Verifikation von Lösungen ausgewählter Probleme aus der Modellierung von Manipulatoren*

Gregor Büdding[1] Daniela Fausten[2] Walter Krämer[3] Thomas Möllers[4]
Holger Traczinski[5]

1 Einleitung

Mit Genugtuung dürfen wir diesen Bericht zur Verifikation der Lösungen ausgewählter Probleme aus der Modellierung von Manipulatoren und der Simulation ihrer Vorwärtskinematik sowie der nichtlinearen Regelung hydrostatischer Antriebssysteme vorstellen. Er ist im Anschluß an ein SFB-Seminar zur Fehlermodellierung und Fehlerfortpflanzung entstanden, das vom Teilgebiet A8 „Numerische Verifikation und Validierung kinematischer und dynamischer Modelle von Schwerlasthandhabungssystemen“ im Rahmen des Sonderforschungsbereichs 291 „Elastische Handhabungssysteme für schwere Lasten in komplexen Operationsbereichen“ veranstaltet wurde.

Als Referenten konnten wir den kommissarischen Leiter des Instituts für Wissenschaftliches Rechnen und Mathematische Modellbildung, Herrn Privatdozent Dr. Walter Krämer, von der Universität Karlsruhe (TH) gewinnen, der mit seinen grundlegenden Beiträgen zur Verifikationsnumerik diesen Bericht initiiert und ganz wesentlich zur erfolgreichen Bearbeitung der Fragestellungen beigetragen hat; dafür sei ihm an dieser Stelle ein herzliches Dankeschön gesagt.

Die behandelten Problemstellungen stammen aus den Bereichen Vorwärts- und Inverse Kinematik von Manipulatoren (Teilprojekt A1: Leitung Prof. Dr. M. Hiller, Gregor Büdding) und aus der Regelungstechnik (Teilprojekt B4: Leitung Prof. Dr. H. Schwarz, Dr. Möllers).

Die Projektbearbeiter haben die Fragestellungen zur Verfügung gestellt, unverifizierte Lösungen in der notwendigen kleinschrittigen Form vorgegeben und wesentliche Hilfestellung bei der Erstellung der Schnittstellen zu den Fehlermodellierungs-, Fehlerfortpflanzungs- und Verifikationswerkzeugen geleistet. Auch Ihnen sei an dieser Stelle herzlich für die gute Kooperation gedankt.

*Diese Arbeit wurde durch die Deutsche Forschungsgemeinschaft im Rahmen des Sonderforschungsbereichs 291 „Elastische Handhabungssysteme für schwere Lasten in komplexen Operationsbereichen“ unterstützt.

[1]Gerhard-Mercator-Universität–GH Duisburg, Mechatronik, D–47048 Duisburg, Germany;
e-mail: buedding@mechatronik.uni-duisburg.de

[2]Gerhard-Mercator-Universität–GH Duisburg, Informatik II, D–47048 Duisburg, Germany;
e-mail: fausten@informatik.uni-duisburg.de

[3]Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung (IWRMM), D–76128 Karlsruhe, Germany; e-mail: Walter.Kraemer@math.uni-karlsruhe.de

[4]Gerhard-Mercator-Universität–GH Duisburg, Meß-, Steuer- und Regelungstechnik (MSRT), D–47048 Duisburg, Germany; e-mail: moellers@mail.msrt.uni-duisburg.de

[5]Gerhard-Mercator-Universität–GH Duisburg, Informatik II, D–47048 Duisburg, Germany;
e-mail: traczinski@informatik.uni-duisburg.de

An ausgewählten wichtigen Fragestellungen aus den Aufgabenbereichen von Teilgebiet A8 (Leitung Prof. Dr. W. Luther, Eva Dyllong, Daniela Fausten, Dr. Otten, Holger Traczinski) bezogen auf die Forschungsfelder der Teilprojekte A1 und B4 wurde exemplarisch belegt:

Es werden mächtige Werkzeuge zur Verifikation von Lösungen nichtlinearer Gleichungssysteme bei der Behandlung kinematischer Schleifen, rekursiver Differenzengleichungen bei der Bearbeitung der Regelung hydraulischer Aktuatoren, zur verifizierten Lösung von Anfangswertproblemen der Dynamik mechanischer Systeme bereitgestellt.

Sie erlauben eine effiziente Fehlermodellierung und Fehlerverfolgung durch die Iterationsschritte hindurch und liefern verifizierte Lösungseinschlüsse auch bei unscharfen Eingabeparametern.

Die Verifikation von Lösungen nichtlinearer Gleichungssysteme erfolgte an einem dynamischen Modell eines einachsigen Manipulators mit einer kinematischen Schleife. Dieser Manipulator stellt ein Teilsystem des im SFB 291 verwendeten dreiachsigen Versuchsträgers HyRob dar.

Mit Hilfe des Kinematik- und Dynamikcompilers SYMKIN [8] können die Koeffizienten der Bewegungsgleichungen, die die Dynamik eines mechanischen Systems beschreiben und die gewöhnliche Differentialgleichungen zweiter Ordnung darstellen, automatisch in symbolischer Form in MATHEMATICA-Syntax [22] generiert werden. Am Beispiel des einachsigen Manipulators wurde eine Methodik zur Übernahme und Verarbeitung des erzeugten Programmcodes in Verifikationstools entwickelt.

Diese Verfahren erlauben in Rückkopplung eine Verbesserung der Modellierung und klassischen Lösungsmethodik. Sie geben Auskunft darüber, wie genau Schätzungen der Eingabeparameter erforderlich sind, um akkurate Einschließungen der Lösungen am Ausgang zu erzielen.

Besonderes Gewicht liegt auf der Schaffung von Schnittstellen zwischen den in den Teilprojekten verwendeten Softwarewerkzeugen und den Verifikationstools. Hier konnten gangbare Wege aufgezeigt werden, die aber verfeinert werden müssen.

Eine Entscheidung, inwieweit Verifikations-Arithmetiken und -Tools bzw. Validierungswerkzeuge in die einzelnen Modellierungs- und Simulationsumgebungen integriert werden können, ist in jedem einzelnen Fall zu untersuchen und zu beantworten.

Damit sind erste Antworten auf gemeinsame Fragestellungen der Projekte A1, A8 und B4 im Sinne der Arbeitspunkte Ic und II des Projekts A8 gegeben.

2 Grundlagen der Intervallarithmetik

Die in diesem Kapitel und in Kapitel 3 und Kapitel 5 vorgestellten theoretischen Grundlagen beruhen auf den Vorträgen von Herrn Dr. Krämer, die er im Rahmen des SFB-Seminars gehalten hat. Sie fassen die von ihm vorgestellten Ergebnisse zusammen, um die nachfolgend gemachten Untersuchungen, die mit Hilfe des unten beschriebenen Fehlerkalküls [**?**, 6, 2, 11, **?**] und mit dem Modul AWA [14, 15, 16, 17, 18] zur Lösung von Anfangswertproblemen gemacht wurden, besser in die Thematik einzuordnen.

In diesem einleitenden Abschnitt werden wir uns mit den Grundlagen der Intervallrechnung beschäftigen. Diese bietet dem Nutzer verschiedene Vorteile: Während beispielsweise eine Funktion bei Floating-Point-Rechnungen auf einem Computer nur an diskreten Stellen ausgewertet werden kann, und somit über das Gesamtverhalten der Funktion keine Aussage möglich ist, bietet Intervallrechnung die Möglichkeit, korrekte kontinuierliche Einschließungen einer Funktion

über kontinuierlichen Bereichen (Intervallen) anzugeben. Das Verhalten von Funktionen in den Lücken zwischen Maschinenzahlen kann dadurch sicher miterfaßt werden. Ein weiterer Vorteil, den wir bei den unten ausgeführten Beispielen besonders genutzt haben, ist, daß Eingabedaten in einem Intervall eingeschlossen werden können, so daß es möglich ist, eine Aussage über viele mögliche Eingabewerte zu treffen bzw. Meßungenauigkeiten zu berücksichtigen.

Ehe wir auf die Maschinenintervallarithmetik eingehen, benötigen wir eine Begriffsbildung der Mengenarithmetik:

Definition 1 *Seien $X, Y \subset \mathbb{R}$ zwei Mengen, $\circ_P$ eine Potenzmengenoperation. Dann ist*

$$X \circ_P Y := \{x \circ y \mid x \in X, y \in Y\}.$$

Der Nachteil hierbei ist, daß die Ergebnismenge $X \circ_P Y$ im allgemeinen nicht über ihre Definition berechenbar ist (man müsste unendlich viele Punkt-Operationen durchführen). Weiter sollten die verwendeten Mengen einfach auf einer Rechenanlage darstellbar sein. Deshalb schränken wir die zugelassenen Mengen auf Intervalle der Form

$$X = [\underline{x}, \overline{x}] := \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$$

ein. Dann gilt für eine Grundoperation $\circ \in \{+, -, \cdot, /\}$

$$\begin{aligned} X \circ_P Y &= \{x \circ y \mid x \in X, y \in Y\} \\ &= \left\{x \circ y \mid \underline{x} \leq x \leq \overline{x}, \underline{y} \leq y \leq \overline{y}\right\} \\ &= \left\{z \mid \min(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}) \leq z \leq \max(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y})\right\}. \end{aligned}$$

Mit $\underline{z} := \min(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y})$ und $\overline{z} := \max(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y})$ erhalten wir dann

$$Z = [\underline{z}, \overline{z}] = X \circ_P Y.$$

Damit sind die Grundoperationen durch maximal vier reelle Operationen, einer Minimums- und einer Maximumsbildung ausführbar. Zur Abkürzung werden wir im folgenden $X \circ Y$ schreiben anstatt $X \circ_P Y$.

Bei diesen Mengenoperationen ist folgendes zu beachten:
Im allgemeinen gilt nur

$$\begin{aligned} X \cdot X &= \{x \cdot y \mid x, y \in X\} \\ &\neq \{x \cdot x \mid x \in X\} \\ &= X^2 \end{aligned}$$

und

$$\begin{aligned} X - X &= \{x - y \mid x, y \in X\} \\ &\neq \{x - x \mid x \in X\} \\ &= [0, 0]. \end{aligned}$$

Dies zeigt z. B. , daß Auslöschungssituationen vermieden werden sollten. Sie führen in der Regel zu einer staken Überschätzung der eigentlich gesuchten Ergebisse.

Definition 2 *Seien* $X = [\underline{x}, \overline{x}]$ *und* $Y = [\underline{y}, \overline{y}]$ *reelle Intervalle. Dann heißt*

$$d(X) := \overline{x} - \underline{x}$$

Durchmesser *von* X *und*

$$q(X, Y) := \max(|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|)$$

Abstand *von* X *und* Y.

Bemerkung 3 *Die Abstandsfunktion* q *definiert eine Metrik auf der Menge der reellen Intervalle [1].*

Ist nun $f : D \to \mathbb{R}$ eine Funktion und $X \subset D$ ein Intervall, so bezeichnen wir mit

$$W_f(X) := \{f(x) \mid x \in X\}$$

den Wertebereich von f. Falls f eine der Standardfunktionen $SF := \{\sin, \cos, \exp, \ln, \ldots\}$ ist, so sind Algorithmen bekannt, welche den Wertebereich von f hochgenau einschließen [10].

Der folgende Satz beinhaltet eine fundamentale Eigenschaft der Intervallarithmetik, die Inklusionsmonotonie [1]:

Satz 4 (Inklusionsmonotonie) *Seien* X, X', Y, Y' *Intervalle. Dann gilt*

$$x \circ y \in X \circ Y \quad \textit{für alle } x \in X, y \in Y,$$

beziehungsweise

$$X \subset X', Y \subset Y' \Rightarrow X \circ Y \subset X' \circ Y'.$$

Sei $f : D \to \mathbb{R}$ *eine Funktion. Dann gilt*

$$X \subset X' \subset D \Rightarrow W_f(X) \subset W_f(X').$$

Damit ist es uns möglich, den Wertebereich $W_f(X)$ einer Funktion f, die aus den Grund- und Standardfunktionen zusammengesetzt ist, in einem Intervall einzuschließen: Sei $f(x) = A_f$, wobei A_f ein Ausdruck für $f(x)$ sei. Dann ersetzen wir alle x in A_f durch das Intervall X und alle reellen Operationen durch die entsprechende Intervalloperation. Diese sogenannte intervallmäßige Auswertung von A_f bezeichnen wir mit $f(X)$. Wenn wir dies Symbol verwenden, wollen wir im folgenden immer annehmen, daß diese Menge tatsächlich existiert (dies muß nicht immer der Fall sein). Es gilt dann

$$f(x) \in W_f(X) \subset f(X) \text{ für alle } x \in X.$$

Zu beachten ist hierbei, daß die Menge $f(X)$ vom zugrundeliegenden Ausdruck A_f abhängt.

Nach diesen allgemeinen Einführungen wollen wir uns im folgenden mit der Maschinenintervallarithmetik beschäftigen.

Statt eines reellen Intervalls betrachten wir nun Maschinenintervalle, deren Intervallgrenzen durch Gleitpunktzahlen gegeben sind. Die Menge der Maschinenintervalle bezeichnen wir mit IS, die der reellen Intervalle mit $I\mathbb{R}$. Die Maschinenintervalloperation $\diamond$ umfaßt das mit reeller Intervallrechnung berechnete Ergebnis, das heißt, die untere Grenze wird nach unten und

die obere Grenze nach oben zur jeweils nächsten Maschinenzahl gerundet. Formal lautet ihre Definition

$$X \diamondsuit Y := \bigcap \{Z \in IS \mid Z \supset X \circ Y\} \supset X \circ Y \ (\in I\!R).$$

Durch diese Setzung bleibt die Inklusionsmonotonie auch für die Maschinenintervallarithmetik erhalten. Ersetzt man insbesondere im Funktionsausdruck A_f alle Operanden durch Maschinenintervalloperanden und jedes x durch das Maschinenintervall X, so schließt das Maschinenergebnis den tatsächlichen Wertebereich $W_f(X)$ ein.

