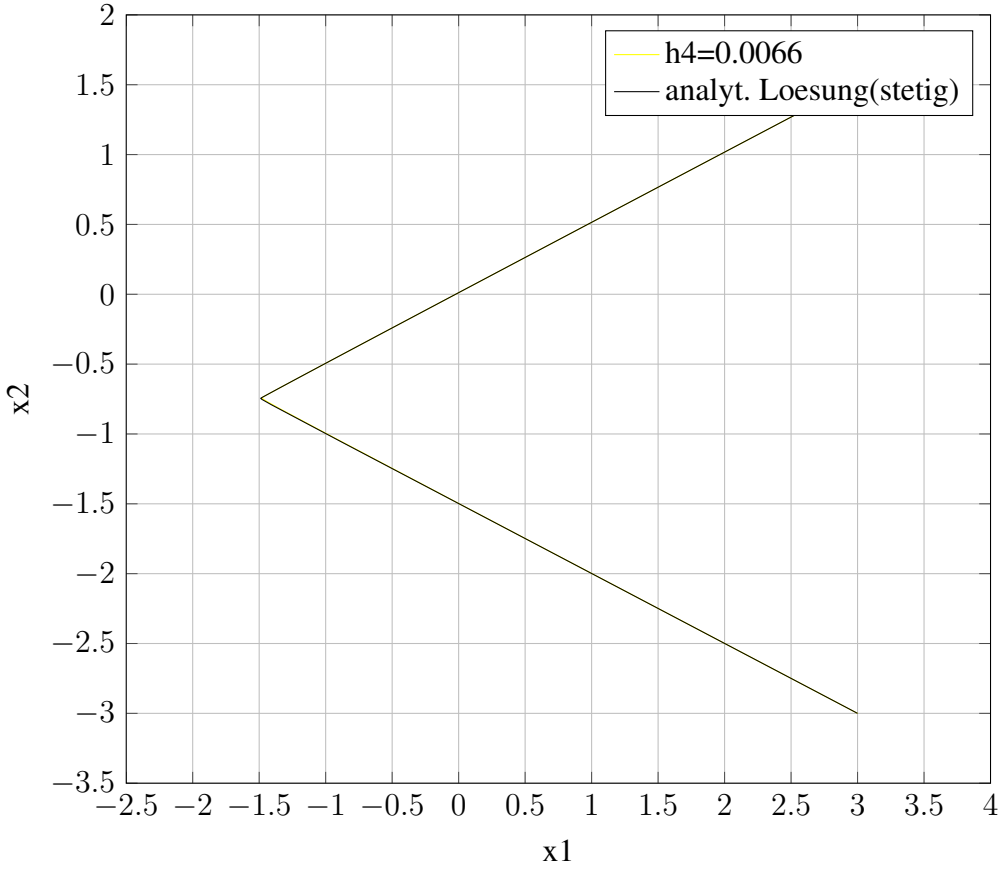


Abbildung 5.24: Steifes Testsystem mit glattem Eingang und implizitem Eulerverfahren fester Schrittweite

Numerische Integration mit variabler Schrittweite

Es wurden folgende numerische Integrationsverfahren verwendet: ode23, ode45, ode12, explizites und implizites Eulerverfahren.
Dabei wurde die relative Toleranz $RelTol$ variiert: $RelTol = (1e-4, 1e-3, 1e-2)$. Es ergaben sich die Simulationsergebnisse aus Tabelle 5.4.

Explizite Integrationsalgorithmen rechnen mit sehr kleiner Schrittweite, um ein gutes Ergebnis liefern zu können. Je höher die Ordnung des numerischen Verfahrens, desto mehr Berechnungsschritte werden durchgeführt. Einen guten Vergleich kann man aus dem Eulerverfahren ziehen. Durch die numerische Integration mit dem impliziten Euleralgorithmus kann das Ergebnis mit weniger Schritten berechnet werden.
Auch hier wirken sich Unstetigkeiten im Eingangssignal auf den Lösungsvorgang aus. Die Anzahl der Berechnungsschritte erhöht sich, jedoch für explizite Löser ist die Differenz nicht so gravierend wie in Abschnitt 5.2.3, da für den schnellen EW des Systems auch schon sehr viele Rechenschritte durchgeführt werden.
Das Standardverfahren von MATLAB liefert dabei sehr gute Werte bezogen auf die Anzahl der Schritte, welche im Gegensatz zu den anderen Verfahren sehr gering sind.

5.2.5 Testergebnisse der DAE-Systeme

In diesem Abschnitt wurden beide DAE Systeme unter den selben Bedingungen simuliert und ausgewertet.
Es wurden folgende numerische Integrationsverfahren verwendet: $ode15s$, $ode15i$, sowie implizites und explizites Eulerverfahren mit fester und variabler Schrittweite.
Der Lösungsalgorithmus $ode15i$ ist ein numerisches Verfahren variabler Ordnung, welches implizite DAE Gleichungen mit konsistenten Anfangswerten löst. Es ist ein LMSV, ein sogenanntes BDF-Verfahren das mit einem Lagrangeschen Polynom interpoliert. [27]
Zusätzlich zu den Zustandsvariablen $\mathbf{x}(t)$ ist auch die algebraische Variable $z(t)$ zu lösen. Somit wird auch für diese eine absolute Fehlertoleranz festgelegt.

Numerische Integration mit fester Schrittweite

Die angeführten Löser mit fester Schrittweite wurden verwendet: explizites und implizites Eulerverfahren.
Folgende Schrittweiten wurden für die Tests eingeführt: $\mathbf{h}^T = (0.01, 0.02, 0.09, 0.12)$. Die Ergebnisse sind in Tabelle 5.5 und 5.6 aufgelistet.

Tabelle 5.4: Ergebnisse der Simulation des steifen Testsystems für variable Schrittweiten

Löser	Schritte	Reltol	E_{sqmean}	E_{max}	Eingang
ode23	308	1e-04	7,8440e-09	2,5589e-04	stetig
	287	1e-03	8,0163e-07	0,0024	
	278	1e-02	8,4338e-05	0,0306	
	333	1e-04	1,0216e-08	3,6923e-04	Treppe
	288	1e-03	7,1081e-07	0,0024	
	277	1e-02	7,1077e-05	0,0306	
ode45	880	1e-04	1,1294e-09	1,7051e-04	stetig
	860	1e-03	1,1043e-07	0,0019	
	848	1e-02	1,0770e-05	0,0155	
	932	1e-04	1,4640e-06	0,0016	Treppe
	872	1e-03	5,4551e-07	0,0018	
	848	1e-02	9,4422e-06	0,0171	
ode15s	73	1e-04	2,7850e-08	5,4933e-04	stetig
	60	1e-03	7,2520e-07	0,0035	
	37	1e-02	5,2855e-05	0,0273	
	228	1e-04	1,1522e-07	0,0011	Treppe
	130	1e-03	1,5561e-05	0,0142	
	57	1e-02	6,0256e-04	0,0438	
expl. Euler	859	1e-04	2,8431e-10	5,8409e-05	stetig
	496	1e-03	5,6200e-08	5,9665e-04	
	391	1e-02	7,6259e-06	0,0074	
	939	1e-04	1,4754e-10	3,5393e-05	Treppe
	526	1e-03	7,0333e-08	0,0010	
	393	1e-02	7,9872e-06	0,0060	
impl. Euler	732	1e-04	6,6944e-11	5,2416e-05	stetig
	240	1e-03	3,5881e-09	3,2792e-04	
	80	1e-02	2,2201e-07	9,4885e-04	
	768	1e-04	2,3243e-10	3,5723e-05	Treppe
	277	1e-03	1,2102e-07	0,0014	
	100	1e-02	1,9924e-05	0,0096	

Tabelle 5.5: Ergebnisse der Simulation des DAE Testsystems ohne $u(t)$ in NB für feste Schrittweiten

Löser	Schrittweite	E_{sqmean}	E_{max}	Eingang
expl. Euler	0,01	6,7758e-04	0,0566	stetig
	0,02	0,0027	0,1115	
	0,09	0,0560	0,4964	
	0,12	0,1002	0,6674	
	0,01	4,1738e-04	0,0454	Treppe
	0,02	0,0017	0,0909	
	0,09	0,0294	0,4646	
	0,12	0,0402	0,4844	
impl. Euler	0,01	6,7268e-04	0,0554	stetig
	0,02	0,0027	0,1105	
	0,09	0,0513	0,4747	
	0,12	0,0918	0,6282	
	0,01	0,0015	0,1191	Treppe
	0,02	0,0063	0,2364	
	0,09	0,0825	0,5694	
	0,12	0,1796	0,8611	

Tabelle 5.6: Ergebnisse der Simulation des DAE Testsystems ohne $u(t)$ mit NB für feste Schrittweiten

Löser	Schrittweite	E_{sqmean}	E_{max}	Eingang
expl. Euler	0,01	6,7758e-04	0,0566	stetig
	0,02	0,0027	0,1115	
	0,09	0,0560	0,4964	
	0,12	0,1002	0,6674	
	0,01	4,1738e-04	0,0454	Treppe
	0,02	0,0017	0,0909	
	0,09	0,0294	0,4646	
	0,12	0,0402	0,4844	
impl. Euler	0,01	6,7268e-04	0,0554	stetig
	0,02	0,0027	0,1105	
	0,09	0,0513	0,4747	
	0,12	0,0918	0,6282	
	0,01	0,0015	0,1191	Treppe
	0,02	0,0063	0,2364	
	0,09	0,0825	0,5694	
	0,12	0,1796	0,8611	

In Tabelle 5.5 wird gezeigt, dass der numerische Fehler bei einem System mit Unstetigkeiten in der Eingangsfunktion einen deutlich höheren Wert als für einen glatten Eingang aufweist. Die unterschiedlichen Fehlerwerte für das implizite und explizite Eulerverfahren wurden im vorangegangenen Kapitel 5.2.3 bereits erwähnt.
Vergleicht man die Simulationsergebnisse beider Tabellen, ist ersichtlich, dass die Fehler betragsmäßig gleich sind. Dies ging auch aus der theoretischen Fehlerauswertung in Abschnitt 5.2.1 hervor und ist somit auch praktisch gezeigt.

Numerische Integration mit variabler Schrittweite

Die angeführten Löser mit Schrittweitensteuerung wurden verwendet: ode15i, ode15s, explizites und implizites Eulerverfahren.
Dabei wurde die relative Toleranz $RelTol$ variiert: $RelTol = (1e-3, 1e-2, 1e0)$. Für $RelTol = 1e-3$ wurde die absolute Toleranz $AbsTol = (1e-6, 1e-3, 1e-1)$ der algebraischen Variable $z(t)$ verändert. Es gibt an, wie genau für die algebraische Variable gelöst werden soll. Dabei ergaben sich die Simulationsergebnisse aus Tabelle 5.7.

