Dichte, parallel-iterierte Runge-Kutta-Verfahren und robuste Fehlerschranken für zusammengesetzte Differentialgleichungen

Udo Schendel Joachim Tews

Forschungsprojekt *Parallele Numerik*
Fachbereich Mathematik und Informatik
Freie Universität Berlin 1999

Vorwort

Gewöhnliche Differentialgleichungen treten in den Naturwissenschaften und technischen Anwendungen häufig auf. Weil man in den meisten Fällen mit Näherungsmethoden arbeiten muß und die numerischen Ersatzprobleme i.a. sehr komplex sind, ist man an parallelen Verfahren interessiert. Sehr oft treten Unstetigkeiten, die aus Modellwechseln herrühren, hinzu. Diese Arbeit befaßt sich mit parallelen Verfahren für unstetige Differentialgleichungen. Sie entstand in dem Forschungsprojekt "Parallele Numerik" unter Leitung von Herrn Prof. Dr. Schendel.

Udo Schendel und Joachim Tews im April 1999

Inhaltsverzeichnis

1 Zusammenfassung

Gegenstand dieser Arbeit ist die mathematische Aufarbeitung und algorithmische Darstellung von **zusammengesetzten Differentialgleichungen** und deren parallele Behandlung. Hierbei wird Wert auf eine exakte und elementare, anwendungsorientierte Formulierung gelegt.
Ausgehend von einem **Basisproblem** wird ein Anfangswertproblem für **zusammengesetzte Differentialgleichungen (Problem D)** formuliert. Eine Klasse von gewöhnlichen **unstetigen Differentialgleichungen** läß sich passend einordnen.
Testbeispiele und Bemerkungen zur **automatischen Integration** bereiten den algorithmischen Teil vor.
Die Entwicklung paralleler Algorithmen gliedert sich in die Bereiche:

- parallel iterierte Runge-Kutta-Verfahren
- dichte Einschrittverfahren
- Schrittweitenregelung
- Fehlerschätzer

Fehlerschätzungen verdienen besondere Aufmerksamkeit, weil nur durch robuste/verläßliche **Fehlerschranken** eine korrekte Problemlösung gewährleistet werden kann.
Desweiteren werden Details der parallelen Programmierung kurz beschrieben. Hierbei werden die verschiedenen **Ebenen der Parallelität und der Adaptivität** erläutert.

Die Querverweise der Form K.A.N**S**S innerhalb dieses Textes enthalten die Kapitelnummer K, die Abschnittsnummer A, eine fortlaufende Nummer N und die Seitenzahl S.

2 Problemstellung

2.1 Grundlagen

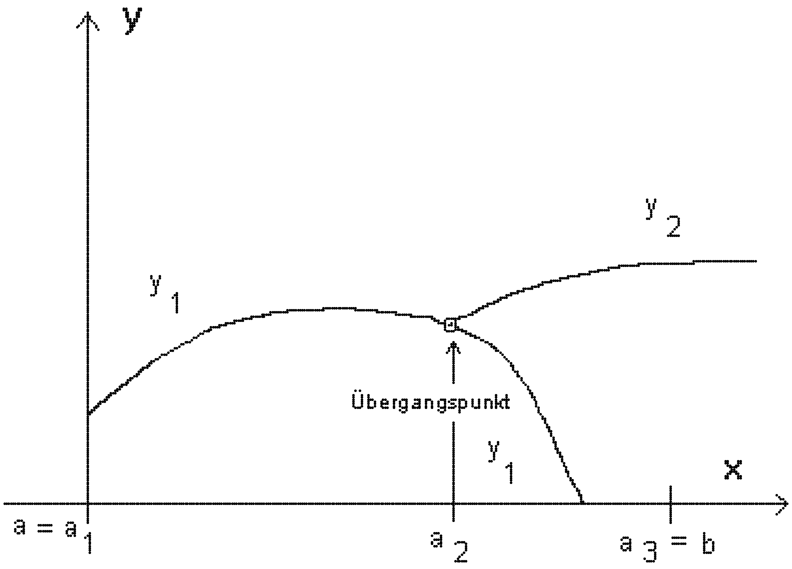
Im allgemeinen werden nur gängige Bezeichnungen verwendet. Alle auftretenden Größen sind, wenn nicht anders angegeben, reell. Die Argumente von Funktionen werden teilweise weggelassen, wenn sie offensichtlich sind. Der Riemannsche Integralbegriff wird durchgängig verwendet.

Die Querverweise der Form K.A.N**S**S innerhalb dieses Textes enthalten die Kapitelnummer K, die Abschnittsnummer A, eine fortlaufende Nummer N und die Seitenzahl S.
Auf die notwendigen Grundlagen wird an den entsprechenden Stellen verwiesen.

2.2 Zusammengesetzte Differentialgleichungen

Die Grundidee der Zusammensetzung von Differentialgleichungen läßt sich am einfachsten anhand der folgenden Grafik einsehen:

Abbildung 2.2.1 (Zusammensetzung) :



Ausgehend vom Anfangspunkt a_1 wird zuerst eine Differentialgleichung für y_1 gelöst. Unter geeigneten Voraussetzungen (siehe 2.2.5S5) besitzt das erste Anfangswertproblem (siehe z.B. [Werner/Arndt]) eine eindeutige Einzellösung. Nachdem nun $y_1(a_2)$ bekannt ist, wird ein zweites Anfangswertproblem (**Basisproblem**) für y_2 gelöst. Die Anfangsbedingung für y_2 ist gerade dadurch gegeben, daß y_1 und y_2 bei a_2 übereinstimmen.
Die Zusammensetzung läßt sich in analoger Form fortsetzen. Die **Übergangspunkte** $a_2, a_3, \ldots$ müssen nicht a priori bekannt sein, sondern können durch **Schaltfunktionen** im Verlauf der Rechnung definiert werden. Beispiel 2.2.2S3 und Abbildung 2.2.3S4 verdeutlichen dies, bevor wir uns den formalen Definitionen zuwenden.

Beispiel 2.2.2 (Joghurt) :

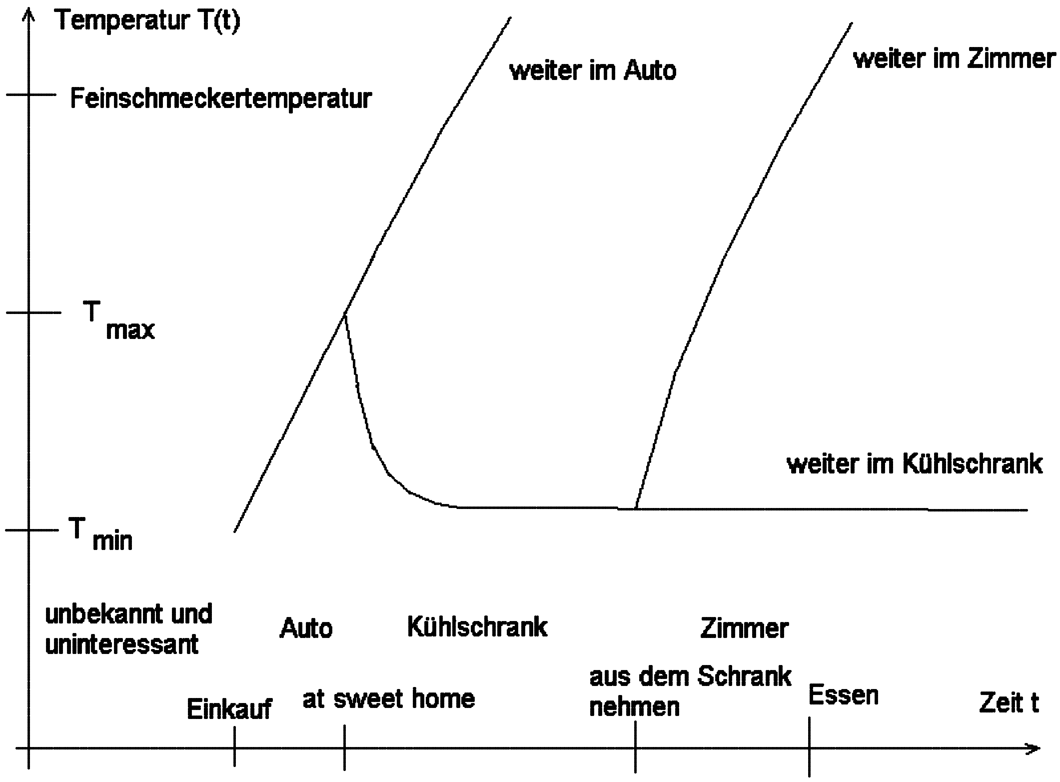
Wir betrachten das Problem, das sich im Sommer beim Einkauf verderblicher Waren ergibt. Ein frisch gekaufter Joghurtbecher erwärmt sich im Auto gemäß dem Newtonschen Temperaturausgleichsgesetz (siehe [Heuser] Seite 74)
$T'(t) = k_{\text{Joghurt/Auto}} * (T_{\text{Auto}} - T(t))$
solange, bis er die Temperatur des Autos erreicht hat.
Sobald der Becher im Kühlschrank steht, wird er bis auf die Kühlschranktemperatur abgekühlt : $T'(t) = k_{\text{Joghurt/Schrank}} * (T_{\text{Schrank}} - T(t))$.
Wir wollen den Temperaturbereich (üblicherweise 5° C bis 10° C) beachten, der die Haltbarkeit des Joghurts garantiert, und fragen:
Wie lange können wir den Joghurt spazierenfahren ?
Wie lange dauert es, bis er sich wieder abgekühlt hat ?
Außerdem wollen wir wissen, wie lange es dauert, bis der Joghurt die angenehme Feinschmeckertemperatur von $15°C$ erreicht hat.