Wenn der Durchmesser $d(X)$ eines Intervalls X zu groß ist, liefert die intervallmäßige Auswertung des Wertebereichs $W_f(X)$ einer Funktion f im allgemeinen eine grobe Überschätzung des tatsächlichen Wertebereichs. Ein möglicher Ausweg aus diesem Problem ist die Zerlegung des Intervalls X in Teilintervalle X_i, $i = 1, \ldots, N$, mit

$$X = \bigcup_{i=1}^{N} X_i.$$

Dann gilt

$$W_f(X) = W_f(\bigcup_{i=1}^{N} X_i) = \bigcup_{i=1}^{N} W_f(X_i) \subset \bigcup_{i=1}^{N} f(X_i).$$

Eine weitere Verbesserung wird erzielt, wenn wir von der *Mittelwertform* ausgehen: Zunächst gilt für eine differenzierbare Funktion f nach dem Mittelwertsatz der Differentialrechnung mit $z \in X$:

$$f(x) = f(z) + f'(\xi)(x - z), \ \ \xi = \xi(x) \in X.$$

Daraus folgt

$$f(x) \in f([z, z]) + f'(X)(X - [z, z]) =: f_m(X).$$

Man kann nun die Güte der Wertebereichseinschließungen angeben (vgl. [1]):

Satz 5 *Sei f eine differenzierbare Funktion, $X = \bigcup_{i=1}^{N} X_i$ ein Intervall. Dann existieren, unabhängig von N, Konstanten c_1 und c_2, mit*

$$q_1 := q(W_f(X), \bigcup_{i=1}^{N} f(X_i)) \leq c_1 \cdot \max_{1 \leq i \leq N} d(X_i)$$

und

$$q_2 := q(W_f(X), \bigcup_{i=1}^{N} f_m(X_i)) \leq c_2 \cdot \max_{1 \leq i \leq N} (d(X_i))^2 .$$

Im Gegensatz zur nur linearen Abhängigkeit der naiven intervallmäßigen Auswrtung erhalten wir durch Verwendung der Mittelwertform eine quadratische Abhängigkeit vom maximalen Durchmesser der Teilintervalle, also bei fortwährend feiner werdender Unterteilung eine wesentlich schnellere Konvergenz der Mittelwertsform-Auswertung gegen den tatsächlichen Wertebereich. Der Nachteil dieses Vorgehens besteht zunächst darin, daß wir den Wertebereich der Ableitung der Funktion f ebenfalls benötigen. Dazu können wir aber die Methode der automatischen Differentiation [**?**, **?**] verwenden, auf die deshalb hier kurz eingegangen werden soll:

Definition 6 *Gegeben sei ein Ausdruck $A(x)$. Gesucht ist nun der Funktions- und der Ableitungswert dieses Ausdrucks an der Stelle $x = x_0$. Wir schreiben dies als Paar $(A(x_0), A'(x_0))$. Gilt nun $A(x) = u(x) \cdot v(x)$ und $U = (u_0, u_1) := (u(x_0), u'(x_0))$ und $V = (v_0, v_1) := (v(x_0), v'(x_0))$, so folgt*

$$A = (a_0, a_1) := U \cdot V = (u_0, u_1) \cdot (v_0, v_1) := (u_0 \cdot v_0, u_1 v_0 + u_0 v_1).$$

Weiter gilt mit denselben Bezeichnungen

$$\begin{aligned} U + V &= (u_0, u_1) + (v_0, v_1) := (u_0 + v_0, u_1 + v_1) \\ U - V &= (u_0, u_1) - (v_0, v_1) := (u_0 - v_0, u_1 - v_1) \\ U/V &= (u_0, u_1)/(v_0, v_1) := \left(\frac{u_0}{v_0}, \left(u_1 - \frac{u_0 \cdot v_1}{v_0}\right) \Big/ v_0\right), \ \textit{falls } v_0 \neq 0. \end{aligned}$$

Für die Verknüpfung zweier differenzierbarer Funktionen f und u gilt nun allgemein

$$f(U) = f(u_0, u_1) := (f(u_0), f'(u_0) \cdot u_1).$$

Damit gilt beispielsweise für die Wurzelfunktion

$$\sqrt{U} := \left(\sqrt{u_0}, \frac{u_1}{2 \cdot \sqrt{u_0}}\right), \ \textit{falls } u_0 > 0.$$

Weiter gelten die Spezialfälle (diese liefern in den Blättern des Ausdrucksbaumes bekannte Zahlenpaare, mit denen dann die anderen Knoten entsprechend obiger Formeln abgearbeitet werden können)

$$f(x) = c \Rightarrow F = (c, 0) \ \textit{ und } \ f(x) = x \Rightarrow F = (x_0, 1).$$

Die automatische Differentiation ist in PASCAL XSC [9] folgendermaßen umgesetzt: Man nutzt den Datentyp `DerivType`, der als Record mit den Komponenten `f` und `df` implementiert ist, und überlädt die Grundoperatoren und Standardfunktionen für diesen Datentyp [**?**]. Dabei ist wichtig, dass die Methode der automatischen Differentiation keine Verfahrensfehler einschleppt, so dass sich durch intervallmäßige Berechnungen automatisch Einschließungen der exakten Ableitungswerte ergeben.

3 Ein Kalkül für absolute und relative Fehlerabschätzungen

Im folgenden Abschnitt beschäftigen wir uns mit einem Fehlerkalkül, das für gegebene Funktionen jeweils eine Fehlerschranke angibt, und so komplexe Berechnungen verifizieren kann.

Das Ziel ist es also, zu einer Funktion f und ihrer Maschinennäherung $\tilde{f}$ eine Fehlerschranke $\varepsilon(f)$ zu finden mit

$$\left|\frac{f(x) - \tilde{f}(x)}{f(x)}\right| \leq \varepsilon(f)$$

für alle möglichen $x \in D_f$.

Da wir auf einer Rechenanlage lediglich Gleitkommazahlen zur Verfügung haben, lautet die Bedingung für die gesuchte Fehlerschranke $\Delta(f)$ folgendermaßen: Sei $x \in X$, $\tilde{x}$ die Maschinennäherung von x mit $|x - \tilde{x}| \leq \Delta(x)$. Gesucht ist $\Delta(f)$ mit

$$\left|f(x) - \tilde{f}(\tilde{x})\right| \leq \Delta(f).$$

Zunächst vereinbaren wir die folgenden Notationen: $S = S(b, l, e_{\min}, e_{\max})$ sei die Menge der Gleitkommazahlen, wobei b die Basis, l die maximale Stellenanzahl der Mantisse und $e_{\min}$ bzw. $e_{\max}$ den kleinsten bzw. größten Exponenten angibt. Weiter sei $\varepsilon := 0.5 \cdot b^{1-l}$ das Maschinenepsilon. Reelle Zahlen und Funktionen werden mit kleinen lateinischen Buchstaben bezeichnet, wie x, f, $f(x)$, während Gleitkommazahlen bzw. Maschinenfunktionen durch eine Tilde gekennzeichnet sind: $\tilde{x}$, $\tilde{f}$, $\tilde{f}(\tilde{x})$, usw. Die zugehörigen absoluten Fehlerschranken bezeichnen wir mit $\Delta(x)$, $\Delta(f)$, Intervallgrößen werden entweder mit eckigen Klammern oder mit lateinischen Großbuchstaben bezeichnet. Alle Größen können auch Vektoren darstellen. Mit der Ungleichung $|x - \tilde{x}| \leq \Delta(x)$ ist dann die Menge von Ungleichungen der Form $|x_i - \tilde{x}_i| \leq (\Delta(x))_i$, $i = 1, \ldots, n$, gemeint.

Definition 7 *Seien $a, b \in S$ Gleitkommazahlen, $\circ \in \{+, -, \cdot, /\}$, $\boxed{\circ}$ die korrespondierende Gleitkommazahloperation. Dann wollen wir zwei Voraussetzungen erfüllt haben:*

1. *Gilt $|a \circ b| \in [MinReal, MaxReal]$, so folgt*

$$\left| \frac{a \circ b - a \boxed{\circ} b}{a \circ b} \right| \leq \varepsilon,$$

   wobei ε das Maschinenepsilon bezeichne. Diese Bedingung ist eine Umformulierung der Forderung $a \boxed{\circ} b = (1 + eps)(a \circ b)$ mit $|eps| \leq \varepsilon$.

2. Underflow*: Gilt $|a \circ b| \leq MinReal$, so folgt $|a \circ b - a \boxed{\circ} b| \leq MinReal$.*

Wir geben nun für die Grundoperationen Schranken direkt an, ohne deren Herleitung zu beweisen. Für die Beweise sei z. B. auf [11] bzw [2] verwiesen. Aus Platzgründen ist es hier nicht möglich, den Fehlerkalkül ausführlich vorzuführen. Deshalb wird an dieser Stelle ausdrücklich die Arbeit [2] zitiert, die auf dem Server `iamk4515.mathematik.uni-karlsruhe.de` im Verzeichnis `/pub/iwrmm/preprints` als Postscript-Datei `prep985.ps` zur Verfügung steht. Demnächts ist geplant, auch die entsprechende Software auf dem angegebenen Server frei zugänglich abzulegen.

Schranke für die Addition: Seien $a \in A$, $b \in B$, $\tilde{a}, \tilde{b} \in S$ mit $|a - \tilde{a}| \leq \Delta(a)$ und $|b - \tilde{b}| \leq \Delta(b)$. Setze

$$\Delta(\text{add}) := \text{MinReal} + \varepsilon |A + B| + (1 + \varepsilon) \left(\Delta(a) + \Delta(b) \right).$$

Hierbei bezeichnet $|A| := \max_{a \in A} |a|$ den maximalen Absolutbetrag des Intervalls A. Dann gilt

$$\left| a + b - \tilde{a} \boxplus \tilde{b} \right| \leq \Delta(\text{add}).$$

Schranke für die Multiplikation: Seien $a \in A, b \in B, \tilde{a}, \tilde{b} \in S$ mit $|a - \tilde{a}| \leq \Delta(a)$ und $|b - \tilde{b}| \leq \Delta(b)$. Setze

$$\Delta(\text{mul}) := \text{MinReal} + |A||B|\varepsilon + (1 + \varepsilon) \left(|A|\Delta(b) + |B|\Delta(a) + \Delta(a)\Delta(b) \right).$$

Dann gilt

$$\left| a \cdot b - \tilde{a} \boxdot \tilde{b} \right| \leq \Delta(\text{mul}).$$

Schranke für die Division: Seien $a \in A$, $b \in B$, $\tilde{a}, \tilde{b} \in S$ mit $|a - \tilde{a}| \leq \Delta(a)$ und $|b - \tilde{b}| \leq \Delta(b)$. Zusätzlich gelte $\Delta(b) < 0.5 \cdot \langle B \rangle$, wobei $\langle B \rangle := \min_{b \in B} |b|$ das Betragsminimum von B sei. Setzen wir dann

$$\Delta(\text{div}) := \text{MinReal} + \frac{1}{\langle B \rangle - \Delta(b)} \cdot \left[\Delta(a) + (|A| + \Delta(a)) \cdot \left(\varepsilon + \frac{\Delta(b)}{\langle B \rangle} + 2 \cdot \left(\frac{\Delta(b)}{\langle B \rangle} \right)^2 \right) \right],$$

so gilt

$$\left| a/b - \tilde{a} \boxtimes \tilde{b} \right| \leq \Delta(\mathrm{div}).$$

Im vorgestellten Fehlerkalkül wird nun der Datentyp `BoundType` eingeführt, der als Record die beiden Komponenten `Enclosure` vom Typ `interval` für die Einschließung einer Zahl und `AbsErr` für die mitgeführte Fehlerschranke beinhaltet.

Weiter ist eine Funktion `DeltaAdd` implementiert, die abhängig von `A`, `B`, `DeltaA` und `DeltaB` eine Maschinenzahl ausgibt mit $\Delta(\mathrm{add}) \leq$ `DeltaAdd`. Für die beiden anderen Grundrechenarten wird die Fehlerschranke entsprechend ermittelt.

Nun können wir die Grundrechenarten überladen, das heißt, ihnen eine andere Interpretation zuteilen. Damit wird es dann möglich, zu analysierende Programme ohne wesentliche Änderungen im Quelltext so ablaufen zu lassen, dass zusätzlich zum Ergebnis die gewünschte Information über den maximalen Fehler berechnet wird.

Für die Multiplikation sieht die Überladung beispielsweise so aus:

```
GLOBAL OPERATOR * (x, y: BoundType) res: BoundType;
BEGIN
res.Enclosure := x.Enclosure * y.Enclosure;
res.AbsErr := DeltaMul(x.Enclosure, y.Enclosure, x.AbsErr, y.AbsErr)
END;
```

Damit gilt für jede der Grundoperationen $\circ \in \{+, -, \cdot, /\}$

$$\begin{pmatrix} [\underline{a}, \overline{a}] \\ \Delta(a) \end{pmatrix} \circ \begin{pmatrix} [\underline{b}, \overline{b}] \\ \Delta(b) \end{pmatrix} = \begin{pmatrix} [\underline{\mathrm{erg}}, \overline{\mathrm{erg}}] \\ \Delta(\mathrm{erg}) \end{pmatrix}$$

mit

$$a \circ b \in [\underline{\mathrm{erg}}, \overline{\mathrm{erg}}]$$

für alle $(a, b) \in [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$, und

$$\left| a \circ b - \tilde{a} \boxdot \tilde{b} \right| \leq \Delta(\mathrm{erg})$$

für alle $(a, b) \in [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$ mit $|a - \tilde{a}| \leq \Delta(a)$, $\left| b - \tilde{b} \right| \leq \Delta(b)$.

Wird nun eine komplexe Rechnung mit Hilfe dieses Fehlerkalküls analysiert, so besteht jeder Eingabewert vom Typ `BoundType` zu Beginn entweder aus einer korrekt darstellbaren Maschinenzahl, dann hat das Eingangsintervall die Form $[\tilde{a}, \tilde{a}]$ und der mitgeführte Fehlerwert `DeltaA` ist identisch Null, oder die eingegebene Zahl ist nicht korrekt darstellbar, dann wird dieser Wert mit Hilfe eines Maschinenintervalls eingeschlossen und der Fehler in `DeltaA` gespeichert. Bei jeder Rechnung, die nun ausgeführt wird, wird nicht nur eine Einschließung für das Ergebnis ermittelt, sondern auch die neue Fehlerschranke aus den übergebenen Werten berechnet. Am Ende der Rechnungen erhält man schließlich eine Einschließung des Gesamtergebnisses und eine absolute Fehlerschranke, die garantiert nicht überschritten wird.