Auch in Tabelle 5.7 ist zu erkennen, dass Unstetigkeiten eine Erhöhung der Schrittanzahl zu

Tabelle 5.7: Ergebnisse der Simulation des DAE Testsystems ohne $u(t)$ in NB für variable Schrittweiten

Löser	Schritte	Reltol	Abstol von z	E_{sqmean}	E_{max}	Eingang
ode15i	57	1e-03	1e-06	1,7520e-06	0,0029	stetig
	56	1e-03	1e-03	8,4426e-06	0,0109	
	52	1e-03	1e-01	1,1515e-05	0,0132	
	35	1e-02	1e-06	0,0012	0,1044	
	12	1	1e-06	1,3086	2,0401	
	218	1e-03	1e-06	2,9179e-05	0,0207	Treppe
	232	1e-03	1e-03	7,2207e-05	0,0198	
	211	1e-03	1e-01	5,3995e-05	0,0410	
	114	1e-02	1e-06	0,0042	0,1562	
	16	1	1e-03	0,9260	1,8915	
ode15s	44	1e-03	1e-06	3,2971e-05	0,0150	stetig
	46	1e-03	1e-03	2,0364e-05	0,0117	
	44	1e-03	1e-01	1,4653e-04	0,0293	
	30	1e-02	1e-06	0,0015	0,0732	
	10	1	1e-06	0,8302	1,7213	
	165	1e-03	1e-06	9,9301e-05	0,0238	Treppe
	175	1e-03	1e-03	1,1640e-04	0,0202	
	162	1e-03	1e-01	1,0289e-04	0,0274	
	68	1e-02	1e-06	0,0062	0,5084	
	10	1	1e-03	0,6902	1,6823	
expl. Euler	253	1e-03	1e-06	4,1107e-05	0,0155	stetig
	82	1e-02	1e-06	2,6577e-04	0,0426	
	51	1	1e-06	5,6708e-04	0,0634	
	284	1e-03	1e-06	3,4872e-05	0,0115	Treppe
	112	1e-02	1e-06	2,5002e-04	0,0367	
	51	1	1e-06	0,0491	0,4159	
impl. Euler	249	1e-03	1e-06	4,5307e-07	0,0015	stetig
	80	1e-02	1e-06	1,6415e-05	0,0082	
	51	1	1e-06	3,6089e-05	0,0108	
	295	1e-03	1e-06	1,1813e-06	0,0026	Treppe
	111	1e-02	1e-06	1,7070e-04	0,0241	
	51	1	1e-06	0,0463	0,4115	

Tabelle 5.8: Ergebnisse der Simulation des DAE Testsystems mit $u(t)$ in NB für variable Schrittweiten

Löser	Schritte	Reltol	Abstol von z	E_{sqmean}	E_{max}	Eingang
ode15i	42	1e-03	1e-06	abgebrochen	abgebrochen	stetig
	49	1e-03	1e-03	abgebrochen	abgebrochen	
	61	1e-03	1e-01	6,2406e-05	0,0168	
	35	1e-02	1e-06	abgebrochen	abgebrochen	
	15	1	1e-06	1,2211	2,1143	
	43	1e-03	1e-06	abgebrochen	abgebrochen	Treppe
	50	1e-03	1e-03	abgebrochen	abgebrochen	
	41	1e-03	1e-01	abgebrochen	abgebrochen	
	44	1e-02	1e-06	abgebrochen	abgebrochen	
	24	1	1e-03	0,7453	2,849	
ode15s	42	1e-03	1e-06	abgebrochen	abgebrochen	stetig
	42	1e-03	1e-03	abgebrochen	abgebrochen	
	46	1e-03	1e-01	2,5264e-04	0,0334	
	25	1e-02	1e-06	abgebrochen	abgebrochen	
	12	1	1e-06	0,6305	1,4333	
	44	1e-03	1e-06	abgebrochen	abgebrochen	Treppe
	31	1e-03	1e-03	abgebrochen	abgebrochen	
	30	1e-03	1e-01	abgebrochen	abgebrochen	
	22	1e-02	1e-06	abgebrochen	abgebrochen	
	21	1	1e-03	abgebrochen	abgebrochen	
expl. Euler	462	1e-03	1e-06	0,0010	0,0842	stetig
	146	1e-02	1e-06	0,0103	0,3084	
	51	1	1e-06	0,1182	0,9264	
	304	1e-03	1e-06	2,4601e-05	0,0131	Treppe
	137	1e-02	1e-06	6,5018e-04	0,0840	
	53	1	1e-06	0,1270	0,8297	
impl. Euler	249	1e-03	1e-06	4,6339e-07	0,0014	stetig
	80	1e-02	1e-06	1,6904e-05	0,0082	
	51	1	1e-06	3,9052e-05	0,0124	
	295	1e-03	1e-06	1,1812e-06	0,0026	Treppe
	111	1e-02	1e-06	1,7070e-04	0,0241	
	51	1	1e-06	0,0463	0,4115	

Folge haben. Ebenfalls haben sie auch negative Auswirkungen auf den Fehler.
Am besten bewährt sich der integrierte Löser ode15i. Auffallend ist, wenn man die Genauigkeit der algebraischen Variable $z(t)$ erhöht, vergrößert sich auch der numerische Fehler.

Aus Tabelle 5.8 geht hervor, dass die NB, die das Eingangssignal $u(t)$ beinhaltet, Auswirkungen auf das Lösungsverfahren der implementierten Löser in MATLAB hat. Wählt man die $AbsTol$ für z zu klein, bricht das Verfahren ab. Für ein kontinuierliches Eingangssignal kann es für $AbsTol = 0.1$ gelöst werden. Hingegen für ein treppenförmiges Eingangssignal abbricht. Dies ist für genaue numerische Integrationsverfahren höherer Ordnung deutlich zu erkennen.
Unterzieht man die Ergebnisse der Eulerverfahren einem Vergleich, ist klar erkennbar, dass das explizite Lösungsverfahren im Fall des Systems 5.9 deutlich mehr Berechnungsschritte benötigt als beim System 5.10. Auch die Fehler weisen einen deutlich höheren Wert auf. Wo hingegen die Werte für das implizite Verfahren annähernd gleich sind.

Kapitel 6

Evaluierung an einem Abschlussbeispiel

Die aus dem vorangegangenen Kapitel erlangten Erkenntnisse sollen nun anhand eines Co-Simulationsbeispieles in ICOS getestet werden. Dazu wird der in [22] beschriebene Gelenkmechanismus in OpenModelica (OM) implementiert. Die Modellbildung erfolgte über die Lagrange Gleichungen 2. Art. In diesem Fall handelt es sich um ein System von gekoppelten nichtlinearen Differentialgleichungen. Die Eingangssignale werden von einem MATLAB-Modell generiert, das nicht-iterativ mit dem OM-Modell gekoppelt wird. Anhand dieses Co-Simulationsbeispieles werden einige Versuche gemacht und schlussendlich durch die Fehlerbewertung mit den vorgestellten Fehlermaßen analysiert.

6.1 Mathematisches Modell

Die Berechnungen für das mathematische Modell des in Abbildung 6.1 dargestellten Gelenkmechanismus (5R Mechanismus) wurden in [22] erbracht. Das Modell besteht aus 4 Gelenken, von denen zwei angetrieben werden können. Durch geeignete Vorgabe der Eingänge in Form eines Spannungsverlaufes werden die Winkel des jeweiligen Armes eingestellt. Dadurch wird die Position des Endeffektors E vorgegeben. Somit ergeben die beiden Winkel β_1 und β_2 die, für die Modellbildung, gewählten generalisierten Koordinaten $\mathbf{q}^T = (q1, q2)$.

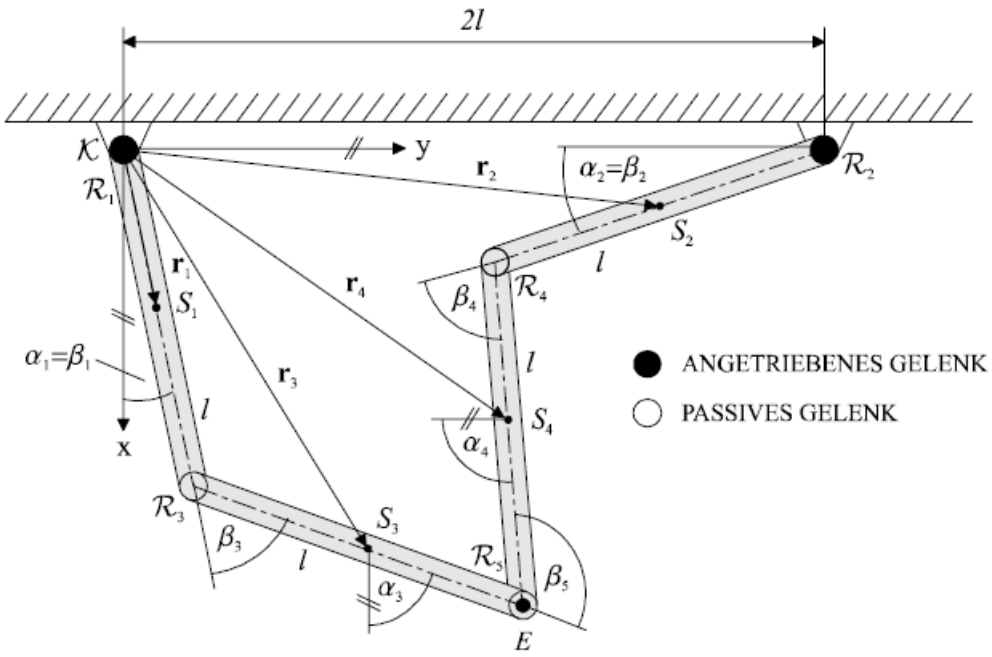


Abbildung 6.1: Absolute und relative Lagegrößen des 5R Mechanismus [22]

Daraus ergibt sich folgendes ODE Gleichungssystem, mit Differenzialgleichungen 2. Ordnung:

$$\ddot{\mathbf{q}} = (\mathbf{M}(\mathbf{q}) + \mathbf{J_{ges}})^{-1} \left(K_2\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - K_1\dot{\mathbf{q}} - \boldsymbol{\tau}_R\right) \tag{6.1}$$