Beispiel 2.2.2S3 besteht aus drei nacheinander zu lösenden Differentialgleichungen. Als **Schaltfunktionen** lassen sich

1. $s_{\text{Auto}}(t,T) = T - T_{\max}$
2. $s_{\text{Schrank}}(t,T) = t - t_{\text{aus dem Schrank nehmen}}$
3. $s_{\text{Zimmer}}(t,T) = T - T_{\text{Feinschmecker}}$

verwenden. Die **Übergangspunkte** zwischen den einzelnen Differentialgleichungen sind gerade die uns interessierenden Zeitpunkte. Die **Schaltfunktionen** besitzen genau an diesen Punkten Nullstellen mit Vorzeichenwechsel. Betrachte hierzu Abbildung 2.2.3S4:

Abbildung 2.2.3 (Temperaturverlauf im Joghurt) :



Wie wir später noch sehen werden, erleichtern uns die **Schaltfunktionen** sowohl die formale Definition, als auch den numerischen Lösungsprozeß.

Im Joghurt-Beispiel 2.2.2S3 wurden die folgenden Punkte idealisiert:

1. Die Joghurttemperatur wird als homogen angenommen oder es wird ein Mittelwert betrachtet; in jedem Fall wird die Wärmeleitung im Joghurt vernachlässigt.
2. Entsprechendes trifft auf die Umgebung zu.
3. Der Einfluß von isolierenden Schichten (z.B. Verpackung) wird vernachlässigt.
4. Der Wechsel von einem Ort (Auto, Kühlschrank, Zimmer) zum anderen erfolgt in einem Zeitpunkt.
5. Gährungsprozesse im Joghurt bleiben unberücksichtigt.

Nun können wir uns an die Formalia wagen. Um die Betrachtungen möglichst einfach zu halten und das wesentliche herauszuarbeiten, beschränken wir uns auf sehr gutartige Fälle. Hierzu definieren wir zunächst die folgende Abkürzung:

Definition 2.2.4 (Abkürzung "bod") :

Die Abbbildung $F : [a,b] \longrightarrow \mathbb{R}^n$ werde abgekürzt "bod" genannt genau dann, wenn sie im offenen Interval (a,b) "beliebig oft differenzierbar" ist und darüberhinaus alle rechtsseitigen Ableitungen bei a und alle linksseitigen Ableitungen bei b besitzt.
Man beäuge auch die Beispiele ab Seite 16.

Die in Definition 2.2.4S5 angegebenen Bedingungen sind äquivalent zur Charackterisierung der Elemente des Funktionenraumes $C^\infty([a,b])$.

Das zu behandelnde Basisproblem besitzt die Gestalt (wir verallgemeinern gleich auf Gleichungssysteme):

Definition 2.2.5 (Basisproblem) :

Betrachte das Anfangswertproblem für ein System linearer,
gewöhnlicher Differentialgleichungen der Dimension n:
$y'(x) = f(x,y) = A(x) * y(x) + B(x)$ für $x \in [a,b]$ und $y(a) = y_0 \in \mathbb{R}^n$.
Es wird vorausgesetzt, daß $B : [a,b] \longrightarrow \mathbb{R}^n$ bod (siehe Def. 2.2.4S5) ist.
Gleiches gelte für den variablen Koeffizienten $A : [a,b] \longrightarrow \mathbb{R}^{n\times n}$.

Der matrixwertige Koeffizient $A(x)$ und die Inhomogenität $B(x)$ können variabel sein; alle Beispiele in dieser Arbeit besitzen allerdings konstante Koeffizienten.
Die oben angegebenen Voraussetzungen garantieren die eindeutige Lösbarkeit des Basisproblems.

Satz 2.2.6 (Eindeutige Lösbarkeit des Basisproblems) :

Vor.: Vorgelegt sei ein Basisproblem 2.2.5S5.
Beh.: Es besitzt eine eindeutige **Einzellösung** y.
Bew.: siehe Satz 2.4.3 auf Seite 66 in [Werner/Arndt]
oder Satz 56.1 auf Seite 501 in [Heuser].qed

In den Beweis dieser Aussage geht wesentlich die Linearität der Differentialgleichung (eigentlich des Differentialoperators) bzw. die Eigenschaft, daß die rechte Seite affin in y ist, ein.
Bei nichtlinearen Differentialgleichungen ist das Interval, in dem die Lösung existiert, i.a. nicht a priori bekannt.
Trivial ist die folgende Glattheitseigenschaft der **Einzellösung**.

Lemma 2.2.7 (Einzellösung ist bod) :

Vor.: Vorgelegt sei ein Basisproblem 2.2.5S5.
Beh.: Die **Einzellösung** y ist bod (siehe 2.2.4S5).
Bew.: Idee: Vollständige Induktion unter Ausnutzung der
Eigenschaft, daß $A(x)$ und $B(x)$ bod sind. $\diamondsuit$

Durch Zusammensetzung einzelner **Basisprobleme** erhält man ein spezielles zusammengesetztes Differentialgleichungssystem:

Definition 2.2.8 (Zusammengesetzte Differentialgleichungen) :
Seien die Basisprobleme $\mathcal{B}_k$ für $k \in \{1, \ldots, m\}$ mit $m > 1$ gegeben.
Jedes Basisproblem $\mathcal{B}_k$ besitzt die Form 2.2.5S5.
Weiter gelte $\forall\ k : A_k, B_k$ sind sogar auf $[a, b] := \bigcup_{l=1}^{m} [a_l, b_l]$ bod
(siehe Def. 2.2.4S5) und $a_{k+1} = b_k > a_k \wedge\ y_{k+1,0} = y_k(b_k) \wedge a_1 = a$.
Es sind also m rechte Seiten und m Einzellösungen auf dem unterteilten Interval $[a, b]$ definiert und nur $y_{1,0}$ ist beliebig.

Nimm nun an, daß a_2 bis a_m aus Definition 2.2.8S6 nicht a priori bekannt sind; stattdessen werden sie mit Hilfe von **Schaltfunktionen** definiert:

Definition 2.2.9 (Schaltfunktion) :
Seien $a \leq \alpha < b \in \mathbb{R}$ und die Abbildung $y : \ [a, b] \longrightarrow \mathbb{R}^n$ gegeben.
$s : \ [a, b] \times \mathbb{R}^n \longrightarrow \mathbb{R}$ ist genau dann eine Schaltfunktion bzgl.
(a, b, α, y), wenn für $N : \ [a, b] \longrightarrow \mathbb{R}$ mit $N(x) := s(x, y(x))$ gilt :
N besitzt eine ausgezeichnete Nullstelle mit Vorzeichenwechsel, genauer

$$\exists\ z \in (\alpha, b) : (\ N(z) = 0 \wedge \forall\ x \in (\alpha, z) : N(x) < 0 \\ \wedge \exists\ \delta > 0\ \forall\ x \in (z, z + \delta) : N(x) > 0\ ).$$

Mit Hilfe der **Schaltfunktionen** lassen sich Differentialgleichungen zusammensetzen:

Definition 2.2.10 (Problem D) :
Definiere induktiv für $1 \leq k < m$:

1.) y_k ist die nach Satz 2.2.6S5 eindeutige **Einzellösung** des Anfangswertproblems des linearen gewöhnlichen Differentialgleichungssystems

$$y_k'(x) = A_k(x) * y_k(x) + B_k(x) \text{ für } x \in [a, b] \text{ und } y_k(a_k) = y_{k,0}.$$

2.) Vorgelegt werde die k. **Schaltfunktion** s_k bzgl. (a, b, a_k, y_k).
$s_k(x, y_k(x))$ besitzt eine ausgezeichnete Nullstelle z_k;
siehe Definition 2.2.9S6.

3.) Setze $a_{k+1} := z_k$ und $y_{k+1,0} := y_k(a_{k+1})$.

Wir halten den wesentlichen Zusammenhang noch einmal fest.

Bemerkung 2.2.11 (Zusammengesetzte D.-G., Schaltfkt., Problem D) :
Problem D (Definition 2.2.10S6) läßt sich als eine Zusammensetzung von einzelnen **Basisproblemen** (Definition 2.2.5S5) analog zu Definition 2.2.8S6 auffassen.
Die einzelnen **Übergangspunkte** (siehe Definition 2.2.12S7) werden durch ausgezeichnete Nullstellen mit Vorzeichenwechsel von sich aus den **Schaltfunktionen/Einzellösungen** ergebenden Abbildungen definiert.
Weil wir im folgenden nur **Schaltfuntkionen** (Definition 2.2.9S6) betrachten, die zu einem Problem vom Typ D gehören, lassen wir den sich aus dem Zusammenhang erklärbaren Zusatz "bzgl. (a, b, α, y)" häufig weg.

Es empfiehlt sich, unsere Definition mit derjenigen von [Schaefer] (Definition 5.6 Anfangswertproblem für ein unstetiges Zustandsmodell auf Seite 80 ebenda) zu vergleichen. Wir heben hier nur das Wichtigste hervor:

1. **Problem D** ist technisch einfacher.
2. Dafür sind die Schwierigkeiten auf den Modellierungsprozeß verlagert worden, der jeder Problemstellung vorausgeht (siehe Bemerkung 2.2.18S9, Beispiel 2.3.2S11 und Abschnitt "Automatische Integration" ab Seite 13).
3. Schäfers Definition ist allgemeiner, weil zusätzlich Übergangsfunktionen verwandt werden, die auch Unstetigkeiten in der Lösung (siehe Definition 2.2.14S8) an den Übergangspunkten zulassen.

Die wesentliche Idee, die zur Verwendung von **Schaltfunktionen** führt, ist die Möglichkeit, **Übergangspunkte** zu charackterisieren.

Definition 2.2.12 (Übergangspunkt) :
Unter Bezugnahme auf **Problem D** (siehe 2.2.10S6) fassen wir einen Punkt $x \in [a,b]$ genau dann als **Übergangspunkt** auf, wenn er zur in Definition 2.2.10S6 induktiv definierten Menge $\{a_2, a_3, \ldots, a_m\}$ gehört.

Zwecks numerischer Tauglichkeit wird man meistens mehr von den Schaltfunktionen verlangen.