Diese Vorgehensweise ermöglicht dem Nutzer, seine vorhandenen, zu verifizierenden Programme nur insofern zu ändern, daß lediglich die vorhandenen Datentypen in den Typ `BoundType` konvertiert werden müssen, die Struktur des Programms kann erhalten bleiben.

Leider werden bei dem bis jetzt beschriebenen Verfahren exakt durchführbare Operationen nicht automatisch als solche erkannt; dadurch kann die errechnete Fehlerschranke zu groß, ja sogar wertlos werden. Deshalb wollen wir Definition 7 um eine weitere Voraussetzung erweitern:

Definition 8 *Zusätzlich zu den in Definition 7 gemachten Voraussetzungen wollen wir folgendes annehmen:*

3. faithful arithmetic: *Wenn das exakte Ergebnis darstellbar ist, dann soll das Ergebnis der korrespondierenden Gleitkommazahl-Operation identisch sein. Wenn das Ergebnis nicht exakt darstellbar ist, so ist das Gleitkommazahl-Ergebnis eine der zwei Gleitkommazahlen, die dem exakten Ergebnis am nächsten sind.*

Wir wollen nun einige Sätze zusammenfassen, die eine Aussage darüber machen, wann die Subtraktion und die Multiplikation exakt berechenbar sind (vgl. [2]). Der Einfachheit halber nehmen wir hier an, dass die exakten Ergebnisse nicht im Unterlaufbereich liegen.

Satz 9 (Sterbenz) *Seien $\tilde{a}, \tilde{b} \in S$ binäre Gleitkommazahlen mit $\frac{1}{2} \leq \frac{\tilde{a}}{\tilde{b}} \leq 2$. Dann ist in faithful arithmetic*

$$\tilde{a} \boxminus \tilde{b} = \tilde{a} - \tilde{b}$$

exakt.

Satz 10 (Ferguson) *Seien $\tilde{x}, \tilde{y} \in S$ Gleitkommazahlen, für die die Mantisse $s(\tilde{x})$ am Ende $z(\tilde{x}) \geq 0$ Nullen hat, ebenso habe $s(\tilde{y})$ am Ende $z(\tilde{y})$ Nullen. Weiter gelte für den Exponenten von $\tilde{x} - \tilde{y}$ die Abschätzung*

$$e(\tilde{x} - \tilde{y}) \leq \min(e(\tilde{x}) + z(\tilde{x}), e(\tilde{y}) + z(\tilde{y})).$$

Dann ist das berechnete Ergebnis $\tilde{x} \boxminus \tilde{y}$ exakt, falls die Gleitkommazahl-Subtraktion faithful ist.

Satz 11 *Die Gleitkommazahl-Multiplikation ist fehlerfrei, falls einer der Faktoren eine Potenz der Basis ist.*

Satz 12 *Seien $\tilde{x}, \tilde{y} \in S$ Gleitkommazahlen der Länge t. Die Mantissen von $\tilde{x}$ und $\tilde{y}$ haben zusammen mindestens t Nullen am Ende. Dann ist*

$$\tilde{x} \boxdot \tilde{y} = \tilde{x} \cdot \tilde{y}$$

exakt.

Mit Hilfe dieser zusätzlichen Verbesserungen ist es dann möglich, Fehlerschranken für die Exponentialfunktion, den Logarithmus, den Sinus usw. über Hornerschemata automatisch zu generieren [6].

Nun wollen wir einen Satz angeben, der für jede differenzierbare Funktion und eine zugehörige Approximation jeweils eine Fehlerschranke für ihre Berechnung explizit angibt (vgl. [2]):

Satz 13 *Sei A ein Intervall, $f : D_f \to \mathbb{R}$ differenzierbar im Intervall $A + [-\Delta(a), \Delta(a)]$ und $\tilde{f}$ eine Approximation an f mit*

$$\begin{aligned} \left| f(\tilde{x}) - \tilde{f}(\tilde{x}) \right| &\leq |f(\tilde{x})| \cdot \varepsilon(f) \text{ für alle } \tilde{x} \in S \text{ mit } |f(\tilde{x})| \in [MinReal, MaxReal], \\ \left| f(\tilde{x}) - \tilde{f}(\tilde{x}) \right| &\leq \Delta_{Underflow}(f) \text{ für alle } \tilde{x} \in S \text{ mit } |f(\tilde{x})| \in [0, MinReal). \end{aligned}$$

Setzen wir

$$\Delta(f) := \varepsilon(f) \cdot |f(A)| + (1 + \varepsilon(f)) \cdot \Delta(a) \cdot \left|f'(A + [-\Delta(a), +\Delta(a)])\right| + \Delta_{Underflow}(f),$$

so gilt

$$\left|f(a) - \tilde{f}(\tilde{a})\right| \leq \Delta(f)$$

für alle $a \in A$, $\tilde{a} \in S$ *mit* $|a - \tilde{a}| \leq \Delta(a)$.

Im vorgestellten Fehlerkalkül sind die Standardfunktionen überladen worden, so daß neben dem Funktionswert automatisch auch der maximale Fehler ermittelt wird. Hier wird wiederum der Datentyp `BoundType` verwendet. Zusätzlich können benötigte, noch nicht überladene Funktionen mit Hilfe der Formel aus Satz 13 ebenfalls eingebunden werden.

Soll nun eine komplizierte Funktionsvorschrift analysiert werden, so wird intern ein Ausdrucksbaum erstellt, dessen Knoten Aufrufe von Grundoperationen bzw. von Standardfunktionen repräsentieren. Die Knoten werden dann der Reihe nach von den Blättern (hier sind die Werte bekannt) zur Wurzel mit den oben beschriebenen Fehlerabschätzungsmethoden abgearbeitet.

Es ist möglich, verschiedene Arithmetiken für Fehlerschranken zu benutzen:

1. Fehlerschranken $\Delta(\circ)$ für absolute Fehler: Seien $a \in A$, $b \in B$, $\tilde{a}, \tilde{b} \in S$ mit $|a - \tilde{a}| \leq \Delta(a)$ und $\left|b - \tilde{b}\right| \leq \Delta(b)$, so gilt
$$\left|a \circ b - \tilde{a} \boxed{\circ} \tilde{b}\right| \leq \Delta(\circ).$$

2. Fehlerfaktoren $k(\circ)$: Seien $a \in A$, $b \in B$, $\tilde{a}, \tilde{b} \in S$ mit $|a - \tilde{a}| \leq k(a) \cdot \varepsilon$ und $\left|b - \tilde{b}\right| \leq k(b) \cdot \varepsilon$, so gilt
$$\left|a \circ b - \tilde{a} \boxed{\circ} \tilde{b}\right| \leq k(\circ) \cdot \varepsilon.$$

3. Fehlerschranken $\varepsilon_\circ$ für relative Fehler: Seien $a \in A$, $b \in B$, $\tilde{a}, \tilde{b} \in S$ mit $\tilde{a} = a \cdot (1 + \varepsilon_a)$, $|\varepsilon_a| \leq \varepsilon(a)$ und $\tilde{b} = b \cdot (1 + \varepsilon_b)$, $|\varepsilon_b| \leq \varepsilon(b)$, so gilt
$$\tilde{a} \boxed{\circ} \tilde{b} = (a \circ b) \cdot (1 + \varepsilon_\circ)$$
mit $|\varepsilon_\circ| \leq \varepsilon(\circ)$.

Wir fassen nun die Fehlerschranken für die Fehlerfortpflanzung durch Addition, Subtraktion und Multiplikation in Tabellenform für den absoluten und für den relativen Fehler zusammen (vgl. [2]). Zunächst für den absoluten Fehler:

Absolute Fehler		
gegeben	$\Delta_\pm$	Δ_{Mul}
$\Delta(a), \Delta(b)$	$\Delta(a) + \Delta(b)$	$\Delta(a)\Delta(b) + \lvert A\rvert\Delta(b) + \lvert B\rvert\Delta(a)$
$\Delta(a), \varepsilon(b)$	$\Delta(a) + \lvert B\rvert\varepsilon(b)$	$\lvert B\rvert\,(\Delta(a)\varepsilon(b) + \lvert A\rvert\varepsilon(b) + \Delta(a))$
$\varepsilon(a), \Delta(b)$	$\lvert A\rvert\varepsilon(a) + \Delta(b)$	$\lvert A\rvert\,(\varepsilon(a)\Delta(b) + \Delta(b) + \lvert B\rvert\varepsilon(a))$
$\varepsilon(a), \varepsilon(b)$	$\lvert A\rvert\varepsilon(a) + \lvert B\rvert\varepsilon(b)$	$\lvert A\rvert \cdot \lvert B\rvert \cdot (\varepsilon(a)\varepsilon(b) + \varepsilon(a) + \varepsilon(b))$

Nun für den relativen Fehler:

Relative Fehler		
gegeben	$\varepsilon_{\pm}$	$\varepsilon_{\mathrm{Mul}}$
$\Delta(a), \Delta(b)$	$\frac{\Delta(a)+\Delta(b)}{\langle A \pm B \rangle}$	$\frac{\Delta(a)}{\langle A \rangle} \cdot \frac{\Delta(b)}{\langle B \rangle} + \frac{\Delta(b)}{\langle B \rangle} + \frac{\Delta(a)}{\langle A \rangle}$
$\Delta(a), \varepsilon(b)$	$\max\limits_{b \in \{\underline{b}, \overline{b}\}} \frac{\Delta(a)+\lvert b \rvert \varepsilon(b)}{\langle A \pm b \rangle}$	$\frac{\Delta(a)}{\langle A \rangle} \cdot \varepsilon(b) + \varepsilon(b) + \frac{\Delta(a)}{\langle A \rangle}$
$\varepsilon(a), \Delta(b)$	$\max\limits_{a \in \{\underline{a}, \overline{a}\}} \frac{\lvert a \rvert \varepsilon(a)+\Delta(b)}{\langle a \pm B \rangle}$	$\varepsilon(a) \cdot \frac{\Delta(b)}{\langle B \rangle} + \frac{\Delta(b)}{\langle B \rangle} + \varepsilon(a)$
$\varepsilon(a), \varepsilon(b)$	$\max\limits_{a \in \{\underline{a}, \overline{a}\}, b \in \{\underline{b}, \overline{b}\}} \frac{\lvert a \rvert \varepsilon(a)+\lvert b \rvert \varepsilon(b)}{\lvert a \pm b \rvert}$	$\varepsilon(a)\varepsilon(b) + \varepsilon(a) + \varepsilon(b)$

Entsprechende Formeln für die Division bzw. für weitere elementare Funktionen sowie notwendige Zusatzüberlegungen zur sicheren Berücksichtigung der Rundungsfehler sind, wie bereits erwähnt, in [2] zu finden.

Der hier kurz vorgestellte Fehlerkalkül nutzt eine schnelle und genaue Funktionsbibliothek [6], die auf das IEEE-double-Zahlformat aufsetzt und nur IEEE-double-Operationen benutzt. Dabei werden keine Anforderungen an den eingestellten Rundungsmodus gestellt. Es werden polynomiale bzw. rationale Bestapproximationen ebenso wie schnelle Tabellenverfahren für die Grundroutinen verwendet. Zusammen mit den a priori bestimmten worst-case Fehlerschranken (diese wurden bereits mit der oben beschriebenen Methodik berechnet) erreicht man sehr kurze Laufzeiten, eine hohe Ergebnisgenauigkeit, sichere Wertebereichseinschließungen und eine gute Portabilität.

Zusammenfassend kann gesagt werden, daß automatisch generierte Fehlerschranken nicht nur alle Rundungsfehler und ihre Fortpflanzung beachten, sondern auch die Konvertierungsfehler, die beispielsweise gemacht werden, falls Konstanten nicht exakt dargestellt werden können. Zusätzlich werden Zwischenergebnisse, die im Underflow-Bereich liegen, berücksichtigt. Die übliche Vernachlässigung von Fehlertermen höherer Ordnung entfällt. Sie werden sicher mit erfasst.

Für die Umsetzung des Fehlerkalküls benötigt man Intervalloperationen, Operator- und Funktionsüberladung sowie die automatische Differentiation.

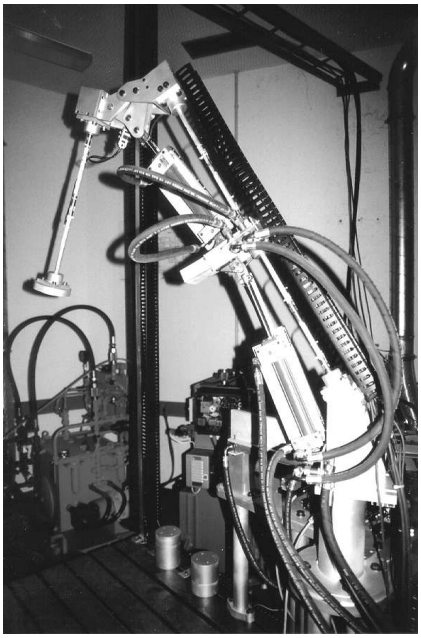
Die Anwendungsmöglichkeiten sind vielfältig, beispielsweise können Programme mit Schleifen, Iterationen und Rekursionen auf ihre Genauigkeitew untersucht werden. So wurde der Kalkül sehr erfolgreich bei der Entwicklung der schnellen Funktionsbibliothek `FI_LIB` [6] eingesetzt. Man kann auch hochgenaue Berechnungen anstellen oder Fehlerschranken für komplexe Programme ermitteln, wie wir es in den folgenden Beispielen getan haben.

4 Verifikation mit Fehlerschrankenarithmetik

In diesem Abschnitt befassen wir uns mit der Verifikation von Beispielen, die im Sonderforschungsbereich 291 eine wichtige Rolle spielen. Dazu setzen wir PASCAL XSC [9] mit der darin implementierten Intervallrechnung ein. Zur Bestimmung von sicheren Abschätzungen für absolute Fehler, die bei Berechnungen auftreten, setzen wir das Modul `abs_ari` ein, das am Institut

für Wissenschaftliches Rechnen und Mathematische Modellbildung in Karlsruhe von Werner Hofschuster und Dr. Walter Krämer entwickelt wurde [**?**].