Zur Auswertung ist die 2x2 Massenmatrix $\mathbf{M}$ und die 2x2 Matrix der Zentrifugal- und Coriolisterme $\mathbf{C}$ notwendig, sowie das Gesamtträgheitsmoment in Form der 2x2 Matrix $\mathbf{J}_{ges}$. Der Zeilenvektor $\boldsymbol{\tau}_R$ drückt das Reibungsmoment aus. Die Konstanten K_1 und K_2 ergeben sich aus dem elektromechanischen Modell des Antriebes. In der Diplomarbeit [22] wurden die Herleitungen für $\mathbf{M}, \mathbf{C}, \mathbf{J}_{ges}, \boldsymbol{\tau}_R, K_1, K_2$ erbracht. Der Eingangsvektor $\mathbf{u}$ besitzt die Dimension 2x1.
Für die Darstellung DGL der (6.1) als Zustandsraummodell, ergibt sich der Zustandsvektor $\mathbf{x}$ wie folgt:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{pmatrix} \tag{6.2}$$

Somit kann das Zustandsraummodell angeschrieben werden als:

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \ddot{x_3} \\ \ddot{x_4} \end{pmatrix} = \begin{pmatrix} x_3 \\ x_4 \\ (\mathbf{M}(\mathbf{x_1}, \mathbf{x_2}) + \mathbf{J_{ges}})^{-1} \left(K_2\mathbf{u} - \mathbf{C}(x_1, x_2, x_3, x_4) \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} - K_1 \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} - \boldsymbol{\tau}_R \right) \end{pmatrix} \tag{6.3}$$

Im Zuge dieser Masterarbeit wurde das Zustandsraummodell (6.3) in OM [23] implementiert, siehe Anhang B.

6.2 Kopplung mit ICOS

Um das OM-Modell des Gelenkmechanismus ($gelenkmechanismus.mo$) an das Programm Independent Co-Simulation ICOS koppeln zu können, wird ein Interface für die Anbindung an die Co-Simulation in den Modellcode eingebunden, welche das Modell in keinster Weise beeinflusst. Es dient ausschließlich zum Datenaustausch. [20]
Als Eingangs- oder Koppelsignale des OM-Modelles werden zwei Sinusschwingungen aufgeschaltet.

$$u_1(t) = 1.3sin(7t - 0.3\pi), u_2(t) = 1.3cos(7t - 0.3\pi)$$

Diese werden in einm MATLAB-Modell generiert, welches in Grafik 6.2 abgebildet ist.

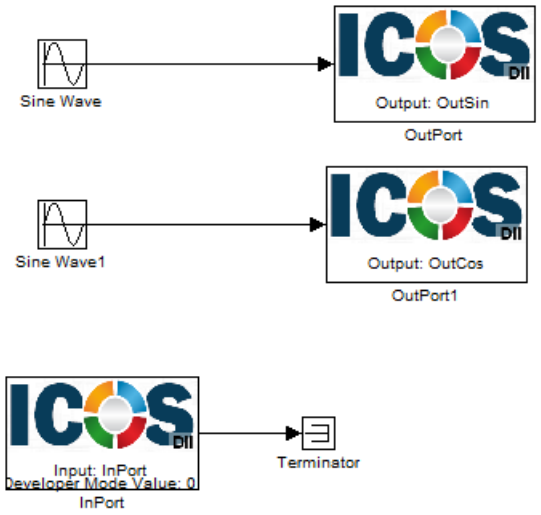


Abbildung 6.2: MATLAB-Modell $ICOS\_Kopplung.mdl$

Der zeitliche Verlauf der Koppelsignale ist in Abbildung 6.3 zu sehen.

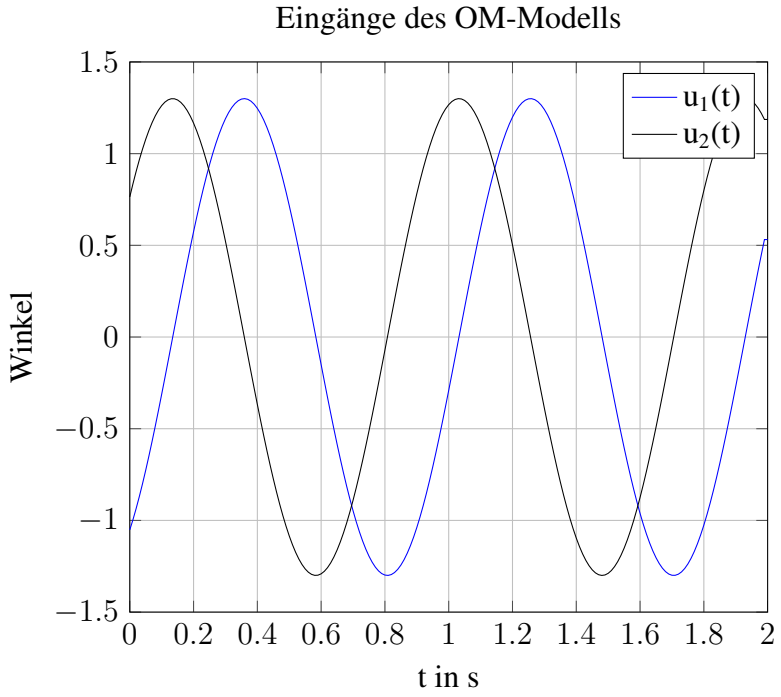


Abbildung 6.3: Kontinuierliche Eingangssignale des OM-Modells

Die Kopplung des MATLAB- und OM-Modelles in ICOS ist in Abbildung 6.4 dargestellt.

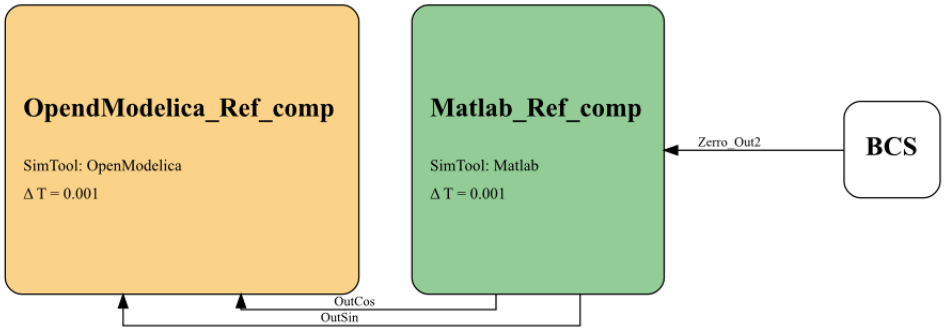
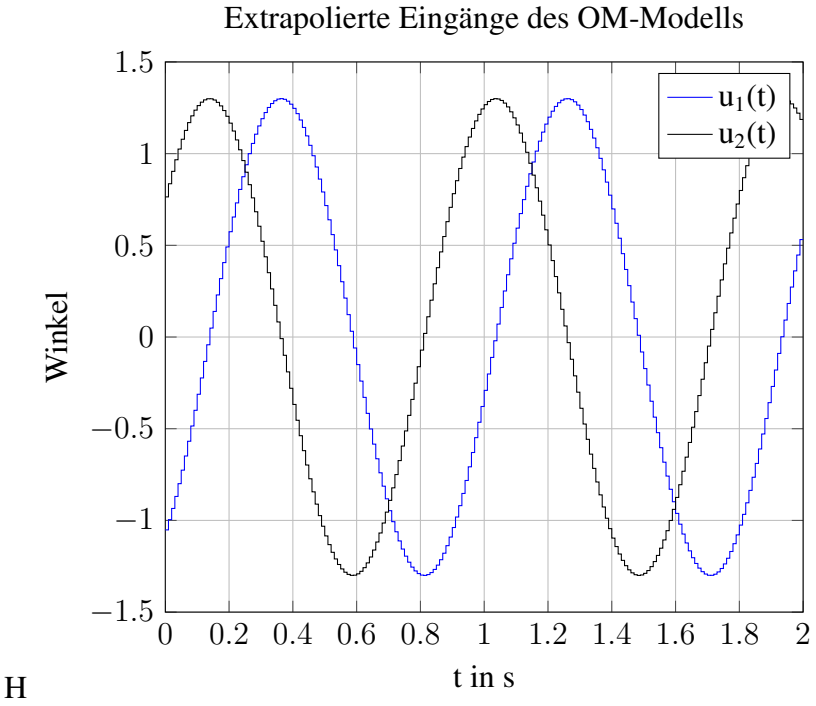


Abbildung 6.4: Kopplung der Modelle des Abschlussbeispiels in ICOS

Wird in ICOS das Matlab-Modell $ICOS\_Kopplung.mdl$ sequentiell vor dem OM-Modell ausgeführt, so werden die Eingangssignale, wie aus Grafik 6.3, an das OM-Modell übergeben und es resultiert die Referenzlösung, die für weitere Vergleiche herangezogen wird. Das OM-Teilsystem wurde mit dem DASSL-Verfahren und einer Fehlertoleranz von $1e^3$ gelöst.

Eine Extrapolation der Koppelgrößen wird hier durch die Ausführungsreihenfolge beider Modelle erzwungen. Durch die sequentielle Berechnung des OM-Modelles vor dem MATLAB-Modell, werden die Koppelgrößen für die Dauer von $\Delta T = 0.01s$ extrapoliert (Extrapolation nullter



H

Abbildung 6.5: Extrapoliertes Eingangssignale des OM-Modells

Ordnung). Für die Schätzung wird das Eingangssignal abgetastet und der Wert zum Koppelzeitpunkt über den aktuellen Makroschritt konstant gehalten. Um eine geeignete Schätzung der Eingangsgrößen zu erhalten, werden die Startwerte der beiden Sinusschwingungen in ICOS festgelegt.
In Abbildung 6.5 sind die extrapolierten (Extrapolation nullter Ordnung) Koppelgrößen dargestellt.
Um zu zeigen, dass die Extrapolation des Einganges Auswirkungen auf die Lösung des Modelles (6.1) hat, wurden einige Versuche durchgeführt. Das DGL-System mit treppenförmigen Eingangssignalen wurde mit dem expliziten und impliziten Eulerverfahren, sowie mit dem DASSL-Verfahren, und einer Fehlertoleranz von $1e^{-3}$ gelöst und einer Referenzlösung gegenübergestellt. Dabei arbeiten die Euler- sowie Runge-Kutta-Algorithmen mit fester Schrittweite. Diese Schrittweite orientiert sich an ΔT und der Gesamtsimulationsdauer T. Die berechneten zeitlichen Verläufe der Gelenkwinkel q_1 und q_2 wurden in den Grafiken 6.6 und 6.7 veranschaulicht.