Definition 2.2.13 (Glatte Schaltfunktion) :
Seien $a \leq \alpha < b \in \mathbb{R}$ und die Abbildung $y : \ [a,b] \longrightarrow \mathbb{R}^n$ gegeben.
$s : \ [a,b] \times \mathbb{R}^n \longrightarrow \mathbb{R}$ ist genau dann eine glatte Schaltfunktion bzgl.
(a, b, α, y), wenn für $N : \ [a,b] \longrightarrow \mathbb{R}$ mit $N(x) := s(x, y(x))$ gilt :
1.) N ist bod (siehe Definition 2.2.4S5).
2.) $\exists \ z \in (\alpha, b) \ : \ (\ N(z) = 0 \wedge \forall \ x \in (\alpha, z) : N(x) < 0$
$\wedge \exists \ \delta > 0 \ \forall \ x \in (z - \delta, z + \delta) : N'(x) > 0 \ ).$

Weil alle in dieser Arbeit vorkommenden Schaltfunktionen glatt sind, lassen wir den Zusatz "glatt" weg.
Man beachte, daß sich die Glattheit einer Schaltfunktion nur bzgl. einer gegebenen Einzellösung beurteilen läßt. Dies ist nicht verwunderlich, weil sich bereits das Prädikat "Schaltfunktion" auf eine Einzellösung bezieht.

Die angegebene Formalisierungsstrategie gestattet die Verallgemeinerung auf Problem D-artige Fragestellungen unter folgenden Voraussetzungen:

1. Die einzelnen Basisprobleme sind auf einem gemeinsamen Intervall erklärbar.
2. Jedes Basisproblem läßt sich als eindeutig lösbares Anfangswertproblem auffassen. Dies bedeutet nicht automatisch, daß es sich um Differentialgleichungen handeln muß !
3. Die Existenz von Schaltfunktionen kann gesichert werden.

Wir überzeugen uns von der eindeutigen Lösbarkeit von **Problem D**, indem wir die Lösung definieren.

Definition 2.2.14 (Lösung von Problem D) :

Vorgelegt sei ein **Problem D** (siehe 2.2.10S6).
Die Zusammensetzung von den **Einzellösungen** $y_1, \ldots, y_m$ sehen wir als die stetige Lösung von **Problem D** an.
Dies ist nach Konstruktion möglich.

Problem D wurde so formuliert, daß sich ein dazugehöriger, serieller Algorithmus zur Lösung des Problems ablesen läßt.

Algorithmus 2.2.15 (Lösungsalgorithmus) :

Bearbeite nacheinander für $1 \leq k < m$:

1.) Löse das Differentialgleichungssystem

$$y_k'(x) = A_k(x) * y_k(x) + B_k(x) \text{ für } x \in [a, b], \text{ wobei}$$

die Anfangsbedingung $y_k(a_k) = y_{k,0}$ gegeben ist.

2.) Bestimme die nach Definition 2.2.9S6 ausgezeichnete Nullstelle z_k von $s_k(x, y_k(x))$.

3.) Setze $a_{k+1} := z_k$ und $y_{k+1,0} := y_k(a_{k+1})$.

Im Algorithmus 2.2.15S8 werden weder die Verfahren zur Lösung der linearen Differentialgleichungen noch die Methoden zur Bestimmung der Nullstellen von nichtlinearen, eindimensionalen Gleichungen festgelegt (siehe Kapitel "Algorithmen"). Wir formulieren eine für die numerische Behandlung zentrale Fragestellung als offenes Problem:

Problem 2.2.16 (Stetige Abhängigkeit) :

Ist es möglich, zu zeigen, daß die Lösung 2.2.14S8 von Problem D (2.2.10S6) stetig von den Daten (den Basisproblemen und den Schaltfunktionen) abhängt ?

Die Einführung einer Konvention erweist sich als platzsparend.

Definition 2.2.17 (Kurzschreibweise für Problem D) :

Gesucht wird die Abbildung

$$y : [a, b] \longrightarrow \mathbb{R}^n \text{ mit } y' = \left\{ \begin{array}{ccccccc} A_1 & y & + & B_1 & \text{für} & s_1(x, y) & < & 0 \\ A_2 & y & + & B_2 & \text{für} & s_2(x, y) & < & 0 \\ & & & & \cdots & & & \\ A_m & y & + & B_m & \text{für} & s_m := -1 & < & 0 \end{array} \right\} \text{ und}$$

$y(a) = y_0$.

In diesem Zusammenhang sprechen wir auch von **unstetigen Differentialgleichungen**, obwohl der Übergang von A_k zu $A_{k'}$ bzw. von B_k zu $B_{k'}$ beliebig glatt sein kann. Sehr oft werden aber Sprünge, zumindest in einer der höheren Ableitungen an den **Übergangspunkten** (siehe Definition 2.2.12S7) auftreten. Im Rahmen dieser Arbeit sind das die interessanteren Beispiele. An den **Übergangspunkten** besitzt die Lösung (siehe Def. 2.2.14S8) von **Problem D** (2.2.10S6) also nur einseitige Ableitungen.

Wir weisen auf einige Schwierigkeiten hin, die sich ergeben, wenn man sich nicht genügend mit der Suche nach einer exakten Problemstellung auseinandersetzt. In der Praxis wird zumeist folgendes Problem zusammen mit dem angegebenen Algorithmus formuliert und "gelöst":

Bemerkung 2.2.18 (Unstetige Differentialgleichung) :

Betrachte z.B. für $n = 1$ und $k = 4$:

$$y' = \left\{ \begin{array}{llllll} A_1 & y & + & B_1 & \text{für} & s_1(x,y)<0 \wedge s_2(x,y)<0 \\ A_2 & y & + & B_2 & \text{für} & s_1(x,y)<0 \wedge s_2(x,y)\geq 0 \\ A_3 & y & + & B_3 & \text{für} & s_1(x,y)\geq 0 \wedge s_2(x,y)<0 \\ A_4 & y & + & B_4 & \text{für} & s_1(x,y)\geq 0 \wedge s_2(x,y)\geq 0 \end{array} \right\} \text{ und}$$

$$y(a) = y_0 \in \mathbb{R}$$

und folgenden "Lösungsalgorithmus":

1. Setze a und y_0 in s_1 und s_2 ein.
2. Je nach Ergebnis ist die 1., 2., 3. oder 4. Gleichung gültig, d.h. wird bis zum Auftreten einer Nullstelle von den mit der Lösung verkettteten Schaltfunktionen s_1 oder s_2 gelöst.
3. Sobald eine neue Nullstelle auftritt, wird diese hinreichend genau bestimmt und anschließend diejenige Differentialgleichung gelöst, für die die entsprechenden Bedingungen erfüllt sind.
4. Dieser Vorgang wird bis zum gewünschten Integrationsendpunkt fortgesetzt.

Obwohl die Problemstellung und der Algorithmus aus Bemerkung 2.2.18S9 intuitiv klar sind (vergleiche mit **Problem D** 2.2.10S6), offenbaren sich folgende Unklarheiten:

- Bereits für den Anfangswert kann eine Nullstelle auftreten. Dann ist nicht klar, welche Gleichung gelten soll.
- Die Nullstellen können sich häufen. Dann versagt der Algorithmus insofern, als eine hinreichend genaue Bestimmung der Nullstellen wegen der endlichen Rechengenauigkeit nicht möglich ist.
- Aber auch sonst muß nicht klar sein, welche Gleichung gilt. Betrachte hierzu

Beispiel 2.2.19 (Unlösbare unstetige Differentialgleichung) :

Wähle $A_1 = A_2 = 0, B_1 = B_2 = B_3 = B_4 = 0, s_1 = 0,$
$A_3 = 1, A_4 = -0.5, s_2 = y - 2$ und $y(0) = 1,$ d.h.

$$y' = \left\{ \begin{array}{ccc} y & \text{für} & y - 2 < 0 \\ -0.5y & \text{für} & y - 2 \geq 0 \end{array} \right\}$$

Was passiert bei $x = \ln 2$?

Beispiel 2.2.19S9 stammt aus der Praxis und soll eigentlich die Temperaturkurve einer Heizung beschreiben, die ab $y = 2$ abgeschaltet wird. <u>Hier</u> fehlt eine adaequate Beschreibung des Modellwechsels zum Auschaltzeitpunkt (vergleiche [EnJaNoTho]).

2.3 Testprobleme

Die folgenden Beispiele dienen als Testprobleme bei der numerischen Lösung von **Problem D** (siehe 2.2.10S6).

Beispiel 2.3.1 (Pseudozufällige Spitzen $(c_0, c, d, \hat{s}_{-1})$**) :**
Setze $q := 33,\ m := 4 * q + 2 = 134,\ t_1 := 0$ und $\tilde{t}_m := 1$.
Wähle $c_0, c, d, \hat{s}_{-1} \in \mathbb{R}^+$ mit $d < \dfrac{1}{q+3}$. $\hat{s}_{-1}$ darf auch verschwinden.
Wähle $\tilde{t}_i \in [0,1]$ für $0 \leq i < q$ zufällig aus einer Gleichverteilung.
Wähle $\tilde{s}_i \in [0,1]$ für $0 \leq i < q$ zufällig aus einer Gleichverteilung.
Wähle $v_i \in [0,1]$ für $0 \leq i < q$ zufällig aus einer Gleichverteilung.
Berechne $t_i := d + (1-2d) * \tilde{t}_i \ \in [d, 1-d]$ für $0 \leq i < q$,

$$\hat{t}_{k+2} := t_{\text{Ganzteil von } k/4} + \frac{d}{4} * (\text{Rest von } k/4) \text{ für } 0 \leq k \leq m-3 \text{ und}$$

$$\hat{s}_i := (c_0 + c * \tilde{s}_i) * \left\{ \begin{array}{rl} 1 \text{ für } & v_i > 0.5 \\ -1 \text{ für } & v_i \leq 0.5 \end{array} \right\} \in [-(c_0+c), c_0+c] \text{ für } 0 \leq i < q.$$

Die Pseudozufälligkeit rührt daher, daß bei der Wahl ein Generator benutzt wird und außerdem sichergestellt wird, daß $\forall\ 0 \leq i \leq q-2 : t_{i+1} > t_i + d$ gilt.
Definiere $A_k := 0$ für $1 \leq k \leq m,\ B_1 := \hat{s}_{-1},\ B_m := 0$,

$$B_{k+2} := \left\{ \begin{array}{rl} 0 & \text{für Rest von } k/4 = 0 \\ \hat{s}_{\text{Ganzteil von } k/4} & \text{für Rest von } k/4 = 1 \\ -\hat{s}_{\text{Ganzteil von } k/4} & \text{für Rest von } k/4 = 2 \\ 0 & \text{für Rest von } k/4 = 3 \end{array} \right\} \text{für } 0 \leq k \leq m-3,$$

$s_k(x,y) := x - \hat{t}_k$ für $1 \leq k < m$ und $s_m := -1$.
Problemstellung:

$$\text{Suche } y : [-1,1] \longrightarrow \mathbb{R} \text{ mit } y' = A_k y + B_k = B_k \text{ für } s_k < 0, \quad 1 \leq k \leq m$$

(Kurzschreibweise siehe Konvention 2.2.17S8 !)
Anschaulich besitzt die Lösung von **Problem D** also 33 Spitzen der Breite $\frac{d}{2}$, die zufällig über das Intervall $(0,1)$ verteilt sind und nach oben oder unten zeigen. Betrachte die Abbildungen auf Seite 45.