Die Teilprojekte A1 (Leitung: Prof. Dr. Hiller) und B4 (Leitung: Prof. Dr. Schwarz) befassen sich mit der Modellierung mechatronischer Systeme. Neben der theoretischen Modellbildung werden auch Ansätze zur experimentellen Modellbildung untersucht. Dazu wurde im Fachgebiet Meß-, Steuer- und Regelungstechnik ein hydraulisch betriebener Versuchsträger (HyRob), der wesentliche Eigenschaften eines Schwerlastmanipulators besitzt, aufgebaut (siehe Bild 1 a)).



a) Foto des Prüfstands

Abbildung 1: Dreiachsiger Versuchsträger HyRob

In Bild 1 b) ist der konstruktive Aufbau des Versuchsträgers zu sehen. Der Ausleger des Manipulators besteht aus zwei über einen Umlenkmechanismus miteinander verbundenen Armen und ist auf einer drehbaren Basis montiert. Diese ermöglicht Drehungen um die Hochachse und besitzt einen Drehbereich von $\Theta_S = 0° - 90°$. Der Arm 1 des Auslegers besteht aus zwei elastischen Segmenten und kann Drehbewegungen um eine horizontale Achse in einem Bereich von $\Theta_A = 0° - 90°$ durchführen. Der Arm 2 ist ebenfalls elastisch und ist mit dem ersten Arm über ein Drehgelenk verbunden, das Drehungen um eine horizontale Achse in einem Bereich von $\Theta_B = -180° - 0°$ zuläßt. Der Ausleger besitzt im ausgestreckten Zustand eine Reichweite von ca. 1.6 m.

Die hydrostatischen Antriebe sind entkoppelt im System angeordnet, das heißt, die Drehbewegung der Säule bzw. die Drehbewegungen der einzelnen Arme können, abgesehen von der dynamischen Koppelung, unabhängig voneinander ausgeführt werden. Die Hydraulikzylinder der einzelnen Antriebseinheiten werden separat über Servoventile angesteuert. Während der Antrieb der Säule und des Arms 1 kinematisch betrachtet mit Hilfe eines einschleifigen Mechanismus erfolgt, erfordert der Antrieb des zweiten Arms aufgrund des großen Schwenkwinkels einen zweischleifigen Umlenkmechanismus. Dieser besteht aus Hydraulikzylinder, Umlenkhebel und Druckstange.

4.1 Der hydraulische Aktuator

Das Teilprojekt B4 befaßt sich u. a. mit der Modellbildung hydraulischer Aktuatoren. Die Modellbildung eines Differentialzylinders von HyRob wurde mit speziellen Parameteridentifikationsverfahren durchgeführt, die im Fachgebiet Meß-, Steuer- und Regelungstechnik entwickelt wurden [12]. Die so identifizierten Modelle werden zur Simulation eingesetzt, wobei die Modellgleichungen iterativ ausgewertet werden. Hierdurch können prinzipiell Fehlerfortpflanzungsprobleme auftreten. Die Verläßlichkeit der Simulationsergebnisse kann gesteigert werden, indem man sichere Schranken für den auftretenden Fehler angibt.

In dem untersuchten Beispielsystem treten vier physikalische Größen auf, nämlich Druck des Hydrauliköls in Zylinderkammer A und Zylinderkammer B, Position des Kolbens und die Ansteuerspannung des Servoventils. Zur Bestimmung des Fehlers wurden für diese Größen Wertebereiche angenommen, die den realen Verhältnissen entsprechen.

Das in MATLAB [20] geschriebene Modell wird in PASCAL XSC übersetzt, wobei die Wertevariablen als `BoundType` deklariert werden, dieser Typ wird von dem Modul `abs_ari` bereitgestellt. Die Routinen zur Berechnung der Exponentialfunktion und des natürlichen Logarithmus ergänzen wir um die Bestimmung des absoluten Fehlers. Zusätzlich benötigen wir die Funktion `sqr`, die eine Variable vom Typ `BoundType` quadriert. Hierbei wird das Ergebnisintervall auf den nichtnegativen Bereich beschränkt, da eine innere Abhängigkeit der Intervallfaktoren vorliegt.

Zur Simulation der Position und der Drücke in den beiden Zylinderkammern sind im Fachgebiet Meß-, Steuer- und Regelungstechnik Modelle entwickelt worden. Zur Bestimmung der neuen Position im nächsten Zeitschritt benötigen wir in dem Positionsmodell drei Parameter, sie wird aus der alten Position, dem Druck in Zylinderkammer B und der Steuerspannung berechnet. Für diese drei Eingangsparameter werden jeweils die größtmöglichen Intervalle, die für die Praxis relevant sind, gewählt. Hierdurch ergeben sich dann allerdings Parameterkonstellationen, die real nicht vorkommen.

Die Berechnung der Funktion führt bei den oben beschriebenen Eingangsparametern zu keinem Ergebnis, da Divisionen durch Intervalle, die die Null enthalten, auftreten. Um dieses zu vermeiden, nehmen wir eine Intervallunterteilung in allen drei Eingangsgrößen vor. Die so entstandenen Würfel überprüfen wir nun einzeln, einen nicht verifizierbaren Würfel unterteilen wir wiederum in acht kleinere. So erhalten wir eine rekursive Prozedur. Zusätzlich soll eine maximale Rekursionstiefe eingegeben werden, um Ergebnisse in einer erwarteten Zeit zu erhalten. Parameterkonstellationen, die bis zur maximalen Rekursionstiefe nicht verifiziert berechnet werden können, geben wir in eine Datei aus. Ferner können wir im Programm einstellen, ob ein korrekt berechneter Würfel trotzdem noch unterteilt werden soll, um auf diese Weise den Wert für den absoluten Fehler zu verbessern. Auf diese Weise ist es uns möglich, 99.9999998% aller möglichen Eingangsparameterkonstellationen mit einem maximalen absoluten Fehler von $8.4 \cdot 10^{-16}$ m zu verifizieren. Die Prozentangabe beschreibt dabei das Verhältnis vom Volumen der Würfel mit verifizierbaren Parameterkonstellationen zum gesamten Würfelvolumen aller Konstellationen. Von den nicht verifizierbaren Parameterkonstellationen treten ca. 70% in der Realität nicht auf. Bei dem auch in Simulationen auftretenden Restbereich liefert das Modell für den Druck in Zylinderkammer A, das vom Teilprojekt B4 entwickelt wurde, sehr schlechte Ergebnisse. An dieser Stelle muß entweder das Modell verbessert werden, oder die Konstellation von Druck in Zylinderkammer B zwischen 2164575.4 Pa und 2164575.6 Pa und der Steuerspannung zwischen 0.3 V und 0.51 V muß bei Simulationen ausgeschlossen werden, da hier keine verifizierbaren Ergebnisse zu liefern sind.

Als nächstes verifizieren wir Positionswerte, die sich aus einer Meßreihe ergeben. Im Abstand von zwei Millisekunden werden am Versuchsträger die Werte für die Position, die beiden Drücke und die Steuerspannung gemessen. Wir benutzen für das Verifikationstool einen Datensatz von jeweils 8000 Werten. Im vorliegenden Modell für die Positionsberechnung gehen zur Bestimmung der Position im Zeitschritt k die Werte für die Position und den Druck in Zylinderkammer B im Zeitschritt $k-1$ sowie der Wert der Ansteuerspannung im Zeitschritt $k-4$ ein. Wir berechnen nun die neue Position mit Hilfe der gemessenen Werte für den Druck und die Spannung, und wir benutzen die im vorangegangenen Zeitschritt bestimmte Position sowie deren Fehler. Für die erste Positionsberechnung nehmen wir den vorher gemessenen Positionswert. So können wir nach 8000 Schritten ein verifiziertes Intervall für die Endposition sowie den maximalen absoluten Fehler, der bei der Berechnung auftritt, angeben. Wir erhalten ein Intervall von 0.10350081702 m – 0.10350081703 m. Der maximale absolute Fehler liegt bei $3.71 \cdot 10^{-12}$ m. Somit treten bei der hier durchgeführten Positionsberechnung keine Probleme auf.

Zusätzlich vergleichen wir unseren berechneten Wert mit dem gemessenen Wert. Hierbei ist die maximal aufgetretene Abweichung kleiner als 0.015 m. Nach 8000 Schritten liegt die Differenz bei weniger als $1.81 \cdot 10^{-4}$ m. Bei dieser Meßreihe liefert das Modell somit sehr gute Werte.

Werden die Eingangswerte mit einer Genauigkeit gemessen, so daß die Meßfehler auf 0.05% für den Druck in Zylinderkammer B bzw. 1% für die Steuerspannung beschränkt sind, so liegt der maximale absolute Fehler nach 8000 Schritten bei $6.9 \cdot 10^{-10}$ m. Der Meßfehler wird dabei gleichverteilt simuliert.

Die Modelle zur Simulation von den Drücken in den beiden Zylinderkammern sind dagegen nicht auf die gleiche Weise umsetzbar. Schon nach nur 35 Schritten sind die berechneten Intervalle und absoluten Fehler so groß, daß die Ergebnisse unbrauchbar sind. Wir können die Simulationswerte für diese Parameter nicht auf die gleiche Weise verifizieren wie die Werte für die Position. Somit können wir auch keine verläßlichen Aussagen über die Güte der im Modell auftretenden Berechnungen machen. Auch Intervallunterteilungsmethoden liefern hier keine Aussagen über lange Meßreihen.

4.2 Der einachsige Manipulator

Der Versuchsträger HyRob stellt aus mechanischer Sicht ein komplexes System dar, da aufgrund der drei hydraulischen Linearantriebe vier kinematische Schleifen auftreten.

Im folgenden wird ein einfaches Teilsystem untersucht. In Abbildung 2 ist eine Darstellung eines einachsigen Manipulators aufgezeigt. Dieses Teilsystem ergibt sich, wenn beim Versuchsträger die Antriebseinheit zur Drehung der Basis, der Umlenkmechanismus mit dem zugehörigen hydraulischen Linearantrieb und der Arm 2 entfernt werden. Das Teilsystem beinhaltet die Antriebseinheit zur Bewegung des Arms 1 in Abbildung 1 b).

In Bild 2 ist die mechanische Modellierung zu erkennen. Die Antriebseinheit wird als Viergelenkkette, bestehend aus einem Drehgelenk R1, einem Schubgelenk P, einem Kardangelenk T, das aus den beiden Drehgelenken R2 und R3 zusammengesetzt ist, und einem Kugelgelenk S modelliert. Der einachsige Manipulator besitzt den Freiheitsgrad $f = 1$. Die Eigendynamik der Hydraulik wird für die in diesem Abschnitt durchgeführten Untersuchungen vernachlässigt, so daß auf das den Hydraulikzylinder nachbildende Schubgelenk ein idealer Aktuator, der eine beliebige Stellkraft `u1` generieren kann, wirkt. Der einachsige Manipulator bewegt eine Endlast mit der Masse `M12`.

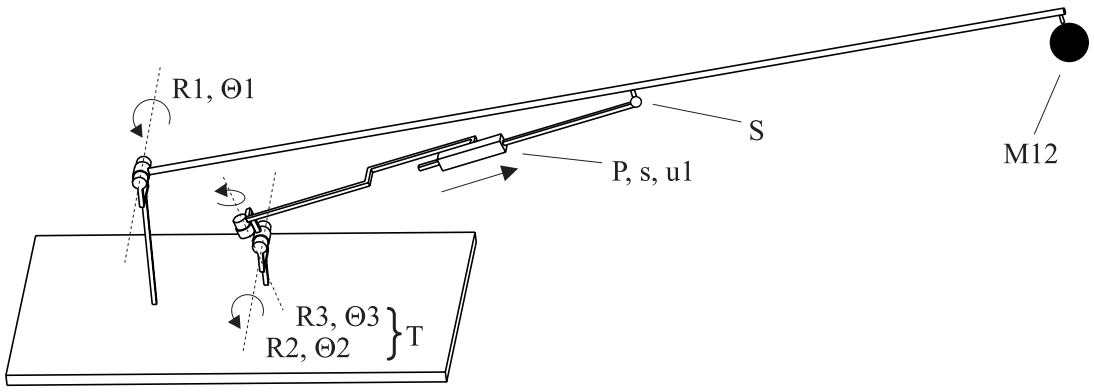


Abbildung 2: Eine schematische Darstellung des einachsigen Manipulators

Zur Beschreibung des dynamischen Verhaltens mechanischer Systeme werden Bewegungsgleichungen aufgestellt. Mit Hilfe des Kinematik- und Dynamikcompilers SYMKIN [8], der in MATHEMATICA [22] programmiert ist, können die Bewegungsgleichungen mechanischer Systeme symbolisch und automatisch in Minimalform generiert werden. In Abhängigkeit von dem Freiheitsgrad f des betrachteten mechanischen Systems erhält man f gewöhnliche Differentialgleichungen zweiter Ordnung mit $2f$ Anfangsbedingungen, die als Anfangswertproblem gelöst werden können.

Bevor wir die Differentialgleichung im Abschnitt 6 lösen werden, führen wir zuerst die Berechnungen der dort auftretenden Werte für das Massenmatrix-Element, die verallgemeinerten Kreiselkräfte und die verallgemeinerten eingeprägten Kräfte mit PASCAL XSC und der Fehlerschrankenarithmetik `abs_ari` aus, und wir erhalten so Intervalleinschlüsse und Aussagen über den absoluten und relativen Fehler.

Als Grundlage dient ein SYMKIN-Programm. Der vorgegebene Programmcode muß zuerst in PASCAL XSC-Syntax umgeformt werden. Alle auftretenden Variablen werden dann als `BoundType` deklariert. Die Funktion `sqr` zum Quadrieren einer `BoundType`-Variablen wird ergänzt, ebenso wie die Argumentfunktion `arctan2(y,x)`, die $\arctan\left(\frac{x}{y}\right)$ im Fall $y > 0$ beziehungsweise $\pi + \arctan\left(\frac{x}{y}\right)$ andernfalls berechnet. Die Sinus- und Kosinusfunktionen werden jeweils um die Berechnung des maximalen absoluten Fehlers erweitert. Ausgegeben werden am Ende der Berechnungen die Werte des Massenmatrix-Elements, der verallgemeinerten Kreiselkräfte und der verallgemeinerten eingeprägten Kräfte.