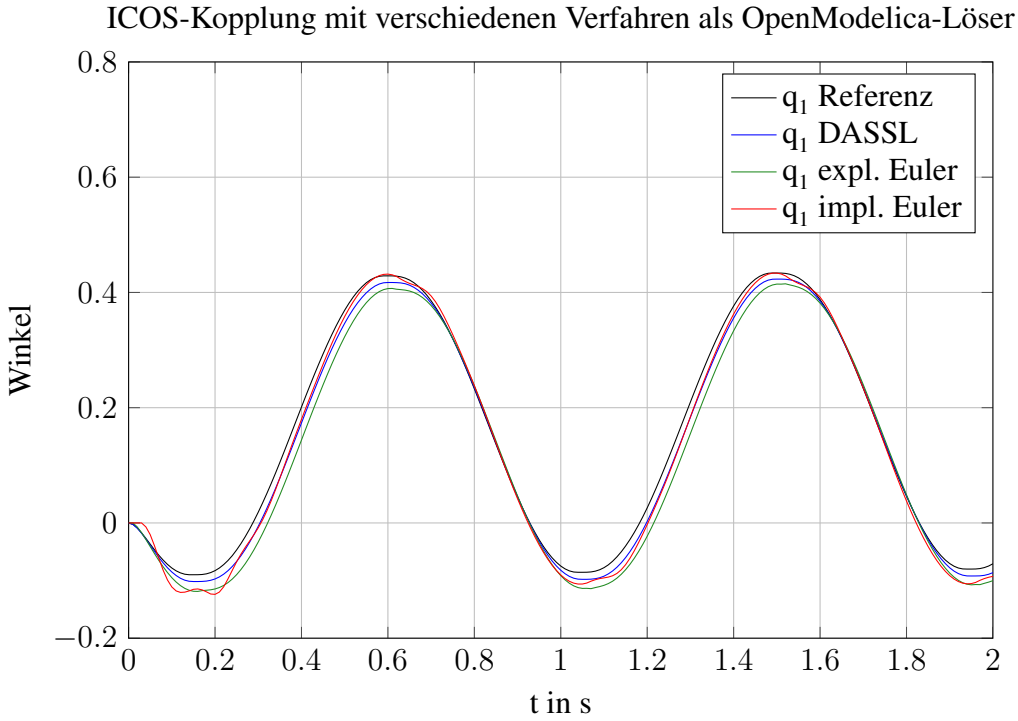


Abbildung 6.6: Berechneter Verlauf von q_1 mit diversen numerischen Integrationsverfahren

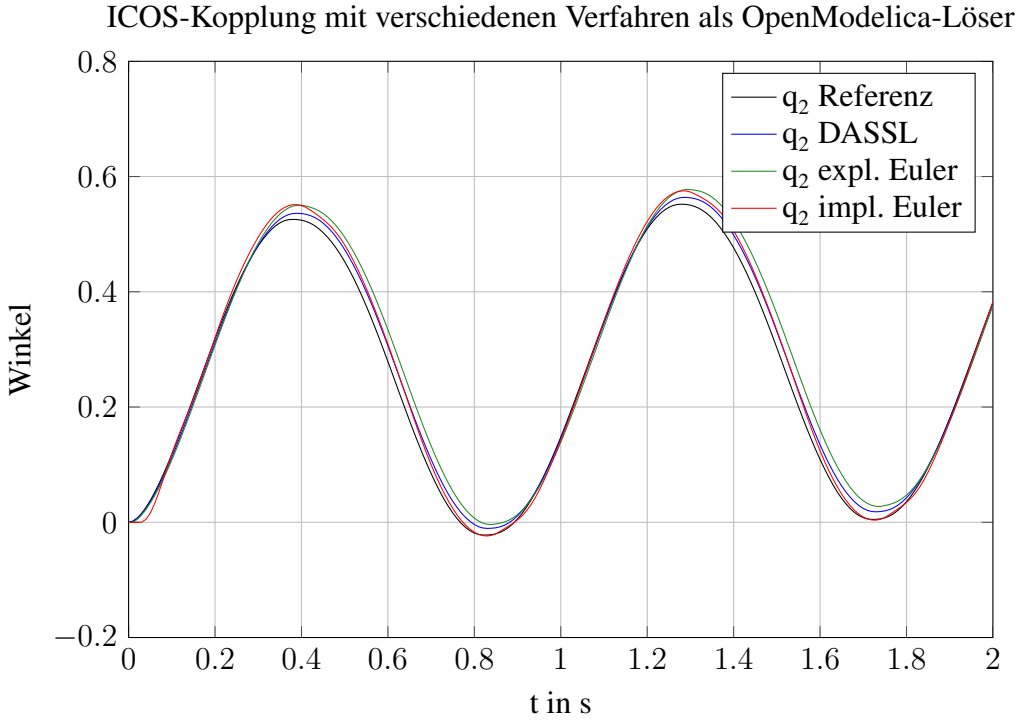


Abbildung 6.7: Berechneter Verlauf von q_2 mit diversen numerischen Integrationsverfahren

Man erkennt, dass die Kopplung der Systeme eine wesentliche Auswirkung auf den Lösungsverlauf besitzt. Die Wahl des numerischen Integrationsverfahrens ist entscheidend. Wie erwartet weichen die berechneten Ergebnisse der Eulerverfahren weiter von der Referenzlösung ab als jene mit dem DASSL-Verfahren.

6.2.1 Auswirkungen der Koppelsignale auf die Lösung

Für die weiteren Betrachtungen wurde als Löser das DASSL-Verfahren und als Simulationszeit $T = 0.5s$ gewählt. Für die folgenden Tests wurden die extrapolierten Eingangsignale teilweise modifiziert. Die Co-Simulationsplattform ICOS stellt Verfahren zur Verfügung, um den durch die Extrapolation entstandenen Schätzfehler zu reduzieren.Es existiert ein Verfahren mit der Bezeichnung Nearly Energy-Preserving Coupling Element (NEPCE), welches dazu dient den Extrapolationsfehler durch Ausnützen der Systemdynamik zu kompensieren. Eine Erweiterung dieses Verfahrens beinhaltet die Anwendung von Glättungsfiltern (erster und zweiter Ordnung) in der Kopplung ohne eine zusätzliche zeitliche Verschiebung der Koppelsignale zu bewirken. Diese Verfahren werden in [3] beschrieben.

Simulationsergebnisse

In Abbildung 6.8 und 6.9 sind die Koppelsignale dargestellt. Die Referenzsignale, das mit nullter Ordnung extrapolierte Koppelsignal, das Koppelsignal mit angewendetem NEPCE und das gefilterte Koppelsignal mit einem Filer erster und zweiter Ordnung. Eine Vergrößerung des Koppelsignales aus Plot 6.10 lässt erkennen, wie sich NEPCE und die Filter auf das extrapolierte Signal auswirken. Durch das Koppelelement NEPCE wird der treppenförmige Verlauf, welcher sich durch die Extrapolation ergibt, angehoben bzw. abgesenkt, um den Schätzfehler zu reduzieren. Anschließend an diese Maßnahme werden die Glättungsfilter erster und zweiter Ordnung angewendet. Es ist ersichtlich, dass sich das mit dem Filter zweiter Ordnung geglättete Signal, dem exakten Signalverlauf annähert. Auch die durch die Glättungsfilter hervorgerufene zeitliche Verschiebung des Koppelsignals wird kompensiert.