Es sticht ins Auge, daß

1. in diesem Beispiel die **Schaltfunktionen** nur von x abhängen,
2. die einzelnen **Übergangspunkte** (siehe 2.2.12S7) a priori bekannt sind,
3. die Ableitung für die einzelnen Basisprobleme jeweils konstant ist.

Eine numerische Lösung könnte also mit dem expliziten Eulerverfahren erfolgen. Man beachte jedoch, daß es sich um ein Testproblem handelt, das zwangsweise entweder trivial oder nicht vollständig analysierbar und verifizierbar ist. Die Verifizierbarkeit der numerischen Lösung durch Vergleich mit der exakten Lösung ist aber für eine glaubwürdige Testreihe unbedingt zu fordern. Die exakte Lösung von Beispiel 2.3.1S10 ist unmittelbar klar, u.a. gilt $y(1) = \hat{s}_{-1}$.

Gleichungen, die nur von x abhängige **Schaltfunktionen** beinhalten, bereiten bei der Formalisierung keinerlei Schwierigkeiten. Anders sieht es aus, wenn mindestens eine **Schaltfunktion** von der Lösung y abhängt. Erst jetzt lohnt es sich, eine Formalisierung gemäß **Problem D** (siehe 2.2.10S6) durchzuführen.
Das nächste Beispiel entspricht example 2 (state events only) aus [Ellison]:

Beispiel 2.3.2 (Ellison) :

"Ellisons" Originalschreibweise:

$$y' = \begin{cases} -y + \sin x, & |y| < 0.5 \\ -0.5y + \sin x, & y \geq 0.5 \\ -0.2y + \sin x, & y \leq -0.5 \end{cases}$$

$y(\pi/4) = 0,\ x \in [\pi/4, 12.5]$
(Man beachte, daß der Originalartikel (siehe [Ellison]) einen Druckfehler enthält:
Die erste Zeile lautet im Original: $-y + \sin t,\ y < 0.5$,
im Widerspruch zur dritten Zeile.)
Wir bevorzugen stattdessen folgende Schreibweise gemäß 2.2.17S8:
$y' = A_k y + B_k$ für $s_k < 0, \quad 1 \leq k \leq 8$ mit

$$B_k = \sin(x)\ \forall\ k,\ A_k = \begin{cases} -1 \text{ für } k \in \{1,3,5,7\} \\ -0.5 \text{ für } k \in \{2,6\} \\ -0.2 \text{ für } k \in \{4,8\} \end{cases} \text{ und}$$

$$s_k(x,y) = \begin{cases} y - 0.5 \text{ für } k \in \{1,5\} \\ y + 0.5 \text{ für } k \in \{4\} \\ 0.5 - y \text{ für } k \in \{2,6\} \\ -0.5 - y \text{ für } k \in \{3,7\} \\ -1 \text{ für } k \in \{8\} \end{cases}.$$

Streng genommen unterscheiden sich die beiden Probleme, aber man kann vermuten, daß Ellison dieses Problem im Sinn hatte, als er obige Formuierung wählte. Die Lösungen der einzelnen **Basisprobleme** haben alle die Form:
$y_k = \beta_k e^{-\gamma_k x} + C_k * sin(x) - D_k * cos(x)$ mit $\gamma_k, C_k, D_k \in \mathbb{Q}^+, \beta_k \in \mathbb{R}$.

Insbesondere ist die exakte Loesung am Endpunkt: ca. -1.12692738
(formal berechnet mit 12 stelliger Arithmetik nach Algorithmus 2.2.15S8).

Man müßte nun noch beweisen, daß die Beispiele wohldefiniert im Sinne von **Problem D** (siehe 2.2.10S6) sind.

- Für Probleme, die nur von x abhängige **Schaltfunktionen** enthalten, ist der Beweis offensichtlich:
  Es kommt darauf an, daß die **Übergangspunkte** (siehe 2.2.12S7) verschieden sind.
- Für Ellisons Beispiel 2.3.2S11 ist ein Beweis, aufgrund der explizit bekannten Lösungen, streng formal möglich, liefert aber keine neuen Erkenntnisse.

Wie schon im Abschnitt "Zusammengesetzte Differentialgleichungen " angedeutet, ist der Praktiker daran interessiert, ein Problem zu lösen, das nicht a priori die Form 2.2.10S6 hat. Einen Ausweg aus dem beschriebenen Dilemma beschreibt der Abschnitt "Automatische Integration".

Für Effizienzuntersuchungen kann man ein Beispiel verwenden, in dem explizit die Anzahl der Rechenoperationen pro Auswertung der rechten Seite vorgebbar ist.

Beispiel 2.3.3 (Stress-CPU) :

Betrachte i.S. von **Probem D** (siehe 2.2.10S6):
$y = \sin(e^x - 1)$ genügt der Differentialgleichung
$y' = z$
$z' = e^x * (\cos(e^x - 1) - e^x * y)$

$$w' = \left\{ \begin{array}{rl} 10^7 & \text{für } \; y \geq 0 \\ -10^7 & \text{für } \; y < 0 \end{array} \right\} \text{ mit } \left. \begin{pmatrix} y \\ z \\ w \end{pmatrix} \right|_{x=0} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Wir bedienen uns hier der abkürzenden Schreibweise des Anwenders; eine formale Umsetzung gemäß **Problem D** beinhaltet keine Schwierigkeit, weil die explizite Lösung erkennen läßt, wo die Nullstellen liegen: $y(\ln(1 + k * \pi)) = 0$ für $k \in \mathbb{N}_0$.
Solange wir ein endliches, nicht zu großes Integrationsintervall (z.B. bis zur 26. Nullstelle) betrachten, beinhaltet das System keine numerischen Schwierigkeiten, abgesehen von den Unstetigkeiten.
Zusätzlich wird in die rechte Seite eine Zählschleife eingebaut:
$s \leftarrow 0$
for $i = 1$ to i_{max}
  $s \leftarrow s + \frac{3}{i}$
end for i
forget(s)
Diese rechte Seite benötigt also ca. $i_{\max}$ Additionen und Divisionen, falls $i_{\max}$ hinreichend groß ist.

Gleichzeitig erkennt man an Beispiel 2.3.3S12, wie man Nullstellen mit Vorzeichenwechsel mit Hilfe von **Unstetigkeiten** bestimmen kann:
Man definiert eine zusätzliche Gleichung für eine weitere Komponente des Lösungsvektors (w im obigen Beispiel). Diese Komponente besitzt eine stückweise konstante Ableitung mit großem Betrag. Die dazugehörige Schaltfunktion wählt man derart, daß die Unstetigkeiten gerade mit den gesuchten Nullstellen einer anderen Komponente (y im obigen Beispiel) zusammenfallen.
Dieses Verfahren läßt sich auch verwenden, um Kandidaten für lokale Extrema (Punkte mit waagerechter Tangente) einzelner Lösungskomponenten zu finden. Hierzu bestimmt man einfach die Nullstellen der Ableitungen (z im obigen Beispiel).
Die Bestimmung einer Nullstelle ohne Vorzeichenwechsel läuft auf die gerade beschriebene Extremasuche hinaus.

2.4 Automatische Integration

Die aus Anwendersicht komfortabelste Lösung besteht zweifellos darin, daß er/sie ein Modul bereistellt, daß die rechte Seite einer Differentialgleichung berechnet. Darüberhinaus will er/sie sich keine weiteren Gedanken über Schaltfunktionen, etc. machen. Betrachte hierzu Ellisons Originalschreibweise in Beispiel 2.3.2S11. Nun können zwei Fälle auftreten:

1. Das Modul liegt im Quelltext vor.
2. Das Modul ist nur als Black-Box verfügbar.

Im zweiten Fall liegt die ganze Beweislast beim Anwender selbst, d.h. er/sie muß sich darüber im klaren sein, ob das gestellte Problem sinnvoll und eindeutig lösbar ist. Im ersten Fall allerdings können die Entwickler der numerischen Software sehr viel Verantwortung übernehmen und zumindest Hilfestellung bei der Entwicklung eines korrekt gestellten Problems leisten. Hierzu wird das Modul als eine erste Form des Problems angesehen, die noch überarbeitet werden muß, um zu einem wohldefinierten Problem zu gelangen.

Eine Überarbeitungsmöglichkeit der Problemstellung liefern wir als Idee:

Problem 2.4.1 (Symbolische Schaltfunktionserstellung) :
Ist es möglich den Quelltext einer gewissen Konventionen genügenden rechten Seite einer Differentialgleichung so automatisch zu analysieren, daß Fallunterscheidungen (entweder explizit oder durch das Auftreten spezieller Funktionen (z.B. Betrag) implizit gegeben) erkannt werden und darüberhinaus hinreichend viele Schaltfunktionen generiert werden ?

Die automatische Erstellung der Schaltfunktionen allein liefert noch kein wohlgestelltes Problem (siehe Beispiel 2.2.19S9). Die Schwierigkeit besteht gerade darin, hinreichend viele Schaltfunktionen zu erzeugen und diese in der richtigen Reihenfolge anzuordnen.
Wie könnte nun aber ein Ordnungsalgorithmus aussehen ?
Es geht um die Angabe einer Konstruktion, die sicherstellt, daß zwischen zwei **Übergangspunkten** genügend viel Platz ist, d.h. das jeweils eine rechtsseitige Umgebung hinter einem **Übergangspunkt** existiert.
Den Ansatz zur automatischen Generierung von Schaltfunktionen wollen wir in dieser Arbeit nicht weiterverfolgen.