Tabelle 1 zeigt die berechneten Werte, der jeweils zweite Wert bezieht sich auf die Situation, daß das Gewicht `M12` um $\pm 10\%$ vom Ursprungswert 1,0 kg abweicht. Die erste Zeile zeigt die mit MOBILE [7] berechneten Werte, danach sind die verifizierten Ergebnisse angegeben. Hierbei sind sowohl die Ergebnisintervalle und deren Durchmesser als auch die auftretenden maximalen absoluten und relativen Fehler aufgeführt. Wir erkennen, daß bei einer unscharfen Eingabe der Masse der Endlast die auftretenden Kräfte sehr unterschiedlich sein können, da sich große Intervalle ergeben. Trotzdem sind die auftretenden Fehler bei einer Berechnung für eine fest vorgegebene Masse zwischen 0,9 kg und 1,1 kg sehr gering. Wir haben somit verifiziert, daß in den angegebenen Toleranzen für die Masse der Endlast — und diese ist i. a. vorher nicht genau bekannt — Rechnungen nur unerheblich kleine Fehler liefern, obwohl die erzielten Ergebnisse stark voneinander abweichen.

	Massenmatrix	Kreiselkräfte	eingepr. Kräfte
MOBILE	45.155411	607.477	50.1157545
Intervall	$[45.1554, 45.1555]$	$[607.4766, 607.4767]$	$[50.1157, 50.1158]$
	$[40.6, 49.7]$	$[546.7, 668.3]$	$[44.7, 55.5]$
Durchmesser	$4.9 \cdot 10^{-12}$	$2.6 \cdot 10^{-8}$	$3.2 \cdot 10^{-12}$
	9.1	121.6	10.8
abs. Fehler	$7.2 \cdot 10^{-12}$	$5.9 \cdot 10^{-8}$	$4.7 \cdot 10^{-12}$
	$7.9 \cdot 10^{-12}$	$6.4 \cdot 10^{-8}$	$5.1 \cdot 10^{-12}$
rel. Fehler	$1.6 \cdot 10^{-16}$	$9.6 \cdot 10^{-11}$	$9.2 \cdot 10^{-14}$
	$2.0 \cdot 10^{-13}$	$1.2 \cdot 10^{-10}$	$1.2 \cdot 10^{-13}$

Tabelle 1: Mit PASCAL XSC und `abs_ari` berechnete Werte für den einachsigen Manipulator

5 Ein automatischer Anfangswertproblemlöser

In diesem Abschnitt stellen wir kurz die Grundprinzipien vor, die die theoretischen Grundlagen für das Modul zur automatischen Lösung von Anfangswertproblemen, AWA [14, 15, 16, 17, 18], bilden, das von Herrn Dr. Lohner am Institut für Angewandte Mathematik in Karlsruhe entwickelt wurde. Auch hier sei noch einmal darauf hingewiesen, daß die vorgestellten Ergebnisse eine Zusammenfassung des von Herrn Dr. Krämer während des Seminars gehaltenen Vortrags sind. Für ein tieferes Verständnis sollte die oben zitierte Literatur durchgesehen werden.

Gegeben sei ein Anfangswertproblem

$$u' = f(t,u) \text{ mit } u(t_0) = u_0 \in D,$$

wobei f hinreichend glatt sein und $u : J \to \mathbb{R}^n$ mit $J = [t_0, T] \subset \mathbb{R}$ gelten möge. D kann ein Intervallvektor sein mit $D = [u_0] = [\underline{u}_0, \overline{u}_0]$. Wir suchen nun eine stetige Einschließung $[\underline{u}(t), \overline{u}(t)]$ der korrekten Lösung $u(t)$ des Anfangswertproblems für alle $t \in J$. Das bedeutet, wir suchen einen Funktionenschlauch

$$[u(t)] = [\underline{u}(t), \overline{u}(t)] = \{u \in C(J) \mid \underline{u}(t) \le u(t) \le \overline{u}(t)\} \text{ für alle } t \in J,$$

mit $u(t) \in [u(t)]$ für alle $t \in J$ und $u(t_0) = u_0 \in [u_0]$.

Es gibt verschiedene Ansätze zur Lösung dieses Problems, wir verweisen unter anderem auf Moore [19], Krückeberg [13], Eijgenraam [5], Bauch [3], Lohner [15] und Rihm [21].

Die meisten Algorithmen benötigen eine erste grobe Einschließung $U(t)$ von $u(t)$ in jedem Zeitschritt, die sogenannte *a priori Einschließung*. $U(t)$ wird dann später benutzt, um eine bessere Einschließung in diesem Zeitschritt zu bestimmen.

Ausgangspunkt für die Berechnung einer a priori Einschließung ist die äquivalente Integralgleichung im j-ten Zeitschritt $t \in J_j := [t_j, t_j + h]$:

$$u(t) = (Tu)(t) = u(t_j) + \int_{t_j}^{t} f(s, u(s))\, ds.$$

Der Operator T ist eine Kontraktion in einer geeignet gewählten gewichteten Maximumsnorm auf $C(J_j)$. T wird benutzt, um eine a priori Einschließung für $u(t)$ zu konstruieren.

Sei $U \subset C(J_j)$ eine abgeschlossene Menge stetiger Funktionen auf J_j. Falls der Operator T die Menge U in sich selbst abbildet, das heißt, falls

$$TU \subset U$$

erfüllt ist, so existiert nach dem Banachschen Fixpunktsatz ein eindeutiger Fixpunkt $u \in U$ von T mit $u = Tu$. Der Funktionenschlauch $U(t)$ aller Funktionen aus U ist dann eine Einschließung der Lösung $u(t)$ des Anfangswertproblems auf J_j, das heisst, U ist eine a priori Einschließung von u.

Es gibt nun verschiedene Möglichkeiten, die Struktur von U festzulegen. In traditionellen Algorithmen hat U eine sehr einfache Struktur, es ist ein Funktionenschlauch mit konstanten Grenzen:

$$U = [u^0] = [\underline{u}^0, \overline{u}^0] = \left\{ u \in C(J_j) \mid \underline{u}^0 \leq u(t) \leq \overline{u}^0, \;\; t \in J_j \right\}$$

mit $\underline{u}^0, \overline{u}^0 \in I\!R^n$. In diesem Fall kann die intervallmäßige Auswertung TU des Integraloperators T einfach berechnet werden. Denn für alle $u^0 \in U$ gilt

$$\begin{aligned} u^1(t) := (Tu^0)(t) &= u^0(t_j) + (t - t_j) f(\tau, u^0(\tau)) \\ &\in u^0(t_j) + [0,h] f(J_j, [u^0]) =: [u^1] \end{aligned}$$

und nach Konstruktion

$$TU \subset [u^1].$$

Nun genügt es, die Bedingung

$$[u^1] \subset [u^0]$$

nachzuprüfen. Ist diese erfüllt, so folgt

$$TU \subset [u^1] \subset [u^0] = U.$$

Die letzten Inklusionen sind sehr einfach zu überprüfen, da sowohl $[u^0]$ als auch $[u^1]$ Intervallvektoren sind.

Dieser Ansatz hat den Nachteil, daß die maximal mögliche Schrittweite h in der Größenordnung der Schrittweite des Euler-Verfahrens liegt, unabhängig von der Ordnung p des nachfolgenden Einschließungsverfahrens. Das reduziert die Effizienz des gesamten Verfahrens und macht häufig eine Schrittweiten- und Ordnungskontrolle sinnlos.

Um diese Schrittweitenbeschränkung zu umgehen, kann man versuchen, Polynome höheren Grades für die Ersteinschließung $U(t)$ heranzuziehen. Sei

$$U(t) = \sum_{i=0}^{p-1} (u)_i (t - t_j)^i + [u_p](t - t_j)^p \tag{1}$$

mit $(u)_i = \frac{u^{(i)}(t_j)}{i!}$ als i-tem Taylor-Koeffizient von u und

$$[u_p] = [\underline{u}_p, \overline{u}_p] = \left\{ u_p \in C(J_j) \mid \underline{u}_p(t) \leq u_p(t) \leq \overline{u}_p(t) \right\}$$

das Taylor-Polynom der Ordnung $p-1$ der exakten Lösung u an der Stelle t_j plus einem Intervallterm. Die Taylor-Koeffizienten $(u)_i$ können rekursiv berechnet werden unter Verwendung der automatischen Differentiation.

Für den Intervall-Koeffizienten $[u_p]$ werden wir einen Inklusionstest, ähnlich wie bei den konstanten Grenzen, entwickeln, der für die Bedingung $TU \subset U$ hinreichend sein wird. Hier ist jedoch die intervallmäßige Auswertung des Integraloperators TU wesentlich komplizierter. Wir gehen folgendermaßen vor:

Sei U^0 ein Funktionenschlauch der obigen Form mit $[u_p] = [\underline{u}_p^0, \overline{u}_p^0]$. Wir wollen dann eine Einschließung U^1 von TU^0 berechnen mit $U^1 \supset TU^0$, wobei U^1 ebenfalls die Form (1) mit $[u_p] = [u_p^1] = [\underline{u}_p^1, \overline{u}_p^1]$ haben soll.

Dies kann tatsächlich erreicht werden, da T die Taylor-Koeffizienten der exakten Lösung reproduziert: Gilt

$$v(t) = \sum_{i=0}^{p-1} (u)_i (t-t_j)^i + v_p(t)(t-t_j)^p,$$

so folgt

$$(Tv)(t) = \sum_{i=0}^{p-1} (u)_i (t-t_j)^i + \tilde{v}_p(t)(t-t_j)^p,$$

wobei $(u)_i$ die Taylor-Koeffizienten der exakten Lösung u und v_p, $\tilde{v}_p$ stetige Funktionen sind.

Dies bedeutet, daß, wenn wir U^1 aus U^0 berechnen wollen, wir nur die Koeffizienten $[u_p^1]$ von $(t-t_j)^p$ zu berechnen brauchen. Die Koeffizienten $(u)_i$ von $(t-t_j)^i$, $0 \le i \le p-1$, sind identisch für U^0 und U^1 und sind gerade die Taylor-Koeffizienten der exakten Lösung u. Diese können wieder durch automatische Differentiation berechnet bzw. eingeschlossen werden.

Die Berechnung des p-ten Koeffizienten kann dadurch realisiert werden, daß man für jede Operation $c = a \circ b$, die in T erlaubt ist, einen Operator implementiert, der den p-ten Taylor-Koeffizienten $[c_p]$ des Ergebnisses unter Benutzung der Taylor-Koeffizienten der Operanden a und b berechnet.

Im hier vorgestellten Programm AWA sind folgende Operatoren implementiert: Die Grundrechenarten $\{+, -, \cdot, /\}$, die Integration $\int_{t_j}^{t} \ldots ds$ und Standardfunktionen, wie sin, cos, exp, ln, usw. [4].

Um nun den Operator T iterativ für Funktionenschläuche U der Form (1) zu benutzen (also für $U^1(t) = (TU^0)(t)$), brauchen wir dann nur den p-ten Koeffizienten von $U^1(t)$ unter Verwendung der obigen Operatoren zu berechnen.

Schließlich reicht es aus, für den Inklusionstest $TU \subset U$ die Inklusion

$$U^1(t) \subset U^0(t)$$

zu testen, denn dies ist äquivalent zu

$$\sum_{i=0}^{p-1} (u)_i (t-t_j)^i + [u_p^1](t-t_j)^p \subset \sum_{i=0}^{p-1} (u)_i (t-t_j)^i + [u_p^0](t-t_j)^p,$$

also zu

$$[u_p^1] \subset [u_p^0].$$

Wieder reduziert sich der Inklusionstest auf einen Inklusionstest für Intervallvektoren, welcher einfach komponentenweise durchgeführt werden kann (komponentenweise Inklusion ist hinreichend für das Enthaltensein der betrachteten Funktionsschläuche). Ist die Inklusion erfüllt, so gilt für die exakte Lösung u

$$u(t) \in U^1(t) = \sum_{i=0}^{p-1} (u)_i (t - t_j)^i + [u_p^1](t - t_j)^p$$

für alle $t \in [0, h]$, wobei für größere Werte von p auch h größere Werte annehmen kann, wie numerische Beispiele zeigen.

Kommen wir nun zu der weiteren Vorgehensweise. Wir betrachten im folgenden ein beliebiges Einschrittverfahren $\Phi(u)$ mit lokalem Diskretisierungsfehler $\frac{z_{j+1}}{h}$ auf $[t_j, t_{j+1}] = [t_j,\, t_j + h]$ und ermitteln damit die Werte $u_j := u(t_j)$ durch

$$u_{j+1} = u_j + h \cdot \Phi(u_j) + z_{j+1}.$$

Dies ist eine exakte Formel, das heißt, u_j ist der exakte Wert der Lösung, da z_{j+1} den exakten lokalen Fehler darstellt. Die Methode Φ ist nun so zu wählen, daß der lokale Fehler z_{j+1} leicht mittels u und/oder Ableitungen von u ausgedrückt und berechnet werden kann.

Eine mögliche Wahl ist die Taylor-Entwicklung in t_j. Dann gilt

$$z_{j+1} = \frac{h^p}{p!} u^{(p)}(\tau_{j+1}), \quad \text{mit } \tau_{j+1} \in (t_j, t_{j+1}).$$

Im Prinzip können implizite und Mehrschritt-Verfahren ähnlich behandelt werden. Jedoch sind die Details komplizierter und noch nicht vollständig gelöst.

Seien nun $z_0, \ldots, z_{j+1}$ als unabhängige Variablen vorausgesetzt. Dies definiert eine Funktion

$$u_{j+1} = u_{j+1}(z_0, z_1, \ldots, z_{j+1}). \tag{2}$$

Benutzen wir die exakten lokalen Fehler z_k, so erhalten wir die exakte Lösung $u(t_{j+1})$. Jedoch sind die lokalen Fehler z_k unbekannt. Daher ersetzen wir sie durch einschließende Intervalle $[z_k]$ mit $z_k \in [z_k]$. Die intervallmäßige Auswertung von (2) ergibt eine Einschließung $[u_{j+1}]$ von $u(t_{j+1})$. Das verbleibende Problem ist nun, die Einschließungen $[z_k]$ der lokalen Diskretisierungsfehler konstruktiv zu bestimmen.

Dies kann unter Verwendung einer a priori Einschließung $U(t)$ von u im Zeitschritt $[t_j, t_{j+1}]$ geschehen. Verwenden wir diese a priori Einschließung im Restglied der Taylor-Entwicklung, unter Anwendung der automatischen Differentiation in Verbindung mit Intervallarithmetik, so erhalten wir die gewünschte Einschließung $[z_{j+1}]$ von z_{j+1}.