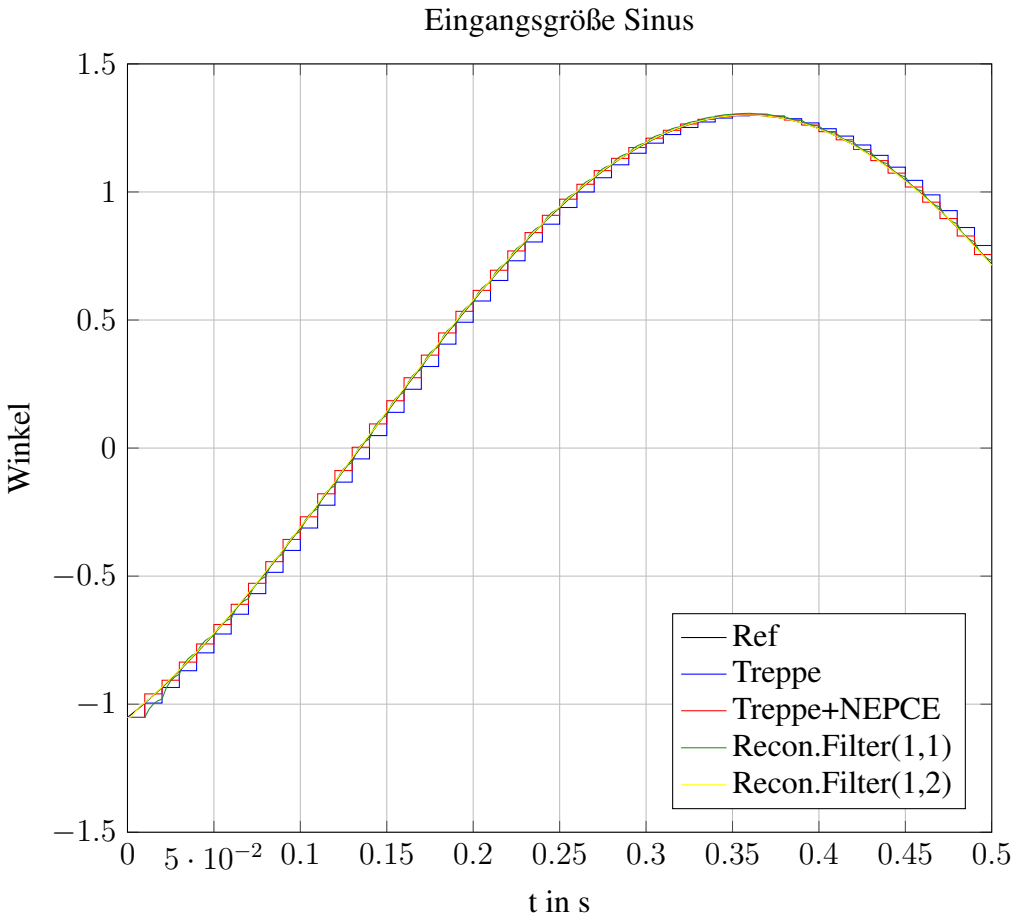


Abbildung 6.8: Koppelsignal für Winkel q_1

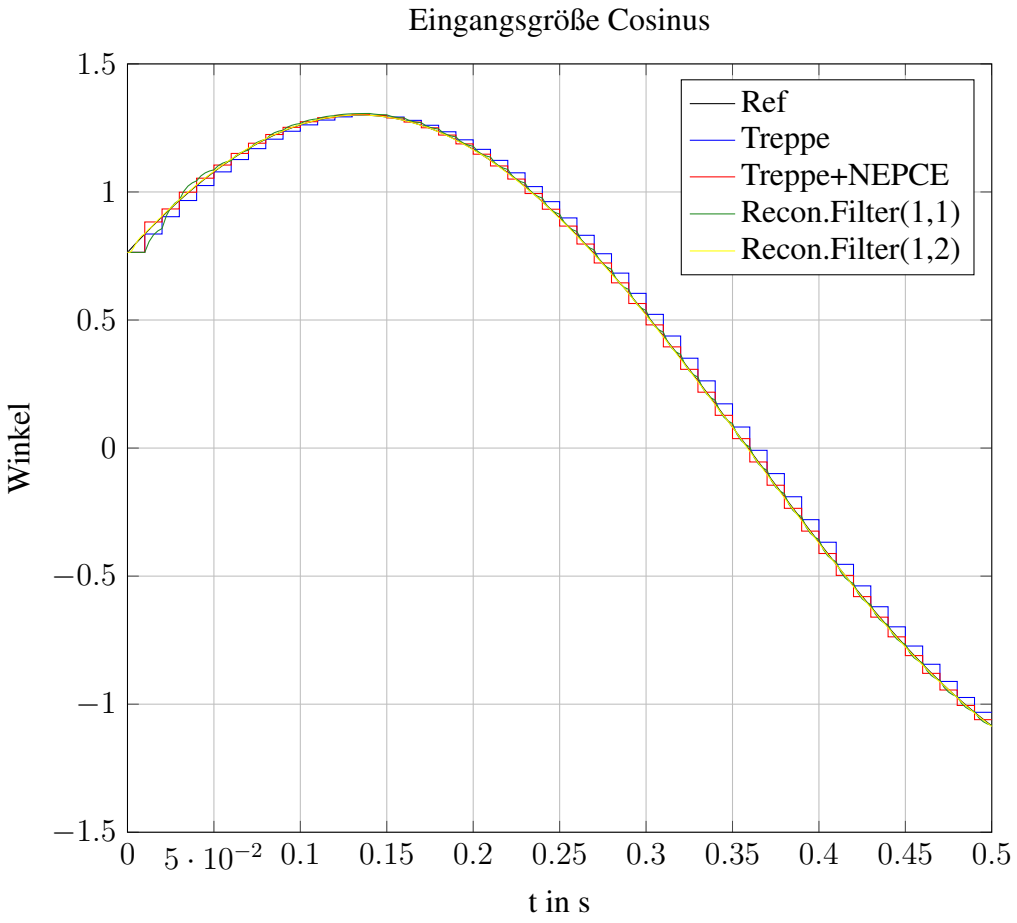


Abbildung 6.9: Koppelsignal für Winkel q_2

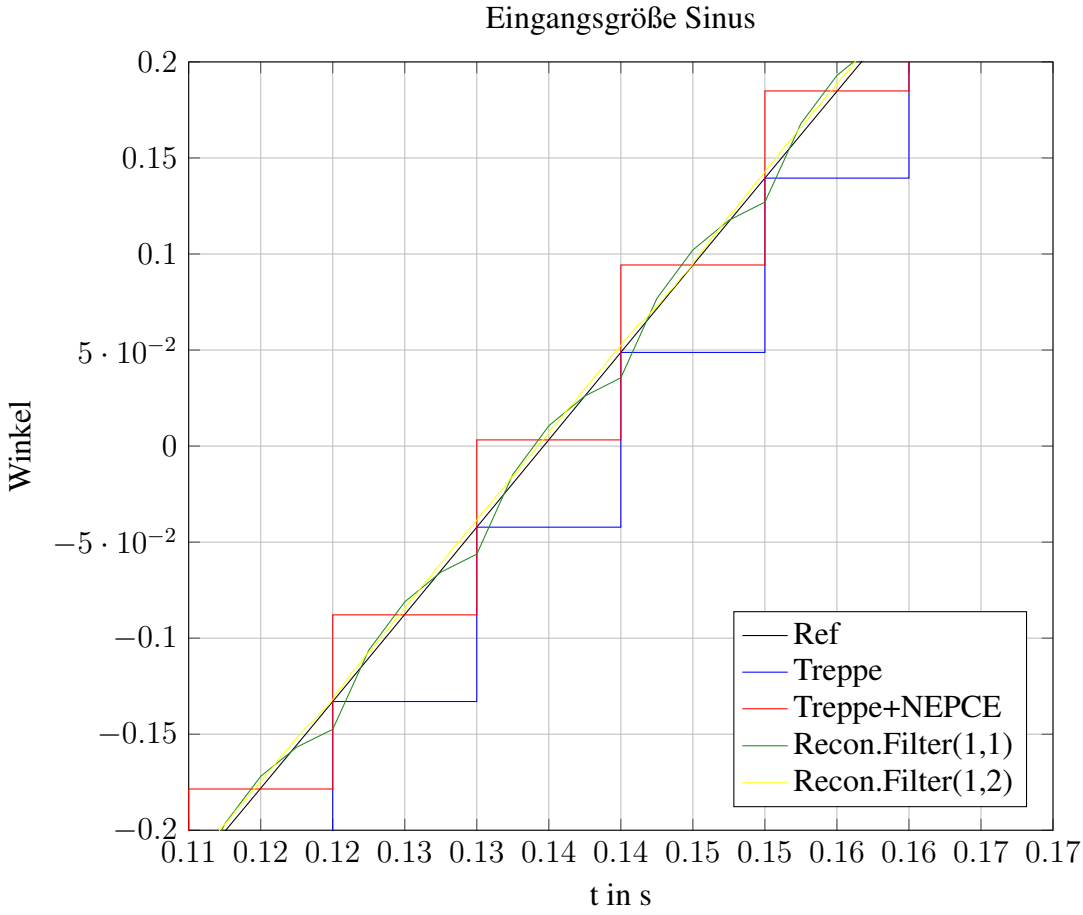


Abbildung 6.10: Vergrößerte Darstellung der Koppelsignale

Aufgrund dieser Koppelsignale wurde das Gelenkmechanismussystem gelöst und folgende Simulationsergebnisse in Grafik 6.11 und 6.12 wurden berechnet.