Stattdessen fassen wir den Fall 2 näher ins Auge. Da der Anwender uns nicht mit **Schaltfunktionen** belastet, können wir fröhlich drauflosintegrieren und davon ausgehen, daß wir einen **Übergangspunkt** (siehe Definition 2.2.12S7) daran erkennen, daß eine **Unstetigkeit** in der rechten Seite vorliegt. Wäre die rechte Seite an einem **Übergangspunkt** beliebig oft differenzierbar, würde das ein beliebiges numerisches Verfahren nicht stören. Da die Verantwortung beim Anwender liegt, gehen wir davon aus, daß nur endlich viele, verschiedene **Unstetigkeitsstellen** vorliegen; dann kann man davon ausgehen, daß alle Stellen den minmalen Abstand δ besitzen. Wir formulieren deshalb im Abschnitt "Probleme mit endlich vielen Unstetigkeiten" ein entsprechendes Problem mit nicht a priori bekannten **Übergangspunkten**.

Die automatische Behandlung von **Unstetigkeiten** gliedert sich in vier Schritte (vergleiche [Gear/Osterby]):

1. Erkennung der Unstetigkeit
2. Hinreichend genaue Bestimmung derselben
3. Überschreiten der Unstetigkeit
4. Restart, d.h. Umstellung des Verfahrens auf erneute Wachsamkeit gegenüber neuen Unstetigkeiten.

Wenn man hinreichend verläßliche Fehlerschranken zur Verfügung hat, dann lassen sich alle vier Punkte vereinen. Man verwendet eine einfache **Schrittweitenregelung** (siehe 3.4.1S29) kombiniert mit einer geeigneten **lokalen Fehlerschranke** (siehe 3.3.1S26 und 3.7.2S34). Die Fehlerschranke ist genau dann geeignet, wenn sie auch beim Auftreten von Unstetigkeiten <u>keinen zu kleinen</u> Fehler anzeigt.

Ein direkteres Vorgehen gemäß Punkt 1 bis 4 erlaubt i.a. eine <u>effizientere</u> Integration. Es handelt sich jedoch um ein offenes Forschungsgebiet, insbesondere Unstetigkeiten höherer Ordnung zu erkennen. Wenn die Erkennung korrekt funktioniert, läßt sich eine Bestimmung notfalls immer mit Hilfe einer Intervallschachtelung durchführen. Ein sehr <u>effizientes</u> Vorgehen ist möglich, wenn die **Schaltfunktionen** bekannt sind:

- Jede Schaltfunktion mißt den Abstand zum jeweiligen **Übergangspunkt** (siehe 2.2.12S7).
- Es liegen kontinuierliche Approximationen der Lösung vor.

Unter diesen Voraussetzungen kann man Verfahren zur Nullstellenbestimmung auf die **Schaltfunktionen** ansetzen und durch die Auswertung der kontinuierlichen Approximation i.a. Aufwand bei der Integration einsparen (vergleiche [EnJaNoTho]).
[Schaefer] erweitert dieses Konzept in dreierlei Hinsicht:

1. Er betrachtet auch Unstetigkeiten in Abbildungen der Lösung.
2. Einige Probleme, die bei der Modularisierung der rechten Seite auftreten, werden behandelt.
3. Die zusätzlichen Schwierigkeiten, die bei steifen Differentialgleichungen (bei der Verwendung von impliziten Verfahren) entstehen, werden dargestellt.

Auch [Schaefer] stellt fest, daß sich durch die Verwendung von Schaltfunktionen und Interpolationspolynomen bei komplexen Problemen Rechenzeit einsparen läßt.

Gedanken zur <u>robusten/verläßlichen</u>, automatischen Integration hingegen machen sich auch [ShaGladBra].

2.5 Probleme mit endlich vielen Unstetigkeiten

Die Definitionen dieses Abchnitts sind in vielerlei Hinsicht unbefriedigend, aber wir können den Kritiker mit der Theorie der **Differentialinklusionen** beruhigen, die er/sie ausführlich in [Aubin/Cellina] studieren kann. Eher in unserem Sinne befaßt sich auch [Mannshardt] mit dem Problem.
Das Ziel besteht darin, der Gleichung $y' = A(x)y + B(x)$ auch dann einen Sinn zu geben, wenn A oder B unstetig sind. Zur Präzisierung wird der Begriff der Unstetigkeit wiederholt. Wir beschränken unsere Betrachtungen in dieser Arbeit auf den Bereich der Unstetigkeiten 0. Ordnung.

Definition 2.5.1 (Unstetigkeit) :
Die Abbildung $B : [a, b] \longrightarrow \mathbb{R}^n$ besitzt bei $x_0 \in [a, b]$ genau dann eine Unstetigkeit, wenn
$\exists\ \epsilon > 0\ \forall\ \delta > 0\ \exists\ x \in [a, b] \cap (x_0 - \delta, x_0 + \delta) :\ \|B(x) - B(x_0)\| \geq \epsilon$
(vergleiche [Berz(2)]).

In Verallgemeinerung von Definition 2.2.4S5 befassen wir uns in diesem Abschnitt mit **prä-boden** Abbildungen.

Definition 2.5.2 (Abkürzung "prä-bod") :
Die Abbbildung $F : [a, b] \longrightarrow \mathbb{R}^n$ werde abgekürzt "prä-bod" genannt genau dann, wenn sie im offenen Interval (a, b) "beliebig oft differenzierbar" ist und darüberhinaus alle rechtsseitigen Grenzwerte der auf (a, b) definierten Ableitungsfunktionen bei a und alle linksseitigen Grenzwerte der auf (a, b) definierten Ableitungsfunktionen bei b existieren.
Diese Definition unterscheidet sich von der Klassifizierung $F \in C^\infty([a, b])$ dadurch, daß F auf $[a, b]$ (also bei a oder bei b) weder (einseitig) differenzierbar noch stetig sein muß !
Man beäuge auch die Beispiele ab Seite 16.

Der Vergleich mit Definition 2.2.4S5 läßt erkennen, worin genau die Abschwächung besteht:
Wir verzichten auf die Stetigkeit und verlangen nur noch die Existenz der Grenzwerte.
Dennoch sind prä-bode Abbildungen beschränkt; beachte hierzu den Beweis von Satz 2.5.10S17.

Vorausschauend definieren wir eine spezielle Art von nicht notwendig unstetigen Abbildungen.

Definition 2.5.3 (eu-stetige Abbildungen) :
Die Abbildung $B : [a, b] \longrightarrow \mathbb{R}^n$ werde "eu-stetig" genannt genau dann, wenn sie die folgenden Eigenschaften besitzt:
1. B besitzt entweder keine oder genau $m_B \geq 1$ Unstetigkeiten $\{u_{B,1}, \ldots, u_{B,m_B}\}$ (siehe Def. 2.5.1S15).
2. Jede Einschränkung $B|_{[c,d]}$ von B, die keine Unstetigkeiten in (c, d) enthält, ist als Abbildung **prä-bod** im Sinne von Definition 2.5.2S15.

Zur Existenz **eu-stetiger** Abbildungen sei bemerkt, daß alle **prä-boden** Abbildungen **eu-stetig** (siehe Def. 2.5.3S15) sind. Wir erinnern daran, daß alle **boden** (siehe Def. 2.2.4S5) Abbildungen **prä-bod** (siehe Def. 2.5.2S15) sind; wohingegen die umgekehrte Aussage nicht gilt.

Lemma 2.5.4 (Einordnung der Begriffe) :

Vor.: Sei die Abbildung $B : [a,b] \longrightarrow \mathbb{R}^n$ gegeben.

Beh.: Die folgende Implikationskette beschreibt die Stärke der einzelnen Begriffe für die Abbildung B:
$C^\infty([a,b])$ – Funktion $\Longleftrightarrow$ bod $\Longrightarrow$ prä-bod $\Longrightarrow$ eu-stetig.

Bew.: klar. $\diamondsuit$

Durch diesen Zusammenhang ist es leicht möglich, zusammengesetzte Beispielabbildungen zu konstruieren, die **eu-stetig** sind.

Beispiel 2.5.5 ($C^\infty([0,1])$-Funktion/bode Abbildung) :

Betrachte $A : [0,1] \longrightarrow \mathbb{R}$ mit $A(x) := \left\{ \begin{array}{l} 0 \text{ für } x \in \emptyset \\ 1 \text{ für } x \in [0,1] \end{array} \right\}$

A ist als $C^\infty([0,1])$-Funktion bod !

Beispiel 2.5.6 (prä-bode Abbildung) :

Betrachte $A : [0,1] \longrightarrow \mathbb{R}$ mit $A(x) := \left\{ \begin{array}{l} 0 \text{ für } x \in \{0\} \\ 1 \text{ für } x \in (0,1] \end{array} \right\}$

A ist zwar eine prä-bode, aber keine bode Abbildung !

Beispiel 2.5.7 (eu-stetige Abbildung) :

Betrachte $A : [0,1] \longrightarrow \mathbb{R}$ mit $A(x) := \left\{ \begin{array}{l} 0 \text{ für } x \in [0,0.5] \\ 1 \text{ für } x \in (0.5,1] \end{array} \right\}$

A ist zwar eine eu-stetige, aber keine prä-bode Abbildung !

Der Vollständigkeit halber bringen wir ein weiteres Beispiel für eine Abbildung, die nicht prä-bod ist.

Beispiel 2.5.8 (<u>Nicht</u>-prä-bode Abbildung) :

Betrachte $A : [0,1] \longrightarrow \mathbb{R}$ mit $A(x) := \left\{ \begin{array}{l} 0 \text{ für } x \in \{0\} \\ \frac{1}{x} \text{ für } x \in (0,1] \end{array} \right\}$

A ist keine prä-bode Abbildung.

Folglich ist die nächste Funktion nicht eu-stetig:

Beispiel 2.5.9 (<u>Nicht</u>-eu-stetige Abbildung) :

Betrachte $A : [-1,1] \longrightarrow \mathbb{R}$ mit $A(x) := \left\{ \begin{array}{l} 0 \text{ für } x \in \{0\} \\ \frac{1}{x} \text{ für } x \in [-1,1] \setminus \{0\} \end{array} \right\}$

A ist keine eu-stetige Abbildung.