Schließlich kann eine kontinuierliche Einschließung der Lösung $u(t)$ auf $[t_j, t_{j+1}]$ als ein Intervall-Polynom angegeben werden:

$$u(t) \in [u_j] + [u_j'](t - t_j) + \ldots + \frac{(t - t_j)^{p-1}}{(p-1)!} [u_j^{(p-1)}] + \left(\frac{t - t_j}{h}\right)^p [z_{j+1}].$$

Es ist möglich, dieses Verfahren auf das nächste Teilintervall anzuwenden. Dazu startet man den Zeitschritt $j + 1$ mit der eben berechneten Einschließung $[u_{j+1}]$ der Lösung ausgewertet am rechten Rand des vorhergehenden Zeitschrittes.

Der Nachteil des beschriebenen Verfahrens ist, daß Intervall-Polynome streng monoton wachsende Durchmesser mit wachsendem t haben, was dann auch zu Einschließungen der Lösungsfunktion mit streng monoton wachsenden Durchmessern führt - meist ist dieses Wachstum sogar exponentiell -, selbst, wenn alle Lösungen gegen Null konvergieren, wie dies z. B. für die Differentialgleichung $u' = -u$ der Fall ist.

Um diese Überschätzung zu vermindern, berechnen wir (2) als Mittelwertform, was für kleine Durchmesser von $[z_k]$ zu wesentlich besseren Einschließungen von $[u_{j+1}]$ führt.

Dazu wählen wir $s_0, \ldots, s_{j+1}$ aus den Intervallen $[z_0], \ldots, [z_{j+1}]$ (beispielsweise die Mittelpunkte) und wenden den Mittelwertsatz im Entwicklungspunkt $(s_0, \ldots, s_{j+1})$ an. Wir erhalten

$$[u_{j+1}] = \tilde{u}_{j+1} + \sum_{k=0}^{j+1} \frac{\partial u_{j+1}([z])}{\partial z_k}([z_k] - s_k)$$

mit

$$\begin{aligned} \tilde{u}_{j+1} &:= u_{j+1}(s_0, \ldots, s_{j+1}) \\ &= \tilde{u}_j + h \cdot \Phi(\tilde{u}_j) + s_{j+1} \text{ für alle } j \geq 0 \end{aligned}$$

und

$$[z] = ([z_0], [z_1], \ldots, [z_{j+1}]).$$

Dies ist eine Erweiterung unter Berücksichtigung aller vorherigen lokalen Fehler. Führen wir zur Abkürzung die Intervallmatrizen

$$A_k := I + h\frac{\partial \Phi([u_k])}{\partial u} \text{ und } A_{j+1,k} := \frac{\partial u_{j+1}([z])}{\partial z_k}$$

ein, so gilt

$$A_{j+1,k} = A_j A_{j-1} \cdots A_k \text{ und } A_{j+1,j+1} = I \text{ (Einheitsmatrix)},$$

und die Mittelwertform wird zu

$$\begin{aligned} [u_{j+1}] &= \tilde{u}_{j+1} + \sum_{k=0}^{j+1} A_{j+1,k}([z_k] - s_k) \\ &= \tilde{u}_{j+1} + A_j \left(\sum_{k=0}^{j} A_{j,k}([z_k] - s_k) \right) + ([z_{j+1}] - s_{j+1}). \end{aligned}$$

Rkursiv geschrieben bedeutet dies

$$[u_{j+1}] = \tilde{u}_{j+1} + [r_{j+1}]$$

mit

$$[r_{j+1}] := A_j[r_j] + ([z_{j+1}] - s_{j+1}).$$

Eine geeignete intervallmäßige Auswertung von $[r_{j+1}]$ sollte nun im Vergleich mit der weiter oben beschriebenen Methode eine engere Einschließung für $[u_{j+1}]$ ergeben. Dabei bieten sich zur Auswertung der Rekursion (man möchte den sogenannten Wrapping-Effekt vermeiden) verschiedene Möglichkeiten an. Die einfachste besteht im Einsatz von achsenparallelen Rechtecken (Intervallvektoren). Bessere Einschließungen werden häufig mit Parallelepipeden erzielt, die mit Hilfe von

geeigneten QR-Zerlegungen bestimmt werden. Zur Vertiefung dieser Thematik verweisen wir auf [15].

In einem letzten Abschnitt wollen wir nun auf die automatische Schrittweitenkontrolle eingehen, die in AWA implementiert ist, und auf Eijgenraam [5] zurückgeht.

Wir versuchen, den lokalen Fehler ε_{j+1} im einzelnen Schritt zu kontrollieren. Wir streben

$$\varepsilon_{j+1} \approx h_{j+1} \left(E_{abs} + E_{rel} \left\| u([t_j, t_{j+1}]) \right\|\right)$$

an. (Mischung eines absoluten und eines relativen Fehlerkriteriums mit zugehörigen Fehlerparametern E_{abs} und E_{rel}.)

Der lokale Fehler wird im wesentlichen durch den Durchmesser des Restgliedes bestimmt:

$$\begin{aligned} \varepsilon_{j+1} &\approx d([(u_{j+1})_p]) \cdot h_{j+1}^p \\ &= \underbrace{\frac{d([(u_{j+1})_p])}{h_{j+1}}}_{\approx \text{konstant}} \cdot h_{j+1}^{p+1} \approx \frac{d([(u_j)_p])}{h_j} \cdot h_{j+1}^{p+1} \end{aligned}$$

Mit

$$E := E_{abs} + E_{rel} \cdot \left\| u([t_j, t_{j+1}]) \right\|,$$

wobei wir für $u([t_j, t_{j+1}])$ eine a priori Einschließung benutzen, erhalten wir eine Berechnungsvorschrift für eine neue Schrittweite

$$h_{j+1} := \left(\frac{E \cdot h_j}{d([(u_j)_p])} \right)^{\frac{1}{p}}.$$

Auf ähnliche Weise kann eine Anfangsschrittweite ermittelt werden:

$$h_1 := \left(\frac{E}{\left\| (u_0)_{p-1} \right\|} \right)^{\frac{1}{p-1}}.$$

Falls dieser Vorschlag sich als unbrauchbar erweist, so muß der Anwender h_1 selbst geeignet vorgeben.

Weiter verwenden wir die Inklusionsbedingungen $[u^1] \subset [u^0]$ bzw. $[u_p^1] \subset [u_p^0]$ für die a priori Einschließung in einer anderen Weise. Wir nutzen sie, um eine maximale Schrittweite zu bestimmen, für die diese Inklusionsbedingungen erfüllt sind. Für die konstante a priori Einschließung muß dann

$$[u_j] + [0, h] \cdot f([t_j, t_j + h], [u^0]) \overset{!}{\subset} [u^0]$$

und für die polynomiale Einschließung

$$(u)_p + \frac{[0, h]}{p+1} [f_p([t_j, t_j + h], [u_p])] \overset{!}{\subset} [u_p]$$

gelten. Die endgültige Schrittweite wird nun in zwei Schritten ermittelt:

1. Berechne Schrittweitenvorschlag h_{j+1} wie oben beschrieben.
2. Falls nötig, reduziere h_{j+1} so lange, bis die obigen Bedingungen erfüllt sind.

Bei der Umsetzung ist darauf zu achten, daß alle berechneten Schrittweiten h_{j+1} und alle berechneten Zwischenstellen maschinendarstellbare Zahlen sind.

6 Verifizierte Lösung der Bewegungsgleichungen des einachsigen Manipulators

Nun werden wir die Differentialgleichung, die sich bei dem Beispiel des einachsigen Manipulators in Abschnitt 4.2 ergibt, mit Hilfe von AWA [14, 15, 16, 17, 18] lösen. Die Bewegungsgleichungen werden mit dem Kinematik- und Dynamik-Compiler SYMKIN [8] erzeugt. Sie liegen dann in der Syntax von MATHEMATICA [22] vor und werden in ein MOBILE-Programm [7] integriert, hier werden sie dann mit Hilfe des Adams-Integrators gelöst.

Zur Berechnung von Intervalleinschlüssen der Lösung an diskreten Stellen, sowie zur Bestimmung des relativen Fehlers wird nun AWA eingesetzt. Hierzu muß die MOBILE-Syntax in eine für AWA verständliche Form konvertiert werden. Das bedeutet zuerst, jedes Semikolon am Zeilenende zu entfernen und danach die Differentialgleichung zweiter Ordnung in ein System von Differentialgleichungen erster Ordnung umzuformen. Beides ist problemlos möglich und läßt sich automatisieren. Die MOBILE-Variablen `s1, s1_d` und `s1_dd` entsprechen der gesuchten Funktion, sowie deren ersten beiden Ableitungen. Die entsprechenden AWA-Variablen `U1, U2, F1, F2` müssen folgendermaßen verwendet werden:
Vor dem übernommenen Code wird

```
s1 = U1
s1_d = U2
```

eingefügt, am Ende werden die Zeilen

```
F1 = U2
F2 = s1_dd
```

ergänzt. Zu beachten ist außerdem, daß in dem MOBILE-Code die Variable `u1` auftritt. Sie wird durch `uu` im gesamten Programm ersetzt. Für eine Automatisierung muß somit am Anfang der erhaltene Code auf das Vorhandensein der internen AWA-Variablen `U`n und `F`n überprüft werden, gegebenenfalls müssen diese durch andere Variablen ersetzt werden.

Das Hauptproblem in diesem Beispiel liegt in der benötigten Funktion

$$\text{arctan2} : \mathbb{R}\backslash\{0\} \times \mathbb{R} \to \left]-\frac{\pi}{2}, \frac{3}{2}\pi\right[,$$

$$\text{arctan2}(y,x) := \begin{cases} \arctan\left(\frac{x}{y}\right) & : \quad y > 0 \\ \pi + \arctan\left(\frac{x}{y}\right) & : \quad y < 0 \end{cases},$$

die in AWA nicht zur Verfügung steht. Die Konstruktion des einachsigen Manipulators läßt bei den vier auftretenden arctan2-Funktionen den Schluß zu, welcher Funktionszweig jeweils auszuwählen ist. Gestartet wird AWA mit vorgegebenen Anfangswerten, einem Integrationsintervall $[0, 1]$ und folgenden AWA-Parametern:

- Die anfängliche Schrittweite wird von AWA gewählt.
- Die Taylorkoeffizienten werden bis zur 20. Ordnung berechnet.
- Die Einschlußmethode wird mit Schnitten von Intervallvektoren und QR-Zerlegung durchgeführt.
- Die Fehlertoleranzen für den absoluten und relativen Fehler liegen jeweils bei 10^{-12}.

Als Ergebnis erhalten wir Schrittweiten schon zu Beginn in der Größenordnung 10^{-8} und eine inakzeptable Dauer jedes Schrittes.

Durch das Einstellen der anfänglichen Schrittweite auf 10^{-2} wird das gesamte Integrationsintervall in 2806 Schritten abgearbeitet. Zusätzlich müssen wir eine erhebliche Optimierung des mit SYMKIN erzeugten Codes vornehmen. Dadurch erhalten wir eine um den Faktor 4.8 höhere Geschwindigkeit. Die Optimierung umfaßt drei Punkte:

- Hilfsvariablen, die nur einmal auftauchen, werden ersetzt.
- Mehrmals durchgeführte gleiche Berechnungen werden nur noch einmal getätigt.
- Der anfängliche Code enthält die drei Winkel `theta1`, `theta2` und `theta3` als Lösung der Relativkinematik. Sie werden durch Arkustangens-Funktionen berechnet. Im weiteren Programm werden jedoch nur Sinus- und Kosinuswerte dieser Winkel benötigt. Im optimierten Code (siehe Anhang A) führen wir nun die notwendigen Berechnungen ohne die trigonometrischen Funktionen durch, indem wir die folgenden Ersetzungen vornehmen:

  Für den Winkel

  $$\Theta_1 = \arctan\left(\frac{B}{A}\right) - \arctan\left(\frac{\sqrt{A^2+B^2-C^2}}{C}\right)$$

  erhalten wir die Werte

  $$\begin{aligned} \sin\Theta_1 &= \frac{BC - A\sqrt{A^2+B^2-C^2}}{A^2+B^2}, \\ \cos\Theta_1 &= \frac{AC + B\sqrt{A^2+B^2-C^2}}{A^2+B^2}. \end{aligned}$$

  Die Winkel Θ_2 und Θ_3 sind von der Bauart

  $$\Theta = \arctan\left(\frac{A}{B}\right),$$

  hierbei ergibt sich

  $$\begin{aligned} \sin\Theta &= \frac{A}{A^2+B^2}, \\ \cos\Theta &= \frac{B}{A^2+B^2}. \end{aligned}$$

Als Ergebnis der Berechnung von AWA erhalten wir am Endpunkt unserer Berechnung ein Intervall mit Durchmesser $4.11 \cdot 10^{-11}$ für den Funktionswert und ein Intervall mit Durchmesser $5.03 \cdot 10^{-11}$ für den Wert der ersten Ableitung. Der maximale relative Fehler am Ende der Berechnung liegt bei $2.7 \cdot 10^{-10}$ bzw. bei $7.2 \cdot 10^{-10}$. Bei einem zweiten Beispiel mit der Stellgröße `uu=67.87` N statt `uu=60.0` N werden sogar noch bessere Ergebnisse erzielt. Hier liegen die Durchmesser der Ergebnisintervalle in einer Größenordnung von 10^{-12} und die relativen Fehler bei 10^{-11}.