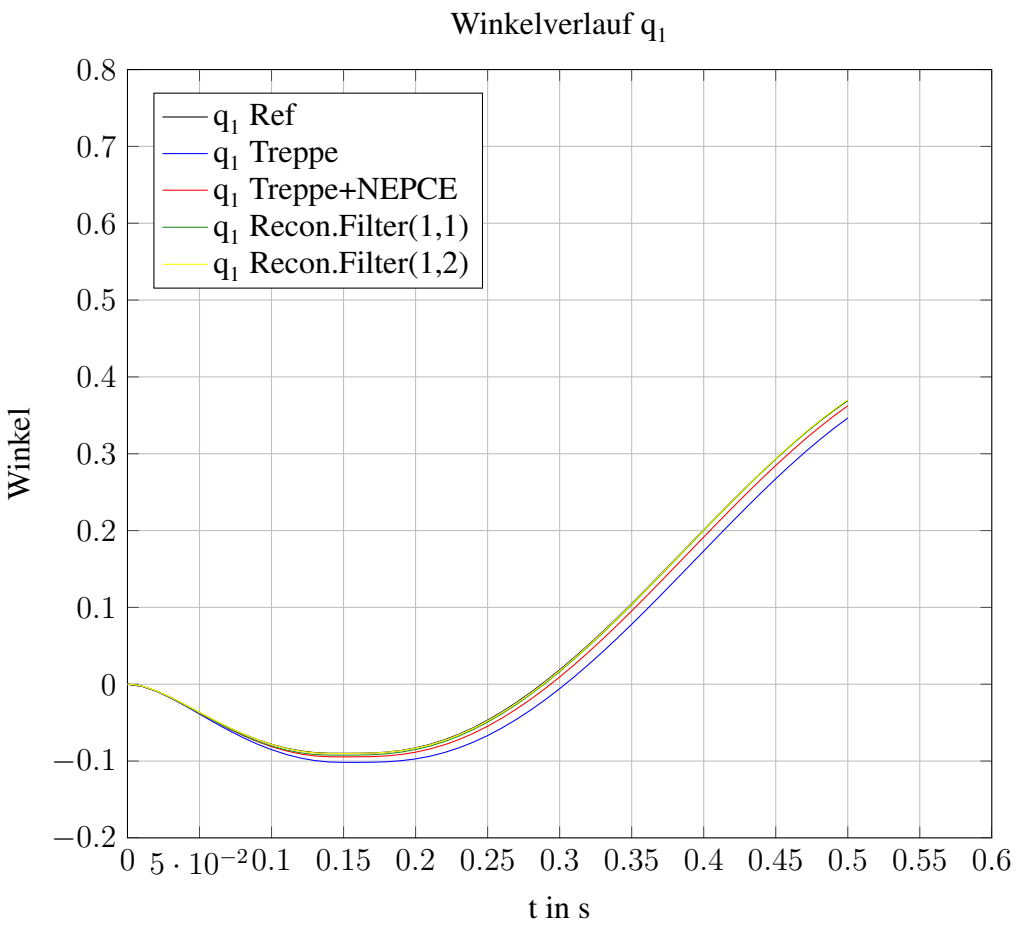


Abbildung 6.11: Ergebnis für Co-Simulation des Winkels q_1

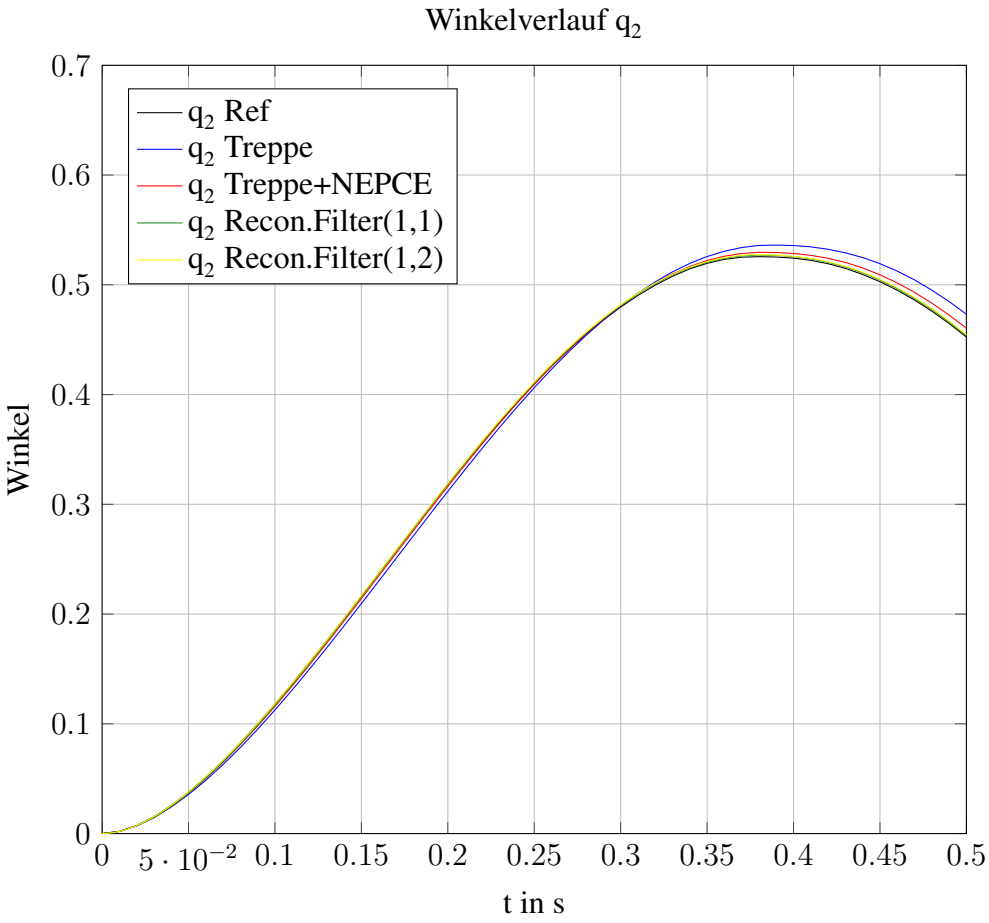


Abbildung 6.12: Ergebnis für Co-Simulation des Winkels q_2

6.3 Ergebnisse

Die Ergebnisse wurde mit eingeführten Fehlermaßen wie in Kapitel 5.2 ausgewertet und in der Tabelle 6.1 aufgelistet.

Tabelle 6.1: Ergebnisse der Co-Simulation

Eingang OM Modell in ICOS	Zustandsvariable	E_{sqmean}	E_{max}
extrapoliert	q_1	3,6747e-04	0,0278
	q_2	8,1226e-05	0,0208
extrapoliert + NPCE	q_1	4,7535e-05	0,0097
	q_2	1,1696e-05	0,0081
extrapoliert + NPCE + Recon. Filter (1,1)	q_1	3,0930e-06	0,0026
	q_2	1,1718e-06	0,0016
extrapoliert + NPCE + Recon. Filter (1,2)	q_1	2,7860e-07	0,0011
	q_2	1,8274e-06	0,0023

Die Extrapolation nullter Ordnung des Koppelsignals verursacht wie erwartet die größten Fehlerwerte. Durch NEPCE werden die Fehler verringert, jedoch beinhaltet dieses Signal noch Unstetigkeiten. Durch die Glättung in Kombination mit NEPCE wird es sichtbar verbessert. Aus Tabelle 6.1 geht hervor, dass der Tiefpassfilter erster Ordnung angewendet auf die Koppelgröße, bessere numerische Werte für die Lösung liefert, als in jenem Fall, bei dem das Eingangssignal mit einem NEPCE allein korrigiert wurde. Aufgrund der Dämpfung des Filters wurden die Unstetigkeiten verringert. Die besten Werte liefert das Modell, dessen Koppelgröße mit dem Rekonstruktionsfilter zweiter Ordnung korrigiert wurde, da es ein glattes Signal liefert, das sehr nahe an der Referenzgröße verläuft. Es wurde gezeigt, dass sich Unstetigkeiten in den Koppelgrößen negativ auf die numerische Lösung der Teilsysteme auswirken.

Kapitel 7

Zusammenfassung

In dieser Arbeit wurde der Einfluss von Unstetigkeiten in der Koppelgröße auf die numerischen Lösung der unterlagerten Systemgleichungen untersucht. In der theoretischen Betrachtung ging man auf den lokalen Diskretisierungsfehlers der jeweiligen Lösungsalgorithmen ein. Dadurch, dass jedes betrachtete ESV Verfahren sowie einer Taylorreihenentwicklung gleichzusetzen ist, kann der Einfluss der Eingangsgröße auf ein numerisches Verfahren formuliert werden. Es konnte festgestellt werden, dass die zeitliche Ableitung der Eingangsgröße, je nach Verfahren, sowohl in den Lösungsalgorithmus als auch in den lokalen Diskretisierungsfehler einfließt. Dies ließ erkennen, dass je höher die Ordnung eines numerischen Integrationsverfahrens ist, desto mehr Einfluss nehmen die Ableitungen von $u(t)$ auf das Verfahren und verringern so den Fehler.
Um diese Aussagen zu bestätigen, wurden einfache Testsysteme eingeführt. Anhand dieser Systeme wurde eine Fehlerbewertung durchgeführt. Es wurde einmal für ein glattes und für das mit nullter Ordnung extrapolierte Eingangsignal die numerische Lösung der Systemgleichungen ermittelt. Dafür wurden numerische Integrationsverfahren verschiedener Ordnung und Implementierung verwendet, wie eben mit oder ohne Schrittweitensteuerung. Für den Vergleich wurde auch jeweils die analytische Lösung berechnet. Diese Ergebnisse wurden durch die eingeführten Fehlermaße bewertet und tabelarisch dargestellt. Es wurde festgestellt, dass Verfahren mit konstanter Schrittweite generell schlechtere Werte liefern, außer man wählt sehr kleine Schrittweiten. Weiters ergeben sich durch Unstetigkeiten in der Eingangsfunktion höhere Fehlerwerte, als bei einem glatten Verlauf. Dabei spielt die Ordnung des Algorithmus eine wesentliche Rolle: Je höher die Ordnung, desto geringer der Fehler. Wählt man für die treppenförmige Eingangsgröße einen Algorithmus mit variabler Schrittweite, so erhöht sich die Anzahl der Berechnungsschritte, da in der Umgebung um die Unstetigkeiten die Schrittweite verkleinert werden muss, um die eingestellte relativen Fehlertoleranz des Verfahrens zu unterschreiten. Für steife Systeme ist es ratsam implizite Lösungsalgorithmen einzusetzen, aufgrund der A-Stabilität dieser. Ein explizites Verfahren mit Schrittweitensteuerung angewendet auf ein steifes System braucht immer sehr viele Berechnungsschritte, egal ob glattes oder treppenförmiges Eingangssignal verwendet wird. Durch die Betrachtungen der beiden DAE-Systeme vom Index 1, die gelöst das selbe Ergebnis liefern, konnte gezeigt werden, dass die Fehlerbewertung die selben Werte liefern. Somit hat das Auftreten der Eingangsgröße in der NB betraglich betrachtet keinen Einfluss auf das Lösungsverfahren. Jedoch kann es dazu führen, dass implementierte Löser den Vorgang abbrechen, da die relative Fehlerschranke nicht unterschritten werden kann. Besonders groß ist diese relative Fehlertoleranz bei Unstetigkeiten in der Eingangsgröße zu wählen.

Numerische Integrationsverfahren höherer Ordnung liefern für schnelle Signaländerungen bessere Ergebnisse als Verfahren niedrigerer Ordnung.
Abschließend wurde am Co-Simulationsbeispiel gezeigt, dass durch die in ICOS vorhandenen Methoden zur Reduzierung des Extrapolationsfehlers, die numerische Lösung an die exakten Ergebnisse angenähert werden kann. Durch geeignete Glättung der Koppelgrößen werden somit die Fehlerterme minimiert.
Wie in dieser Arbeit gezeigt wurde, ist eine Dämpfung der Unstetigkeiten in den Koppelgrößen notwendig, um genaue numerische Ergebnisse der unterlagerten Systemgleichungen zu erhalten. Bessere numerische Ergebnisse konnte schon durch ein NEPCE erzielt werden, das den Effekt des Extrapolationsfehlers stark reduziert. Eine wesentliche Verbesserung liefert ein Tiefpassfilter erster Ordnung, wie er in ICOS implementiert wurde. Besonders ratsam ist dies für DAE-Systeme, bei denen die Koppelgrößen in den NB vorkommen.

Anhang A

Berechnungen

A.1 Berechnung der Transitionsmatrix

Die Transitionsmatrix $\Phi(t)$ wurde in MATLAB berechnet.

```
function [phi] = statetrans(A)
    t = sym('t');
    phi = expm(A * t);
end
```

```
Phi_ = statetrans(A);
```

A.2 Fehlerberechnung

Ausgehend von (4.22) wird der lokale Diskretisierungsfehler für beide DAE Systeme berechnet. Somit ergibt sich für den lokalen Diskretisierungsfehler:

$$\begin{aligned} \bar{\varepsilon}_{d,lokal} &= \left(\mathbf{A}\left(\mathbf{A}\mathbf{x}+\mathbf{b}u+\mathbf{F}z\right)+\mathbf{b}\dot{u}+\mathbf{F}\dot{z}\right)\frac{h^2}{2} \\ &= \left(\mathbf{A}^2\mathbf{x}+\mathbf{A}\mathbf{b}u+\mathbf{A}\mathbf{F}z+\mathbf{b}\dot{u}+\mathbf{F}\dot{z}\right)\frac{h^2}{2} \end{aligned} \tag{A.1}$$

A.2.1 Fehlerberechnung DAE-System mit u(t) in Nebenbedingung

$$\begin{aligned}
\frac{\partial \mathbf{f_2}}{\partial \mathbf{x}} &= \begin{pmatrix} -0.25 & 0.5 \\ 0.125 & -0.5 \end{pmatrix} =: \mathbf{A}_4 \\
\frac{\partial \mathbf{f}_1}{\partial u} &= \begin{pmatrix} 1.5 \\ 2 \end{pmatrix} =: \mathbf{b}_2 \\
\frac{\partial \mathbf{f}_1}{\partial z} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} =: \mathbf{F}_1
\end{aligned} \tag{A.2}$$