Die **Unstetigkeiten** in **eu-stetigen** Abbildungen sind sämtlich Sprungstellen.

Satz 2.5.10 (Eigenschaften der Unstetigkeiten von eu-stetigen Abbildungen) :

Vor.: Sei die **eu-stetige** (siehe Definition 2.5.3S15) Abbildung $B : [a,b] \longrightarrow \mathbb{R}^n$ gegeben.

Beh.: 1.) Die Abbildung B ist beschränkt (insbesondere auch bei den Unstetigkeiten).

2.) Zu jeder Unstetigkeit $u_{B,q}$ von B existiert mindestens eine Komponente B_r mit $1 \leq r \leq n$ von B, die bei $u_{B,q}$ eine Sprungstelle besitzt.

Bew.: zu 1.): Die Aussage ist für alle abgeschlossenen Teilintervalle, in denen B beliebig oft differenzierbar ist, klar.
Da wir nur endlich viele, sauber getrennte, Unstetigkeiten zulassen, genügt es, das Verhalten von B an diesen Stellen zu untersuchen; wir können dann aus den Schranken für die einzelnen Teilintervalle die größte auswählen.
Also reicht es aus, das Verhalten einer prä-boden Abbildung D am Anfangspunkt a des Definitionsintervalls $[a,b]$ zu studieren.
Offensichtlich können wir uns auf den skalaren Fall zurückziehen.

$$F(x) := \left\{ \begin{array}{r} \lim\limits_{t \to a,\ t > a} D'(t) \text{ für } x \in \{a\} \\ D'(x) \text{ für } x \in (a,b) \\ \lim\limits_{t \to b,\ t < b} D'(t) \text{ für } x \in \{b\} \end{array} \right\} \text{ und } Y(x) := \int_a^x F(t)\ \mathbf{d}t$$

sind wohldefiniert.
Es gilt $\forall\ x \in (a,b) :\ Y'(x) = F(x) = D'(x)$!
Folglich unterscheiden sich Y und D im Intervall (a,b) nur um eine Konstante.
Y ist stetig auf $[a,b]$ und daher beschränkt.
Ergo ist D beschränkt und wir haben das Ziel erreicht. $\diamondsuit$

zu 2.): klar nach 1.) und [Berz(2)]. qed

Bemerkung 2.5.11 (Stückweise prä-bode Abbildungen) :

Im Sinne des Satzes kann man eu-stetige Abbildungen auch als stückweise prä-bod ansehen.

Auf den folgenden Seiten zeigen wir einige nützliche Vererbungseigenschaften.

Lemma 2.5.12 (Eu-Stetigkeit von Zusammensetzungen) :

Vor.: Seien die **eu-stetigen** (siehe Definition 2.5.3S15) Abbildungen $A : [a,b] \longrightarrow \mathbb{R}^n$ und $B : [b,c] \longrightarrow \mathbb{R}^n$ gegeben.
Definiere $C : [a,c] \longrightarrow \mathbb{R}^n$ mit $C(x) := \left\{ \begin{array}{lll} A(x) & \text{für} & x \in [a,b] \\ B(x) & \text{für} & x \in (b,c] \end{array} \right\}$

Beh.: C ist **eu-stetig** (siehe Definition 2.5.3S15).

Bew.: klar nach Bemerkung 2.5.11S17. $\diamondsuit$

Lemma 2.5.12S17 läßt sich auf die Zusammensetzung <u>endlich</u> vieler Abbildungen übertragen.

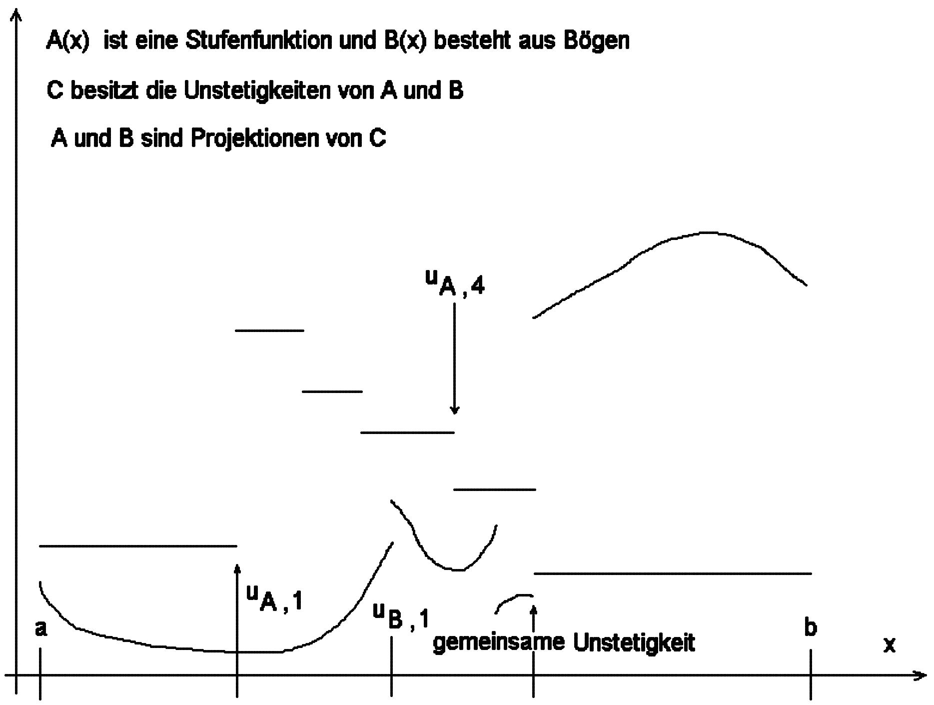
Lemma 2.5.13 (Eu-Stetigkeit der Paarabbildung) :

Vor.: Seien die **eu-stetigen** (siehe Definition 2.5.3S15) Abbildungen $A : [a, b] \longrightarrow \mathbb{R}^p$ und $B : [a, b] \longrightarrow \mathbb{R}^n$ gegeben.
Definiere die Abbildung $C := (A, B) : [a, b] \longrightarrow \mathbb{R}^p \times \mathbb{R}^n \cong \mathbb{R}^{p \cdot n}$.

Beh.: C ist **eu-stetig** (siehe Definition 2.5.3S15).

Bew.: zu 1.): zusätzliche Unstetigkeiten können wir uns nicht einfangen.
Unstetigkeiten von C sind Unstetigkeiten von A oder B.
Komponenten von A oder B sind Komponenten von C.
C kann also nur die Unstetigkeiten $\{u_{B,1}, \ldots, u_{B,m_B}, u_{A,1}, \ldots, u_{A,m_A}\}$ besitzen.
zu 2.): Sei also $C|_{[c,d]}$ eine Einschränkung von C.
Weil sie keine Unstetigkeiten im Innern des Definitionsbereichs enthält, trifft dies auch auf die Komponenten zu, die zu A gehören.
Folglich ist $A|_{[c,d]}$ prä-bod (siehe 2.5.2S15) nach Eigenschaft 2.5.3S15/2.
Analog schließen wir für $B|_{[c,d]}$.
Dann ist aber die Paarabbildung $C|_{[c,d]} = (A|_{[c,d]}, B|_{[c,d]})$ ebenfalls prä-bod; dies sieht man komponentenweise ein.
Zum Beweis des Lemmas beachte man auch die Skizze 2.5.14S18. $\diamondsuit$

Abbildung 2.5.14 (Skizze zum Beweis von Lemma 2.5.13S18) :



Auch die elementaren Verknüpfungen erhalten die **Eu-Stetigkeit**.

Lemma 2.5.15 (Eu-Stetigkeit von Summe und Vervielfachung) :

Vor.: Seien die **eu-stetigen** (siehe Definition 2.5.3S15) Abbildungen $A : [a,b] \longrightarrow \mathbb{R}^n$, $B : [a,b] \longrightarrow \mathbb{R}^n$ und $\alpha \in \mathbb{R}$ gegeben.

Beh.: $A + B$ und $\alpha \cdot A$ sind **eu-stetig** (siehe Definition 2.5.3S15).

Bew.: analog zu Lemma 2.5.13S18. $\diamondsuit$

Die Abbildung, die durch Bildung des Skalarprodukts im Bildraum entsteht, ist ebenfalls **eu-stetig**.

Lemma 2.5.16 (Eu-Stetigkeit des Bildraum-Skalarprodukts) :

Vor.: Seien die **eu-stetigen** (siehe Definition 2.5.3S15) Abbildungen $A : [a,b] \longrightarrow \mathbb{R}^n$, $B : [a,b] \longrightarrow \mathbb{R}^n$ gegeben.
Definiere $C : [a,b] \longrightarrow \mathbb{R}$ mit $C(x) := < A(x), B(x) >$.

Beh.: C ist **eu-stetig** (siehe Definition 2.5.3S15).

Bew.: analog zu Lemma 2.5.15S19. $\diamondsuit$

Ebenso folgt sofort:
Wenn die Abbildungen A und B matrixwertig sind und ihr Matrixprodukt existiert, dann überträgt sich die Eu-Stetigkeit.

Lemma 2.5.17 (Eu-Stetigkeit des Matrizenprodukts) :

Vor.: Seien die **eu-stetigen** (siehe Definition 2.5.3S15) Abbildungen $A : [a,b] \longrightarrow \mathbb{R}^{m \times n}$, $B : [a,b] \longrightarrow \mathbb{R}^{n \times p}$ gegeben.
Definiere $C : [a,b] \longrightarrow \mathbb{R}^{m \times p}$ mit $C(x) := A(x) * B(x)$.

Beh.: C ist **eu-stetig** (siehe Definition 2.5.3S15).

Bew.: analog zu Lemma 2.5.15S19. $\diamondsuit$

Die Kehrwertbildung ist, falls separiert von der Null, ebenfalls eu-stetigkeitstreu.

Lemma 2.5.18 (Eu-Stetigkeit des Kehrwerts) :

Vor.: Sei die <u>skalare</u> **eu-stetige** (siehe Definition 2.5.3S15) Abbildung $A : [a,b] \longrightarrow \mathbb{R}$ gegeben mit $\exists\, c > 0 \;\forall\, x \in [a,b] : |A(x)| \geq c$.