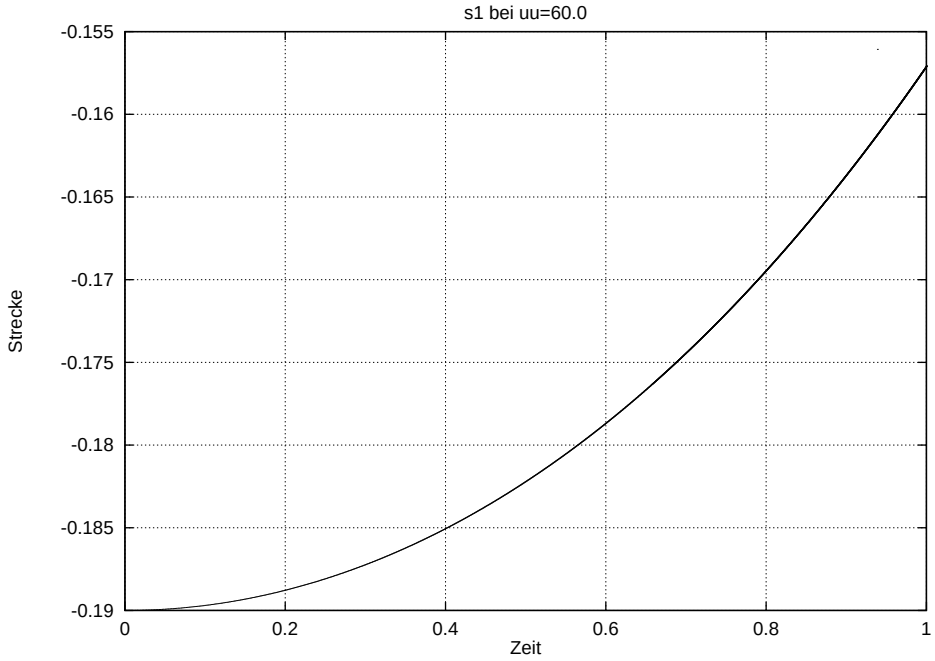


Abbildung 3: Funktionsplot der Differentialgleichungs-Lösung bei `uu=60.0`

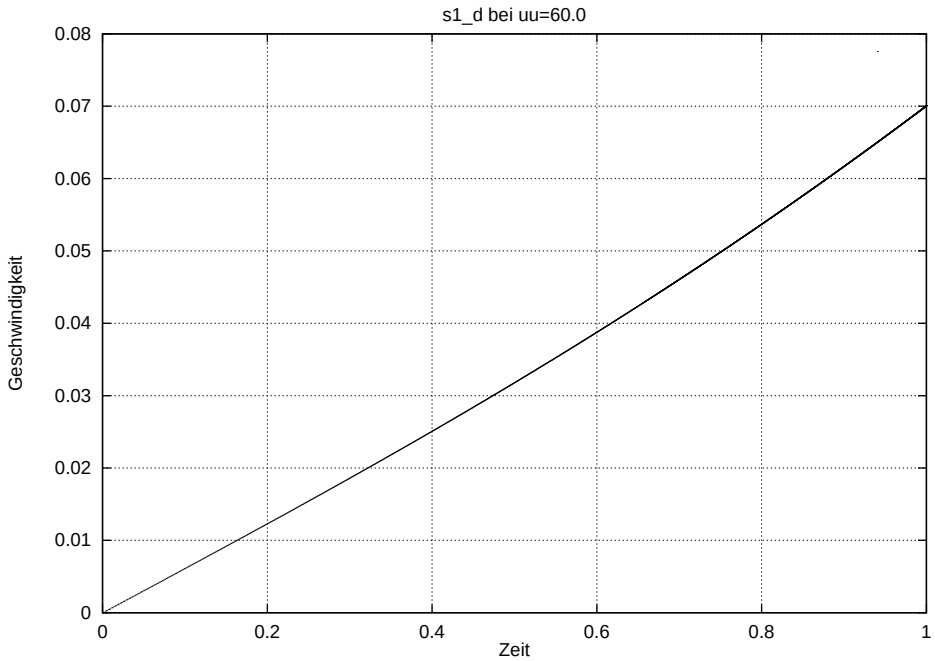


Abbildung 4: Funktionsplot der Ableitung der Differentialgleichungs-Lösung bei `uu=60.0`

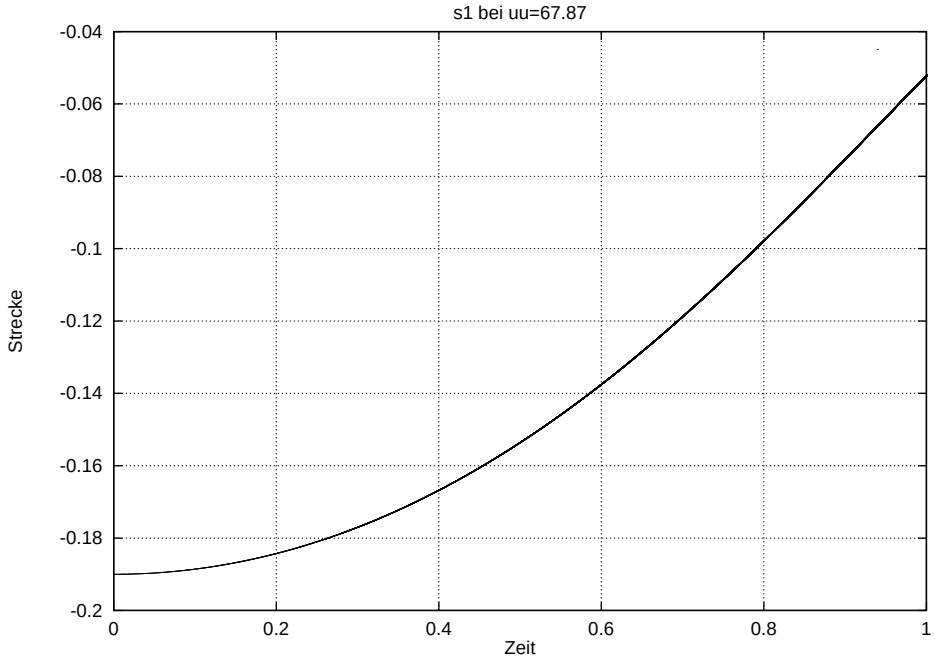


Abbildung 5: Funktionsplot der Differentialgleichungs-Lösung bei `uu=67.87`

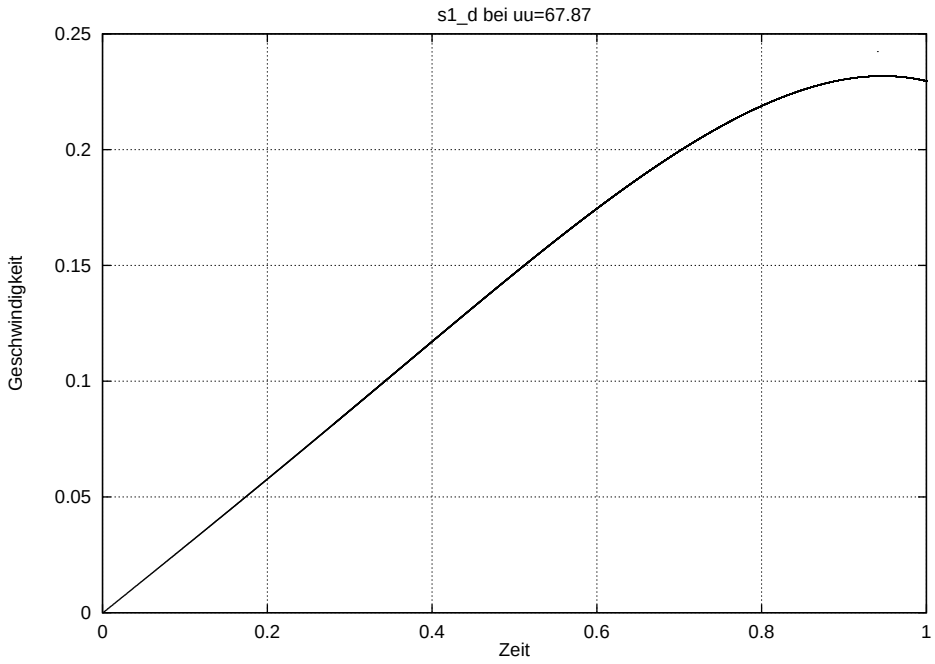


Abbildung 6: Funktionsplot der Ableitung der Differentialgleichungs-Lösung bei `uu=67.87`

7 Schlußbemerkungen

Entsprechend den Aufgaben des Teilprojektes A8 haben wir im Rahmen des SFB-Seminars anhand von ausgewählten Problemen aus der Modellierung von Manipulatoren Lösungen verifiziert berechnet und Aussagen über das Verhalten der Systeme bei unscharfen Eingangsgrößen gemacht. Wir haben Wege aufgezeigt, wie mit MOBILE erzielte Ergebnisse mit Hilfe von verschiedenen Tools validiert und verifiziert werden können. Hierzu mußten die Problemstellungen in die richtige Form für die neuen Verifikationstools überführt werden. Eine automatische Konvertierung ist hier von Relevanz. Die Beispiele haben gezeigt, daß es zunächst nicht realistisch ist, Verifikationstools auf der Basis von AWA in die MOBILE-Umgebung zu integrieren. Die in MOBILE vorhandenen Differentialgleichungslöser lassen allerdings noch keine Intervall- und Fehlerarithmetik zu. Zur Fehlerverfolgung und bei Gleichungslösern sind die vorgestellten Verifikationstools geeignet. Sie dafür in MOBILE zu integrieren, so daß direkt Intervall- und Fehlerrechnungen durchgeführt werden können, ist ein gangbarer Weg, wie die im Seminar erzielten Ergebnisse aufzeigen konnten.

A Quelltexte der Programme

Mit dem folgenden optimierten AWA-Programm wurde das Beispiel aus Abschnitt 6 dieser Arbeit gerechnet:

```
Gravity = 9.81
v1110x = 0.42
v12x   = 0.07
v12y = -0.1395
v12z = 0.001
v45x = 0.372
v67x = 0.2
v1012x = 0.5
M12    = 1.0
I12z = 0.0
uu = 60.0

ADH_3_1_1 = sqr( v12x) + sqr( v12y) + sqr( v12z)
ADH_4_1_1 = sqr( v1110x)
ADH_5_1_1 = sqr( U1 + v45x + v67x)
A_1 = - 2 * v1110x * v12x
B_1 = - 2 * v1110x * v12y
C_1 = - ADH_3_1_1 - ADH_4_1_1 + ADH_5_1_1
A_1qplusB_1q = sqr(A_1) + sqr(B_1)
WURZEL = sqrt( A_1qplusB_1q - sqr(C_1))
SINTHETA1 = (B_1 * C_1 - A_1 * WURZEL) / A_1qplusB_1q
COSTHETA1 = (A_1 * C_1 + B_1 * WURZEL) / A_1qplusB_1q
JACR_1_4 = - v1110x * SINTHETA1
JACR_1_5 = v1110x * COSTHETA1
ADH_6_1_4 = - v12x + JACR_1_5
```

```
ADH_6_2_4 = - v12y - JACR_1_4
N_2 = sqrt( sqr( ADH_6_2_4) + sqr (ADH_6_1_4) )
SINTHETA2 = ADH_6_2_4 / N_2
COSTHETA2 = ADH_6_1_4 / N_2
ADH_8_1_4 = ADH_6_1_4 * COSTHETA2 + ADH_6_2_4 * SINTHETA2
N_3 = sqrt( sqr(v12z) + sqr(ADH_8_1_4))
SINTHETA3 = v12z / N_3
COSTHETA3 = ADH_8_1_4 / N_3
ADH_23_1_1 = COSTHETA2 * COSTHETA3
ADH_23_2_1 = COSTHETA3 * SINTHETA2
ADH_20_1_4 = U1 + v45x + v67x
ADH_19_3_4 = - ( ADH_20_1_4 * SINTHETA3)
ADH_27_1_4 = ADH_20_1_4 * ADH_23_1_1
ADH_27_2_4 = ADH_20_1_4 * ADH_23_2_1
JACR_3_4 = - ( ADH_20_1_4 * COSTHETA2 * SINTHETA3)
JACR_3_5 = - ( ADH_20_1_4 * SINTHETA2 * SINTHETA3)
theta1_d = ( ( ADH_23_1_1 * ADH_27_1_4 + ADH_23_2_1 * ADH_27_2_4 -
  ADH_19_3_4 * SINTHETA3) *  U2) / ( ADH_6_1_4 * JACR_1_4 + ADH_6_2_4 *
  JACR_1_5)
theta2_d = ( ( - ( ADH_23_2_1 * JACR_3_4) + ADH_23_1_1 * JACR_3_5) *  U2 +
  ( JACR_1_5 * JACR_3_4 - JACR_1_4 * JACR_3_5) *  theta1_d) / ( ADH_27_1_4 *
  JACR_3_4 + ADH_27_2_4 * JACR_3_5)
theta3_d = ( - ( ADH_23_1_1 *  U2) + JACR_1_4 *  theta1_d + ADH_27_2_4 *
  theta2_d) / JACR_3_4
ADH_40_1_4 = - v1012x - v1110x
JACR_3_6 = - ( ADH_20_1_4 * ( ADH_23_1_1 * COSTHETA2 + ADH_23_2_1 *
  SINTHETA2))
Pseudo1theta1_d = ( ADH_23_1_1 * ADH_27_1_4 + ADH_23_2_1 * ADH_27_2_4 -
  ADH_19_3_4 * SINTHETA3) / ( ADH_6_1_4 * JACR_1_4 + ADH_6_2_4 * JACR_1_5)
Pseudo1DJACR_12_5 = - ( ADH_40_1_4 *  Pseudo1theta1_d)
Pseudotheta1_dd = ( - ( ADH_6_1_4 * (- ( JACR_1_5 *  theta1_d) *  theta1_d)) -
  ADH_6_2_4 * (JACR_1_4 *  theta1_d *  theta1_d) + ADH_27_1_4 *
  ((- ( ADH_23_2_1 *  theta2_d) - COSTHETA2 * SINTHETA3 *  theta3_d) *
  U2 + ( - ( ADH_23_2_1 *  U2) - ADH_27_1_4 *  theta2_d - JACR_3_5 *
  theta3_d) * theta2_d + (- ( COSTHETA2 * SINTHETA3 *  U2) - JACR_3_5 *
  theta2_d + COSTHETA2 * JACR_3_6 *  theta3_d) *  theta3_d) + ADH_27_2_4 *
  ((ADH_23_1_1 *  theta2_d - SINTHETA2 * SINTHETA3 *  theta3_d) *  U2 +
  (ADH_23_1_1 *  U2 - ADH_27_2_4 * theta2_d + JACR_3_4 *  theta3_d) *
  theta2_d + (- ( SINTHETA2 * SINTHETA3 *  U2) + JACR_3_4 * theta2_d +
  JACR_3_6 * SINTHETA2 *  theta3_d) * theta3_d) + ADH_19_3_4 *
  ((- ( ( ADH_23_1_1 * COSTHETA2 + ADH_23_2_1 * SINTHETA2) * theta3_d)) *
  U2 + (- ( ( ADH_23_1_1 * COSTHETA2 + ADH_23_2_1 * SINTHETA2) *  U2) -
  ( COSTHETA2 * JACR_3_4 + JACR_3_5 * SINTHETA2) *  theta3_d) *  theta3_d)) /
  ( ADH_6_1_4 * JACR_1_4 + ADH_6_2_4 * JACR_1_5)

F1 = U2
F2 = -1 / (M12 * sqr( Pseudo1DJACR_12_5) + I12z *  sqr( Pseudo1theta1_d)) *
  (M12 * (- ( ADH_40_1_4 *  Pseudotheta1_dd)) * Pseudo1DJACR_12_5 + I12z *
```

```
  Pseudo1theta1_d *  Pseudotheta1_dd - uu + Gravity * M12 *
  Pseudo1DJACR_12_5 * COSTHETA1)

;;

1 0.0 1.0 0.01 20

$  out       =    1, jeden Schritt ausgeben
$  T_start   =  0.0, Anfang des Integrationsintervalls
$  T_end     =  1.0, Ende des Integrationsintervalls
$  h_initial = 0.01, anfaengliche Schrittweite von AWA
$  p         =   20, Ordnung der Methode

4 0

$ 4 : Einschliessungsmethode 4, d. h. Schnitte von Intervallvektoren
$     und QR-Zerlegung
$ 0 : Ausgabe des Einschliessungsintervalls der Funktionswerte der Loesung

-0.19 0.0

$ -0.19 0.0 : Anfangswerte fuer beide Komponenten der Loesung
$             u (0) = -0.19
$             u'(0) = 0.0

n

$ n : keine Ausgabe der Ueberschaetzung der Loesungsmenge

1e-12 1e-12

$ 1e-12 1e-12   : Fehlertoleranzen E_a = 1e-12 und E_r = 1e-12

n
n
n

$ n : nicht weiter integrieren
$ n : keine anderen Anfangswerte
$ n : keine andere Differentialgleichung
```