Die Ableitung der NB wurde wie folgt berechnet:

$$z = -\frac{1}{4}x_1 - 2u \qquad /\frac{d}{dt} \tag{A.3}$$

$$\dot{z} = \frac{1}{16}x_1 - \frac{1}{8}x_2 - \frac{1}{4}z - \frac{3}{8}u - 2\dot{u} \tag{A.4}$$

Das DAE System und $\dot{z}$ aus (A.4) eingesetzt und zusammengefasst ergibt für die erste Zeile:

$$\bar{\varepsilon}_{1,d,lokal} = \frac{3}{16}x_1 - \frac{1}{2}x_2 - \frac{1}{4}u - \frac{1}{2}z - \frac{1}{2}\dot{u} \tag{A.5}$$

Setzt man für z aus (A.3) erhält man abschließend:

$$\bar{\varepsilon}_{1,d,lokal} = \frac{5}{16}x_1 - \frac{1}{2}x_2 + \frac{3}{4}u - \frac{1}{2}\dot{u} \tag{A.6}$$

A.2.2 Fehlerberechnung DAE-System ohne u(t) in Nebenbedingung

$$\begin{aligned}
\frac{\partial \mathbf{f_2}}{\partial \mathbf{x}} &= \begin{pmatrix} -0.25 & 0.5 \\ 0.125 & -0.5 \end{pmatrix} =: \mathbf{A}_4 \\
\frac{\partial \mathbf{f}_2}{\partial u} &= \begin{pmatrix} -0.5 \\ 1 \end{pmatrix} =: \mathbf{b}_1 \\
\frac{\partial \mathbf{f}_2}{\partial z} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} =: \mathbf{F}_2
\end{aligned} \tag{A.7}$$

Die Ableitung der NB wurde wie folgt berechnet:

$$z = -\frac{1}{4}x_1 - z \qquad /\frac{d}{dt} \tag{A.8}$$

$$\dot{z} = \frac{1}{16}x_1 - \frac{1}{8}x_2 - \frac{1}{4}z - \frac{3}{8}u \tag{A.9}$$

Das DAE System und $\dot{z}$ aus (A.9) eingesetzt und zusammengefasst ergibt für die erste Zeile:

$$\bar{\varepsilon}_{2,d,lokal} = \frac{3}{16}x_1 - \frac{1}{2}x_2 + \frac{3}{4}u - \frac{1}{2}z - \frac{1}{2}\dot{u} \tag{A.10}$$

Setzt man für z aus (A.8) erhält man abschließend:

$$\bar{\varepsilon}_{2,d,lokal} = \frac{5}{16}x_1 - \frac{1}{2}x_2 + \frac{3}{4}u - \frac{1}{2}\dot{u} \tag{A.11}$$

Anhang B

Gelenkmechanismus in OpenModelica

```
1  model gelenkmechanismus
2    Modelica.Blocks.Interfaces.RealInput u[2] "Connector 1 of Real
        input signals" annotation(Placement(visible = true,
       transformation(origin = {-120,60}, extent = {{-20,-20},{20,20}
       }, rotation = 0)));
3    Modelica.Blocks.Interfaces.RealOutput q_[2] "Connector 1 of Real
        input signals" annotation(Placement(visible = true,
       transformation(origin = {120,60}, extent = {{-20,-20},{20,20}}
       , rotation = 0)));
4    Modelica.Blocks.Interfaces.RealOutput qd_[2] "Connector 1 of Real
         input signals" annotation(Placement(visible = true,
       transformation(origin = {120,0}, extent = {{-20,-20},{20,20}},
        rotation = 0)));
5    Real q[2,1](start = [0;0]);
6    Real q_d[2,1](start = [0;0]);
7    Real q_dd[2,1](start = [0;0]);
8  protected
9    constant Real km = 0.00767;
10   constant Real kg = 70;
11   constant Real ke = 5;
12   constant Real Rm = 2.6;
13   constant Real E[2,2] = [1,0;0,1];
14   //constant Real Jges_M[2,2];
15   constant Real J1 = 1.0 * 10 ^ (-7);
16   constant Real J2 = 2.27 * 10 ^ (-5);
17   constant Real J3 = 1.4 * 10 ^ (-6);
18   constant Real J4 = 1.4 * 10 ^ (-6);
19   constant Real Jm = 3.87 * 10 ^ (-7);
20   constant Real K1 = (km * kg) ^ 2 / Rm;
21   constant Real K2 = km * kg / Rm;
22   constant Real Jges = kg ^ 2 * Jm + ke ^ 2 * J1 + J2 + J3 + J4;
23   constant Real Jges_M[2,2] = Jges * E;
```

```
24    Real beta[4,1];
25    Real Psi[4,4];
26    Real Rho[4,2];
27    Real beta_d[4,1];
28    Real Psi_d[4,4];
29    Real Rho_d[4,2];
30    Real M_strich[4,4];
31    Real M[2,2];
32    Real C_strich[4,4];
33    Real C[2,2];
34    Real Tr[2,1];
35    // Berechnung beta
36    Real A;
37    Real B;
38    Real C_;
39    Real tempsin1;
40    Real tempsin2;
41    Real tempcos1;
42    Real tempcos2;
43    //Berechnung Psi
44    Real psi11;
45    Real psi12;
46    Real psi21;
47    Real psi22;
48    // Berechnung Rho
49    constant Real I[4,2] = [0,0;0,0;1,0;0,1];
50    // Berechnung beta_d
51    // Berechnug Psi_d
52    Real psi11d;
53    Real psi12d;
54    Real psi13d;
55    Real psi14d;
56    Real psi21d;
57    Real psi22d;
58    Real psi23d;
59    Real psi24d;
60    // Berechnug M-strich
61    constant Real m = 0.065;
62    constant Real l = 0.127;
63    constant Real Js = 3.49 * 10 ^ (-4);
64    Real m11;
65    Real m22;
66    Real m33;
67    Real m13;
68    Real m24;
69    //Berechnung C_strich
70    Real h1;
71    Real h2;
```

```
72    //Berechnung Tr
73    constant Real Tc1 = 0.036;
74    constant Real Tc2 = 0.022;
75    constant Real dv = 0.0064;
76  algorithm
77    //  beta := calcBeta(q);
78    q_[1]:=q[1,1];
79    q_[2]:=q[2,1];
80    qd_[1]:=der(q[1,1]);
81    qd_[2]:=der(q[2,1]);
82    beta[1,1]:=q[1,1];
83    beta[2,1]:=q[2,1];
84    A:=2 * (cos(q[1,1]) - sin(q[2,1]));
85    B:=2 * (sin(q[1,1]) + cos(q[2,1]) - 2);
86    C_:=(cos(q[1,1]) - sin(q[2,1])) ^ 2 + (sin(q[1,1]) + cos(q[2,1])
        - 2) ^ 2;
87    tempsin1:=((-B * C_) - A * sqrt(A ^ 2 + B ^ 2 - C_ ^ 2)) / (A ^ 2
        + B ^ 2);
88    tempsin2:=((-B * C_) + A * sqrt(A ^ 2 + B ^ 2 - C_ ^ 2)) / (A ^ 2
        + B ^ 2);
89    tempcos1:=((-A * C_) + B * sqrt(A ^ 2 + B ^ 2 - C_ ^ 2)) / (A ^ 2
        + B ^ 2);
90    tempcos2:=((-A * C_) - B * sqrt(A ^ 2 + B ^ 2 - C_ ^ 2)) / (A ^ 2
        + B ^ 2);
91    beta[3,1]:=atan2(tempsin1, tempcos1) - q[1,1];
92    beta[4,1]:=atan2(A / 2 + tempcos1, (-B / 2) - tempsin1) - q[2,1];
93    //  Psi := calcPsi(beta);
94    psi11:=(-sin(beta[1,1])) - sin(beta[1,1] + beta[3,1]);
95    psi12:=(-cos(beta[2,1])) - cos(beta[2,1] + beta[4,1]);
96    psi21:=cos(beta[1,1]) + cos(beta[1,1] + beta[3,1]);
97    psi22:=(-sin(beta[2,1])) - sin(beta[2,1] + beta[4,1]);
98    Psi[1,1]:=psi11;
99    Psi[1,2]:=psi12;
100   Psi[1,3]:=-sin(beta[1,1] + beta[3,1]);
101   Psi[1,4]:=-cos(beta[2,1] + beta[4,1]);
102   Psi[2,1]:=psi21;
103   Psi[2,2]:=psi22;
104   Psi[2,3]:=cos(beta[1,1] + beta[3,1]);
105   Psi[2,4]:=-sin(beta[2,1] + beta[4,1]);
106   Psi[3,1]:=1;
107   Psi[3,2]:=0;
108   Psi[3,3]:=0;
109   Psi[3,4]:=0;
110   Psi[4,1]:=0;
111   Psi[4,2]:=1;
112   Psi[4,3]:=0;
113   Psi[4,4]:=0;
114   //  Rho := calcRho(Psi);
```

```
115   Rho:=Modelica.Math.Matrices.inv(Psi) * I;
116   //  beta_d := calcBetaDer(Rho, q_d);
117   beta_d:=Rho * der(q);
118   //  Psi_d := calcPsiDer(beta, beta_d);
119   psi11d:=(-beta_d[1,1] * cos(beta[1,1])) - (beta_d[1,1] + beta_d[
          3,1]) * cos(beta[1,1] + beta[3,1]);
120   psi12d:=beta_d[2,1] * sin(beta[2,1]) - (beta_d[2,1] + beta_d[4,1]
          ) * sin(beta[2,1] + beta[4,1]);
121   psi13d:=-(beta_d[1,1] + beta_d[3,1]) * cos(beta[1,1] + beta[3,1])
          ;
122   psi14d:=(beta_d[2,1] + beta_d[4,1]) * sin(beta[2,1] + beta[4,1]);
123   psi21d:=(-beta_d[1,1] * sin(beta[1,1])) - (beta_d[1,1] + beta_d[
          3,1]) * sin(beta[1,1] + beta[3,1]);
124   psi22d:=(-beta_d[2,1] * cos(beta[2,1])) - (beta_d[2,1] + beta_d[
          4,1]) * cos(beta[2,1] + beta[4,1]);
125   psi23d:=-(beta_d[1,1] + beta_d[3,1]) * sin(beta[1,1] + beta[3,1])
          ;
126   psi24d:=-(beta_d[2,1] + beta_d[4,1]) * cos(beta[2,1] + beta[4,1])
          ;
127   Psi_d[1,1]:=psi11d;
128   Psi_d[1,2]:=psi12d;
129   Psi_d[1,3]:=psi13d;
130   Psi_d[1,4]:=psi14d;
131   Psi_d[2,1]:=psi21d;
132   Psi_d[2,2]:=psi22d;
133   Psi_d[2,3]:=psi23d;
134   Psi_d[2,4]:=psi24d;
135   Psi_d[3,1]:=0;
136   Psi_d[3,2]:=0;
137   Psi_d[3,3]:=0;
138   Psi_d[3,4]:=0;
139   Psi_d[4,1]:=0;
140   Psi_d[4,2]:=0;
141   Psi_d[4,3]:=0;
142   Psi_d[4,4]:=0;
143   //  Rho_d := calcRhoDer(Psi, Psi_d, Rho);
144   Rho_d:=-Modelica.Math.Matrices.inv(Psi) * Psi_d * Rho;
145   //  M_strich := calcM_strich(beta);
146   m11:=m * l ^ 2 * (3 / 2 + cos(beta[3,1])) + 2 * Js;
147   m22:=m * l ^ 2 * (3 / 2 + cos(beta[4,1])) + 2 * Js;
148   m33:=1 / 4 * m * l ^ 2 + Js;
149   m13:=m * l ^ 2 * (1 / 4 + 1 / 2 * cos(beta[3,1])) + Js;
150   m24:=m * l ^ 2 * (1 / 4 + 1 / 2 * cos(beta[4,1])) + Js;
151   M_strich:=[m11,0,m13,0;0,m22,0,m24;m13,0,m33,0;0,m24,0,m33];
152   //  M := calcM(M_strich, Rho);
153   M:=transpose(Rho) * M_strich * Rho;
154   //  C_strich := calcC_strich(beta, beta_d);
155   h1:=-m * l ^ 2 / 2 * sin(beta[3,1]);
```

```
156   h2:=-m * l ^ 2 / 2 * sin(beta[4,1]);
157   C_strich:=[h1 * beta_d[3,1],0,h1 * (beta_d[1,1] + beta_d[3,1])
          ,0;0,h2 * beta_d[4,1],0,h2 * (beta_d[2,1] + beta_d[4,1]);-h1 *
          beta_d[1,1],0,0,0;0,h2 * beta_d[2,1],0,0];
158   //  C := calcC(M_strich, C_strich, Rho, Rho_d);
159   C:=transpose(Rho) * C_strich * Rho + transpose(Rho) * M_strich *
          Rho_d;
160   //  Tr := calcTr(beta_d);
161   Tr[1,1]:=Tc1 * sgn(beta_d[1,1]) + dv * beta_d[1,1];
162   Tr[2,1]:=Tc2 * sgn(beta_d[2,1]) + dv * beta_d[2,1];
163 equation
164   q_d = der(q);
165   q_dd = der(q_d);
166   q_dd = Modelica.Math.Matrices.inv(M + Jges_M) * (K2 * [u[1];u[2]]
          - C * q_d - K1 * q_d - Tr);
167   //Grafik
168   annotation(Icon(coordinateSystem(extent = {{-100,-100},{100,100}}
          , preserveAspectRatio = true, initialScale = 0.1, grid = {2,2}
          ), graphics = {Text(origin = {-1.0254,7.17347}, extent = {{-54
          .03,11.57},{54.03,-11.57}}, textString = "Gelenkmechansimus"),
          Rectangle(origin = {0.292826,-0.292826}, extent = {{-99.8536
          ,99.8536},{99.8536,-99.8536}})}), experiment(StartTime = 0,
          StopTime = 2, Tolerance = 0.001, Interval = 0.004));
169 end gelenkmechanismus;
```