Beh.: $\frac{1}{A}$ ist **eu-stetig** (siehe Definition 2.5.3S15).

Bew.: analog zu Lemma 2.5.13S18. $\diamondsuit$

Hieraus ergibt sich:

Korollar 2.5.19 (Eu-Stetigkeit von rationalen Abbildungen) :

Alle <u>skalaren</u> rationalen Abbildungen sind, eingeschränkt auf abgeschlossene Intervalle ohne Definitionslücken, eu-stetig (siehe Definition 2.5.3S15). $\diamondsuit$

Auch die **Eu-Stetigkeit** der transponierten Abbildung oder des dyadischen Produkts von vektorwertigen Abbildungen sind trivial.

Zur Ernüchterung bringen wir hier auch ein Gegenbeispiel zur **Eu-Stetigkeit** der Verkettung.

Beispiel 2.5.20 (<u>Nicht</u>-Eu-Stetigkeit der Verkettung) :

Betrachte $A : [0,1] \longrightarrow \mathbb{R}$ und $B : [0,1] \longrightarrow [0,1]$ mit

$A(x) := \left\{ \begin{array}{c} 1 \text{ für } x \in [0, 0.5) \\ -1 \text{ für } x \in [0.5, 1] \end{array} \right\}$ und

$B(x) := 0.5 +$ **bode** Abbildung, die genau abzählbar unendlich viele Nullstellen mit Vorzeichenwechsel in $[0,1]$ besitzt !

$A \circ B$ ist definiert, aber nicht **eu-stetig**.

Dies alles ergibt sich aus den entsprechenden Tatsachen für **prä-bode** Abbildungen (siehe Def. 2.5.2S15) zusammen mit der komponentenweisen Berücksichtigung von <u>endlich</u> vielen **Unstetigkeiten**.

Wir ergänzen weitere strukturtheoretische Beobachtungen. Lemma 2.5.15S19 impliziert zusammen mit elementaren Überlegungen zur Lage der Unstetigkeiten, daß die **eu-stetigen** Abbildungen einen Vektorraum bilden.

Satz 2.5.21 (Vektorraum der eu-stetigen Abbildungen) :

Vor.: 1.) Definiere $V := \{ A : [a,b] \longrightarrow \mathbb{R}^n \mid A$ eu-stetig (siehe Definition 2.5.3S15) ist $\}$.

2.) Definiere die Summe und die Vervielfachung von Abbildungen wie üblich.

3.) Das Nullelement sei die konstante Nullabbildung.

Beh.: Das Quadrupel $(V, +, \cdot, 0)$ ist ein $\mathbb{R}$-Vektorraum.

Bew.: klar nach Lemma 2.5.15S19. qed

Da die **boden** Abbildungen ebenfalls einen Vektorraum bilden, gilt

Lemma 2.5.22 (Unterraum der boden Abbildungen) :

Vor.: Sei der Vektorraum der eu-stetigen Abbildungen gegeben.

Beh.: Die $C^\infty([a,b])$-Funktionen bilden einen Unterraum.

Bew.: klar nach Lemma 2.5.4S16. $\diamondsuit$

Der **Vektorraum der eu-stetigen Abbildungen** läßt sich als normierter Unterraum der beschränkten Abbildungen auffassen:

Lemma 2.5.23 (Normierter Unterraum) :

Vor.: 1.) Sei der Vektorraum V der eu-stetigen Abbildungen gegeben.

2.) Definiere die Norm eines Elementes: $\|A\| := \sup_{x\in[a,b]} \|A(x)\|_\infty$.

Beh.: 1.) $\|A\|$ ist eine Norm.

2.) V ist normierter Unterraum von $\mathcal{B}([a,b],\mathbb{R}^n) := \{ A : [a,b] \longrightarrow \mathbb{R}^n \mid A$ beschränkt$\}$.

Bew.: klar nach Satz 2.5.10S17. $\diamondsuit$

Die folgenden topologischen Überlegungen sind hilfreich, wenn man Problem 2.2.16S8 bearbeiten möchte.

Lemma 2.5.24 (Vollständigkeit von $\mathcal{B}([a,b],\mathbb{R}^n)$) :

Vor.: Sei der Raum der beschränkten Abbildungen gegeben.
Beh.: $\mathcal{B}([a,b],\mathbb{R}^n)$ ist vollständig.
Bew.: siehe [Heuser(2)], ab Seite 60. $\diamondsuit$

Vermutung 2.5.25 (Abgeschlossenheit) :

Der Grenzwert einer konvergenten Folge von eu-stetigen Abbildungen ist eu-stetig.

Gelänge der Beweis von Vermutung 2.5.25S21, so wäre der normierte Raum der eu-stetigen Abbildungen vollständig.

Satz 2.5.26 (Banachraum der eu-stetigen Abbildungen) :

Vor.: Sei der normierte Raum V der eu-stetigen Abbildungen gegeben.
Beh.: V ist Banachraum.
Bew.: abhängig von Vermutung 2.5.25S21.
V ist als abgeschlossener Unterraum ebenso vollständig wie $\mathcal{B}([a,b],\mathbb{R}^n)$ (siehe Lemmata 2.5.23S20 und 2.5.24S21).

Wir definieren eine weitere wichtige Eigenschaft einer vektorwertigen Abbildung.

Definition 2.5.27 (Eigenschaft E) :

Für eine Abbildung $C : [a,b] \longrightarrow \mathbb{R}^p$ besagt Eigenschaft E:

E. a) C ist bei a und bei b stetig.
b) C besitzt mindestens eine Unstetigkeit.
c) Zu jeder Einschränkung $C|_{(c,d)}$ von C, wobei entweder c und d Unstetigkeiten von C sind oder entweder $c = a$ und d Unstetigkeit oder $d = b$ und c Unstetigkeit ist und in jedem Fall (c,d) keine Unstetigkeiten enthält, ist eine Fortsetzung $C_{c,d} : [a,b] \longrightarrow \mathbb{R}^p$ explizit gegeben.
d) Jede dieser Fortsetzungen $C_{c,d}$ soll als Abbildung **bod** im Sinne von Definition 2.2.4S5 sein.

Nach diesen Vorbereitungen gelingt uns die versprochene Einordnung von speziellen unstetigen Differentialgleichungen:

Definition 2.5.28 (Problem EU) :

Wie in den Testbeispielen 2.3.1S10, 2.3.2S11 und 2.3.3S12 werden zunächst Hilfsgrößen und dann ein spezielles **Problem D** definiert.
Beachte die Definition der **Eu-stetgkeit** 2.5.3S15 !

1. Sei die **eu-stetige** Abbildung $A : [a, b] \longrightarrow \mathbb{R}^{n\times n}$ gegeben.
2. Sei die **eu-stetige** Abbildung $B : [a, b] \longrightarrow \mathbb{R}^{n}$ gegeben.
3. Definiere die Paarabbildung $C := (A, B) : [a, b] \longrightarrow \mathbb{R}^{n(n+1)}$.
4. C soll die **Eigenschaft E** besitzen (siehe Definition 2.5.27S21).
5. Folgerungen: C ist nach Lemma 2.5.13S18 **eu-stetig** und besitzt die Unstetigkeiten $\{u_{B,1}, \ldots, u_{B,m_B}, u_{A,1}, \ldots, u_{A,m_A}\}$.
6. Ordne die m_C Unstetigkeiten u_k von C an: $u_{k+1} > u_k$ für $1 \leq k < m_C$.
7. Setze $A_{c,d} := (C_{c,d})_{1\leq i\leq n*n}$ und $B_{c,d} := (C_{c,d})_{n*n+1\leq i\leq n(n+1)}$.

Problemstellung:
Löse **Problem D** (siehe 2.2.10S6) mit
$m := m_C + 1,\ u_0 := a,\ u_m := b,\ y(a) = y_0,$
$s_k := x - u_k$ für $1 \leq k < m,\ s_m := -1$ und
$A_k := A_{u_{k-1},u_k},\ B_k := B_{u_{k-1},u_k}$ für $1 \leq k \leq m.$

Im beschriebenen Sinne ist also ein Anfangswertproblem für unstetige Differentialgleichungen $y' = A(x)y + B(x)$ mit **eu-stetigen** Abbildungen A und B erklärt worden. Für die Praxis bedeutet dies keine Einschränkung, weil alle auf dem Rechner darstellbaren Abbildungen aus rationalen Funktionen, die nach Korollar 2.5.19S19 **eu-stetig** sind, zusammengesetzt sind. Defintion 2.5.28S22 gibt Anlaß zu

Satz 2.5.29 (Eindeutige Lösbarkeit von Problem EU) :

Vor.: siehe 2.5.28S22.
Beh.: Problem EU ist eindeutig lösbar.
Bew.: Unter den angegebenen Voraussetzungen ist **Problem EU** eindeutig lösbar, weil **Problem D** es nach Konstruktion ist, **Problem EU** ein spezielles **Problem D** ist und **Eigenschaft E** (siehe 2.5.27S21) dafür sorgt, daß die Basisprobleme 2.2.5S5 im ganzen Intervall $[a, b]$ definiert sind.qed

Man erkennt ferner, daß die Beispiele aus Abschnitt "Testprobleme" sich auch unter die Definition 2.5.28S22 einordnen:

Lemma 2.5.30 (Testprobleme vom Typ EU) :

Vor.: Seien die Testprobleme 2.3.1S10, 2.3.2S11 und 2.3.3S12 gegeben.
Beh.: Sie lassen sich als EU-Probleme auffassen.
Bew.: klar. $\diamondsuit$

Jetzt sind wir in der Lage, uns genauer mit der Lösung der gestellten Probleme zu befassen.