Die Berechnungen für das Beispiel aus Abschnitt 4.1 wurden mit folgendem PASCAL XSC-Programm vorgenommen:

```
program gesamtausgang(messdaten,messout);

use
```

```
  i_ari, abs_ari;

const
  c_ny = 1.3;
  c_c = 4;
  c_g = 2;
  c_yn = 1;
  c_gyn = 3;   {c_g + c_yn}
  c_gyn1 = 4; {c_g + c_yn + 1}

type
  vecc = array[1..c_c] of BoundType;
  vecv = array[1..c_gyn] of BoundType;
  veckonklu = array[1..c_gyn1] of BoundType;
  matv = array[1..c_c,1..c_gyn] of BoundType;
  matkonklu = array[1..c_c,1..c_gyn1] of BoundType;
  veclese = array[1..4] of BoundType;

  modelrec = record
               c : integer;
               v : matv;
               konklu : matkonklu;
               g : integer;
               ny : BoundType;
               yn : integer;
             end;

var
  modell : modelrec;
  x : vecv;
  Ausgang : BoundType;
  messdaten : file of veclese;
  messout : text;
  u : array[1..4] of Boundtype;
  lese : veclese;

function Sqr(x : BoundType) : BoundType;
{ Quadriert eine BoundType-Variable }
var tmp : BoundType;
begin
  tmp := x * x;
  if inf(tmp.enclosure) < 0 then
    tmp.enclosure := intval(0.0 ,sup(tmp.enclosure));
  sqr := tmp;
end;

function exp(x : BoundType) : BoundType;
{ e-Funktion von einer BoundType-Variablen }
```

```
begin
  exp.Enclosure := exp(x.enclosure);
  exp.AbsErr := Eps53 *> MaxAbs(exp(x.enclosure)) +> (1 +> eps52) *> x.AbsErr
                *> Maxabs(exp(x.enclosure + intval(-x.abserr, x.abserr)))
end;

function ln(x : BoundType) : BoundType;
{ natuerlicher Logarithmus einer BoundType-Variablen }
begin
  ln.Enclosure := ln(x.enclosure);
  ln.AbsErr := Eps53 *> MaxAbs(ln(x.enclosure)) +> (1 +> eps52) *> x.AbsErr
                *> Maxabs(1.0/(x.enclosure + intval(-x.abserr, x.abserr)))
end;

function muprob(i: integer; ny : BoundType; c: integer ; d : vecc) : BoundType;
{ probabilistische Zugehoerigkeitsfunktion - gehoert zu gesamtausgang }
var
  d_null : boolean;
  j : integer;
  nenner : BoundType;
begin
  d_null := false;
  for j := 1 to c do
    d_null := d_null OR (0 in d[j].Enclosure);
  if 0 in d[i].enclosure then
    muprob :=1.0
  else begin
     if d_null then
       muprob := 0.0
     else begin
       nenner := 1.0;
       for j := 1 to c do
         if j<>i then
   nenner := nenner + exp( ln(d[i]/d[j]) * (2/(ny-1)) );
       muprob := 1/nenner
     end
   end
end;

function ausgang_teilmodell(x : vecv; konklu_i : veckonklu; g : integer;
                            yn: integer) : BoundType;
{ Ausgang eines Teilmodells - gehoert zu gesamtausgang }
var
  summe : BoundType;
  i : integer;
begin
  summe := konklu_i[1];
  for i := 1 to yn do
```

```
    summe := summe - konklu_i[i+1] * x[i];
  for i := yn+1 to (g+yn) do
    summe := summe + konklu_i[i+1] * x[i];
  ausgang_teilmodell := summe
end;

function gesamtausgang(x : vecv; model : modelrec) : BoundType;
{ Berechnung des Gesamtausgangs eines FIMO Modells }
var
  zeile_i : veckonklu;
  d : vecc;
  i, j : integer;
  tmp, summe : BoundType;
begin
  for i := 1 to model.c do begin
    tmp := 0.0;
    for j := 1 to (model.yn+model.g) do
      tmp := tmp + sqr(x[j] - model.v[i,j]);
    d[i] := sqrt(tmp)
  end;
  summe := 0.0;
  for i:=1 to model.c do begin
    for j := 1 to model.yn + model.g + 1 do
      zeile_i[j] := model.konklu[i,j];
    summe := summe + muprob(i,model.ny,model.c,d) *
             ausgang_teilmodell(x,zeile_i,model.g,model.yn)
  end;
  gesamtausgang := summe
end;

begin
{ Initialisierung der Modellparameter }
  modell.c := c_c;
  modell.g := c_g;
  modell.ny.Enclosure := c_ny;
  modell.yn := c_yn;

  modell.konklu[1,1].Enclosure := 7.5636199e-05 ;
  modell.konklu[1,2].Enclosure := -9.9999398e-01;
  modell.konklu[1,3].Enclosure := -1.5416656e-11;
  modell.konklu[1,4].Enclosure :=  5.7488025e-05;

  modell.konklu[2,1].Enclosure := -9.5874304e-05;
  modell.konklu[2,2].Enclosure := -1.0012091e+00;
  modell.konklu[2,3].Enclosure := 3.0908273e-12;
  modell.konklu[2,4].Enclosure := 9.3164366e-05;

  modell.konklu[3,1].Enclosure := 1.5524187e-05;
```

```
  modell.konklu[3,2].Enclosure := -9.9974215e-01;
  modell.konklu[3,3].Enclosure := -4.4652061e-13;
  modell.konklu[3,4].Enclosure := 8.3244391e-05;

  modell.konklu[4,1].Enclosure := -9.9907425e-05;
  modell.konklu[4,2].Enclosure := -1.0001661e+00;
  modell.konklu[4,3].Enclosure := 6.6502200e-11;
  modell.konklu[4,4].Enclosure := 8.2845829e-05;

  modell.v[1,1].Enclosure :=   1.0474199e-01;
  modell.v[1,2].Enclosure :=   5.4239085e+06;
  modell.v[1,3].Enclosure :=   -4.5641599e+00;

  modell.v[2,1].Enclosure := 7.3378302e-02;
  modell.v[2,2].Enclosure := 3.1518045e+06;
  modell.v[2,3].Enclosure := 2.2579128e-01;

  modell.v[3,1].Enclosure := 7.3225103e-02;
  modell.v[3,2].Enclosure := 2.1645755e+06;
  modell.v[3,3].Enclosure := 4.0656880e-01;

  modell.v[4,1].Enclosure := 1.1336200e-01;
  modell.v[4,2].Enclosure := 1.3875264e+06;
  modell.v[4,3].Enclosure := 4.6694999e+00;

{ Einlesen der Messdaten und Berechnung der Position }

  reset(messdaten);
  rewrite(messout);
  read(messdaten,lese);
  u[1]:=lese[4];
  read(messdaten,lese);
  u[2]:=lese[4];
  read(messdaten,lese);
  u[3]:=lese[4];
  read(messdaten,lese);
  u[4]:=lese[4];
  x[1] := lese[1];
  x[2] := lese[3];
  x[3] := u[1];
  ausgang := gesamtausgang(x,modell);
  writeln(ausgang.Enclosure);
  while not eof(messdaten) do begin
    u[1] := u[2];
    u[2] := u[3];
    u[3] := u[4];
    read(messdaten,lese);
    u[4] := lese[4];
```

```
    x[1] := ausgang;
    x[2] := lese[3];
    x[3] := u[1];
    ausgang := gesamtausgang(x,modell)
  end;
  writeln(ausgang.Enclosure);
  writeln(ausgang.AbsErr)
end.
```

Literatur

[1] Alefeld, G. ; Herzberger, J.: *Introduction to interval computations*. New York : Academic Press, 1983. – ISBN 0–12–049820–0

[2] Bantle, A. ; Krämer, W.: Ein Kalkül für verläßliche absolute und relative Fehlerschätzungen / Universität Karlsruhe. 1998 (98/5). – Preprint. Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung

[3] Bauch, H.: Zur iterativen Lösungseinschließung bei Anfangswertproblemen mittels Intervallmethoden. In: *Z. Angew. Math. Mech.* 60 (1980), S. 137–145. – ISSN 0044–2267

[4] Eiermann, M. C.: *Adaptive Berechnung von Integraltransformationen mit Fehlerschranken*, Universität Freiburg, Dissertation, 1989

[5] Eijgenraam, P.: *Mathematical Centre tracts*. Bd. 144 : The solutions of initial value problems using interval arithmetic. Amsterdam : Mathematisch Centrum, 1981. – ISBN 90–6196–230–7

[6] Hofschuster, W. ; Krämer, W.: Eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-Double-Format / Universität Karlsruhe. 1998 (98/7). – Preprint. Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung

[7] Kecskeméthy, A.: *Objektorientierte Modellierung der Dynamik von Mehrkörpersystemen mit Hilfe von Übertragungselementen*, Universität Duisburg, Dissertation, 1993

[8] Kecskeméthy, A. ; Krupp, T. ; Hiller, M.: Symbolic Processing of Multiloop Mechanism Dynamics Using Closed-Form Kinematics Solutions. In: *Multibody system dynamics* 1 (1997), Nr. 1, S. 23–45. – ISSN 1384–5640

[9] Klatte, R. ; Kulisch, U. ; Neaga, M. ; Ratz, D. ; Ullrich, C.: *PASCAL-XSC*. Berlin Heidelberg : Springer, 1992. – ISBN 3–540–55137–9

[10] Krämer, W.: Die Berechnung von Standardfunktionen in Rechenanlagen. In: Chatterji, S. D. (Hrsg.) ; Fuchssteiner, B. (Hrsg.) ; Kulisch, U. (Hrsg.) ; Liedl, R. (Hrsg.) ; Purkert, W. (Hrsg.): *Jahrbuch Überblicke Mathematik 1992*. Braunschweig : Vieweg, 1992. – ISSN 0172–8512, S. 97–115

[11] Krämer, W.: A Priori Worst Case Error Bounds for Floating-Point Computations. In: *IEEE Trans. Comput.* 47 (1998), Nr. 7, S. 750–756. – ISSN 0018–9340

[12] KROLL, A.: *Fuzzy-Systeme zur Modellierung und Regelung komplexer technischer Systeme*, Universität Duisburg, Dissertation, 1997

[13] KRÜCKEBERG, F.: Ordinary Differential Equations. In: HANSEN, E. R. (Hrsg.): *Topics in interval analysis*. Oxford : Clarendon Press, 1969, S. 91–97

[14] LOHNER, R.: Enclosing the solutions of ordinary initial and boundary value problems. In: KAUCHER, E. (Hrsg.) ; KULISCH, U. (Hrsg.) ; ULLRICH, C. (Hrsg.): *Computerarithmetic: scientific computation and programming languages*. Stuttgart : Teubner, 1987. – ISBN 3–519–02448–9, S. 255–286

[15] LOHNER, R.: *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*, Universität Karlsruhe, Dissertation, 1988

[16] LOHNER, R.: Einschließungen bei Anfangs- und Randwertaufgaben gewöhnlicher Differentialgleichungen. In: KULISCH, U. (Hrsg.): *Wissenschaftliches Rechnen mit Ergebnisverifikation*. Braunschweig : Vieweg, 1989. – ISBN 3–528–08943–1, S. 183–207

[17] LOHNER, R.: Praktikum „Einschließung bei Differentialgleichungen". In: KULISCH, U. (Hrsg.): *Wissenschaftliches Rechnen mit Ergebnisverifikation*. Braunschweig : Vieweg, 1989. – ISBN 3–528–08943–1, S. 209–223

[18] LOHNER, R.: Computation of Guaranteed Enclosures for the Solutions of Ordinary Initial and Boundary Value Problems. In: CASH, J. R. (Hrsg.) ; GLADWELL, I. (Hrsg.): *Computational ordinary differential equations*. Oxford : Clarendon Press, 1992. – ISBN 0–19–853659–3, S. 425–435

[19] MOORE, R. E.: The automatic analysis and control of error in digital computation based on the use of interval numbers. In: RALL, L. B. (Hrsg.): *Error in Digital Computation* Bd. 1. New York : John Wiley, 1965, S. 61–130

[20] REDFERN, D.: *The MATLAB 5 handbook*. New York : Springer, 1998. – ISBN 0–387–94200–9

[21] RIHM, R.: *Über Einschließungsverfahren für gewöhnliche Anfangswertprobleme und ihre Anwendung auf Differentialgleichungen mit unstetiger rechter Seite*, Universität Karlsruhe, Dissertation, 1993

[22] WOLFRAM, S.: *Mathematica: a system for doing mathematics by computer*. Addison-Wesley, 1988. – ISBN 0–201–19330–2