Listing B.1: OM Modell

Literaturverzeichnis

[1] M. Andres und T. Schmitt. *Modellbildung und Simulation Mechatronischer Systeme, Skriptum*. Fachhochschule Vorarlberg, Masterstudiengang Mechatronik, 2011 (siehe Seite 38).

[2] M. Arnold. *Simulation Algorithms in Vehicle System Dynamics*. Technischer Bericht. Martin-Luther-University Halle, Department of Mathematics und Computer Science, 2004 (siehe Seite 2).

[3] M. Benedikt. *Eine Kopplungsmethode für die nicht-iterative Co-Simulation, Dissertation*. Kompetenzzentrum - Das Virtuelle Fahrzeug mbh Graz, 2012 (siehe Seiten 1, 2, 4, 5, 17, 65).

[4] M. Benedikt u. a. *Moderne Kopplungsmethoden - Ist Co-Simulation beherrschbar?* In: *NAFEMS Online-Magazin, Zeitschrift für numerische Simulationsmethoden und angrenzende Gebiete* 22 (Juli 2012), Seiten 63–74 (siehe Seiten 1–3, 5).

[5] M. Benedikt u. a. *NEPCE - A Nearly Energy-Preserving Coupling Element for Weak-Coupled Problems and Co-Simulations*. In: *V International Conference on Computational Methods for Coupled Problems in Science and Engineering, COUPLED PROBLEMS 2013*. Herausgegeben von M. Papadrakakis S. Idelsohn und B. Schreer. 2013 (siehe Seiten 4, 5).

[6] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. A John Wiley und Sons, Inc., Publication, 2004.

[7] A. Hofer. *Computergestütze Modellbildung und Simulation, Skriptum*. Institut für Regelungs- und Automatisierungstechnik, TU Graz, 2004 (siehe Seiten 8–11, 13, 16, 18).

[8] M. Hommel. *Parallelisierte Simulationsprozesse für virtuelles Prototyping in der Automobilindustrie, Dissertation*. Von der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität Carolo - Wilhelmina zu Braunschweig, 2006 (siehe Seite 1).

[9] M. Horn und N. Dourdoumas. *Regelungstechnik, Rechnerunterstützter Entwurf zeitkontinuierlicher und zeitdiskreter Regelkreise*. Pearson Studium, 2004 (siehe Seiten 6, 32).

[10] *ICOS*. 2014. URL: `http://www.v2c2.at/research/ee-software/co-simulation/` (siehe Seite 3).

[11] K. Janschek. *Systementwurf mechatronischer Systeme*. Springer-Verlag Berlin Heidelberg, 2010 (siehe Seiten 7–9, 11, 19–21).

[12] K. Janschek, E. Giebler und S. Dyblenko. *Simulationstechnik, Skriptum*. Technische Universität Dresden, 2006 (siehe Seite 13).

[13] S. Knorr. *Multirate-Verfahren in der Co-Simulation gekoppelter dynamischer Systeme mit Anwendung in der Fahrzeugdynamik, Masterarbeit*. Universität Ulm, Fakultät für Mathematik und Wirtschaftswissenschaften, Oktober 2002 (siehe Seite 11).

[14] R. Kübler und W. Schiehlen. *Modular Simulation in Multibody System Dynamics*. In: *Multibody System Dynamics, No. 4, pp. 107-127* (2000) (siehe Seite 2).

[15] J.D. Lambert. *Numerical methods for ordinary differential systems, the inital value problem*. John Wiley und Sons, West Sussex, U.K., 1991 (siehe Seite 13).

[16] G. Lube. *Numerische Mathematik II, Skriptum*. Institut für Numerische und Angewandte Mathematik (NAM) der Georg-August-Universität Göttingen, 2005 (siehe Seite 41).

[17] MathWorks. *Matlab-Central*. 2014. URL: `http://www.mathworks.com/matlabcentral/answers/98293-is-there-a-fixed-step-ordinary-differential-equation-ode-solver-in-matlab-8-0-r2012b` (siehe Seite 38).

[18] *MATLAB*. 2014. URL: `http://www.mathworks.de/de/help/matlab/index.html` (siehe Seiten 41, 42, 47).

[19] *Maxima, a Computer Algebra System*. 2014. URL: `http://maxima.sourceforge.net/` (siehe Seite 32).

[20] Kompetenzzentrum - Das Virtuelle Fahrzeug Forschungsgesellschaft mbH. *ICOS User Manual*. Virtual Vehicle, 2014 (siehe Seite 61).

[21] M.Braack. *Numerik für Differentialgleichungen, Skriptum*. Christian-Albrechts Universität zu Kiel, 2011 (siehe Seiten 14, 16).

[22] M. Monsberger. *Entwurf und Realisierung einer Mehrgrössenregelung für einen Gelenkmechanismus, Diplomarbeit*. 2002 (siehe Seiten 59, 60).

[23] *OpenModelica*. 2014. URL: `https://openmodelica.org/` (siehe Seite 61).

[24] F. Bornemann P. Deuflhard. *Numerische Mathematik 2, Gewöhnliche Differentialgleichungen*. Walter de Gruyter, 2002.

[25] R. Rannacher. *Numerische Mathematik 1, (Numerik gewöhnlicher Differentialgleichungen), Skriptum*. Institut für Angewandte Mathematik Universität Heidelberg, 2012 (siehe Seite 12).

[26] T. Richter. *Numerische Methoden für gewöhnliche und partielle Differentialgleichungen, Skriptum*. Universität Heidelberg, 2011 (siehe Seite 20).

[27] L. F. Shampine. *Solving 0=F(t,y(t),y'(t)) in Matlab*. In: *J. Numer. Math., Vol. 10, No. 4, pp. 291–310 (2002)* (2002) (siehe Seite 52).

[28] B. Simeon. *Numerik gewöhnlicher Differentialgleichungen, Skriptum*. TU München, Zentrum Mathematik, 2003 (siehe Seiten 10, 14, 16, 20, 41, 42).

[29] C. Tischendorf. *Numerik differential-algebraischer Gleichungen*. 2010 (siehe Seite 8).

[30] M. Trcka. *Co-simulation for Performance Prediction of Innovative Integrated Mechanical Energy Systems in Buildings, Dissertation*. Eindhoven University of Technology, 2008 (siehe Seite 12).

[31] W. Vogt. *Zur Numerik gewöhnlicher Differentialgleichungen Teil 1 Anfangswertprobleme, Skriptum*. Technische Universität Ilmenau Institut für Mathematik Postfach 100565 98684 Ilmenau, 2002 (siehe Seite 20).

[32] B. von Harrach. *Numerik Mathematik 2, Skriptum*. Universität Stuttgart, Fachbereich Mathematik - IMNG, Lehrstuhl für Optimierung und inverse Probleme, 2013 (siehe Seite 41).

[33] J. Weickert. *Mathematik fur Informatiker I, Skriptum Wintersemester 2003*. Mathematical Image Analysis Group, Universität Saarland, 2004 (siehe Seite 19).