3 Algorithmen

3.1 Einschrittverfahren

Die Formulierung von **Problem D** suggeriert ein schrittweises Vorgehen beim Lösen der Anfangswertprobleme (vergleiche mit Algorithmus 2.2.15S8). Tatsächlich stellt sie aber keine Anforderungen an das Integrationsverfahren, weil die vorkommenden Funktionen bod (siehe 2.2.4S5) sind. Man könnte auch eine globale Diskretisierung ansetzen, lösen, den 1. Übergangspunkt (siehe 2.2.12S7) bestimmen, neu diskretisieren, usw. Der Einfachheit halber betrachten wir im folgenden nur Einschrittverfahren; die Berücksichtigung der Besonderheiten bei der Verwendung von Mehrschrittverfahren ist evident.
Zur Lösung der **Basisprobleme** 2.2.5S5 kann man Einschrittverfahren verwenden:

Algorithmus 3.1.1 (Einschrittverfahren) :

Vorgelegt sei ein Basisproblem $\mathcal{B}$.
Mit einer Verfahrensfunktion ϕ werden die diskreten Näherungen berechnet:
Seien die Schrittweiten h_l für $0 \leq l < r$ gegeben.
Berechne nacheinander:
$\tilde{x}_0 := a,\ \tilde{y}_0 := y_0.$
$\tilde{x}_{l+1} := \tilde{x}_l + h_l,\ \tilde{y}_{l+1} := \tilde{y}_l + h_l * \phi(\tilde{x}_l, \tilde{y}_l, h_l, \mathcal{B}).$

Während die Anwendung von Einschrittverfahren hier keine **zusätzlichen Voraussetzungen** benötigt, gilt dies jedoch für die effiziente Anwendung sehr wohl. Wir stellen hier kurz die **zusätzlichen Vorausetzungen** an das **Problem D** auf, die sich aus Effizienzforderungen ergeben und hier nicht im einzelnen ausführlich begründet werden sollen:

1. Weil in dieser Arbeit Stabilitätsgesichtspunkte nicht berücksichtigt werden sollen, verlangen wir, daß:
   Alle Eigenwerte der Matrizen $A_k(x)$ mit negativem Realteil sind für alle Argumente und alle $1 \leq k \leq m$ betragsmäßig kleiner als 2.

2. Der minimale Abstand zwischen zwei **Unstetigkeitsstellen** ist größer als 10^{-5}.

Die letzte Voraussetzung verdient besondere Aufmerksamkeit; wir kommen darauf im Abschnitt "Serielle Ergebnisse" zurück.

Man prüft leicht nach, daß

Lemma 3.1.2 (Vor. für Testprobleme erfüllt) :

Vor.: siehe Beispiel 2.3.1S10, 2.3.2S11 und 2.3.3S12.

Beh.: Die **Testprobleme** erfüllen die **zusätzlichen Voraussetzungen** auf Seite 23.

Bew.: klar. $\diamondsuit$

gilt. Ferner beachte man

Lemma 3.1.3 (Nicht-steife Probleme/Verfahren) :

Vor.: siehe Seite 23.

Beh.: Unter den angegebenen **zusätzlichen Vorausetzungen** lassen sich alle Einschrittverfahren mit einem beschränkten nicht-leerem Stabilitätsgebiet verwenden, sofern man die Schrittweite hinreichend klein wählt; dies ist hier jedoch für die meisten gängigen Verfahren keine echte Einschränkung.

Bew.: siehe Abschnitt 9.3.2 ab Seite 403 in [Schwarz] oder Kapitel 5 in [Werner/Arndt]. $\diamondsuit$

Wir beschränken uns also auf sogenannte **nicht-steife** Probleme und benutzen die entsprechenden Verfahren. Man könnte auch sagen, wir benutzen nur Verfahren, die die Eigenschaft **nicht-steif** besitzen.

3.2 Dichte Ausgabe von Lösungswerten

Bisher wurde noch nicht festgelegt, wie die **Übergangspunkte** (siehe Definition 2.2.12S7) bestimmt werden sollen. Zunächst benötigt man neben der disketen Lösung an den Gitterpunkten eine kontinuierliche Approximation.
Dies ist prinzipiell immer möglich, indem man zu den Stützstellen x_l und x_{l+1} und allen zur Verfügung stehenden Daten (mindestens y_l und y_{l+1}) ein Interpolationspolynom aufstellt. Die Schwierigkeit besteht darin, daß man an einer möglichst genauen Interpolation interessiert ist. Einerseits muß man möglichst viele Daten berücksichtigen und darf anderseits nur Polynome mit nicht zu hohem Grad verwenden, um die Oszillationen zu begrenzen. Man behilft sich in der Praxis damit, daß man Hermiteinterpolation mit nicht mehr als 4 Stützstellen verwendet und beschränkt damit den Grad des Polynoms auf 7. Gleichzeitig kann die kontinuierliche Approximation in diesem Fall aber nur für Einschrittverfahren der Ordnung kleiner als 8 eine kontinuierliche Approximation mit einer im ganzen Interpolationsintervall der Integrationsordnung entsprechenden Genauigkeit sein, weil sonst bereits die um eins verringerte Anzahl der Interpolationsbedingungen kleiner wäre als die diskrete Integrationsordnung.
In dieser Arbeit sollen die Probleme der Konstruktion hinreichend genauer Interpolationen nicht weiter verfolgt werden (siehe z.B. [Bogacki/Shampine]).
Abschließend werde ein dichtes Einschrittverfahren definiert:

Definition 3.2.1 (Dichtes Einschrittverfahren) :
Ein dichtes Einschrittverfahren stellt neben der diskreten Approximation $\tilde{y}_l$ (vergleiche 3.1.1S23) eine kontinuierliche Näherung $p_l(x)$ bereit, für die gilt: $\|y_l(x) - p_l(x)\| = O(h_{l-1}^{p+1})$ gleichmäßig für $x \in [\tilde{x}_{l-1}, \tilde{x}_l]$.
Hierbei steht p für die **dichte Ordnung** des Integrationsverfahrens.
Man weiß dann auch:
"Die **dichte Ordnung** p kann kleiner als die **diskrete Ordnung** q sein".
Die **diskrete Ordnung** q ist bekanntlich über
$\|y_l(x_l) - \tilde{y}_l\| = O(h_{l-1}^{q+1})$ lokal definiert (vergleiche [Schwarz]).

Um die Genauigkeit einer Interpolation mit Daten, die aus Integrationsprozessen gewonnen wurden, zu beurteilen, führt man den Begriff der **Datenordnung** ein.

Definition 3.2.2 (Datenordnung) :
Jeder numerische Integrationsprozeß liefert Ergebnisse, die von der Integrationsschrittweite abhängen (im einfachsten Fall die diskreten Näherungen $\tilde{y}_l$).
Man verallgemeinert nun den nur für $\tilde{y}_l$ definierten Begriff der **diskreten Ordnung** auf alle Ergebnisse $\tilde{z}$, die während des Prozesses anfallen. Dies können sowohl Zwischenwerte für y_l, als auch beliebige Teilausdrücke, die im Verlauf der Rechnung gebildet werden, sein. Man schreibt ihnen die **Datenordnung** r genau dann zu, wenn gilt: $\|z - \tilde{z}\| = O(h_{l-1}^{r+1})$.

Die **Datenordnung** spielt eine wichtige Rolle, wenn man die **dichte Ordnung** eines Einschrittverfahrens beweisen möchte (vergleiche Satz 3.6.2S32).

3.3 Fehlerschranken

Der wichtigste Punkt bei der automatischen seriellen Integration ist die Verwendung hinreichend robuster Fehlerschranken. Wir unterscheiden ausdrücklich zwischen Fehlerschätzern und Fehlerschranken.

Definition 3.3.1 (lokale Fehlerschranke) :

Ein Einschrittverfahren (siehe Def. 3.1.1S23) liefere die diskrete Näherung y_l. Eine **diskrete lokale Fehlerschranke** e_l liegt genau dann vor, wenn der beim Integrationsschritt von x_{l-1} nach x_l zusätzliche, durch die Verwendung eines Einschrittverfahrens bedingte, Verfahrensfehler im Punkt x_l, ohne Berücksichtigung der Rundungsfehler, durch e_l nach oben abgeschätzt werden kann. Die Betrachtung ist lokal, weil man den bisher aufgelaufenen Fehler im Vorgängerwert y_{l-1} nicht berücksichtigt.
Eine **dichte lokale Fehlerschranke** $e_l(x)$ liegt genau dann vor, wenn sich eine geeignete Norm $\|p_l(x)-\hat{y}_l(x)\|$ jeweils für ein $x \in [x_{l-1}, x_l]$ abschätzen läßt. Hierbei steht $\hat{y}_l$ für die Lösung des Anfangswertproblems zum Anfangswert y_{l-1}.

Im Gegensatz dazu liefert ein Fehlerschätzer mehr Information:

Definition 3.3.2 (lokaler Fehlerschätzer) :

Ein **diskreter lokaler Fehlerschätzer** e_l liegt vor, wenn der beim Integrationsschritt von x_{l-1} nach x_l entstehende zusätzliche lokale Diskretisationsfehler (vergleiche [Schwarz]) im Punkt x_l durch e_l geschätzt werden kann.
Ein **dichter lokaler Fehlerschätzer** $e_l(x)$ liegt vor, wenn sich die Differenz $\hat{y}_l(x) - p_l(x)$ jeweils für ein $x \in [x_{l-1}, x_l]$ schätzen läßt; $\hat{y}_l$ wie in 3.3.1S26.
Man wird die **Fehlerschätzer** als **sinnvoll** ansehen, wenn der Fehler des Fehlerschätzers z.B. unter 90% liegt oder von einer höheren Ordnung (i.S. des Integrationsverfahrens) ist.

Oft lassen sich aus Fehlerschätzern Fehlerschranken gewinnen.

Lemma 3.3.3 (Schranken, die auf Schätzern beruhen) :

Vor.: Sei ein **sinnvoller Fehlerschätzer** $e_l(x)$ nach Definition 3.3.2S26 gegeben:
$\|(e_l(x) + p_l(x)) - \hat{y}_l(x)\| < 0.9 * \|p_l(x) - \hat{y}_l(x)\|$ bzw.
$\|e_l(x) - (\hat{y}_l(x) - p_l(x))\| < 0.9 * \|\hat{y}_l(x) - p_l(x)\|$.

Beh.: $10 * \|e_l(x)\|$ ist eine **Fehlerschranke**; siehe 3.3.1S26.

Bew.: Aus der Voraussetzung folgt: $\|p_l(x) - \hat{y}_l(x)\| < 10 * \|e_l(x)\|$.
Man beachte, daß diese Schranke auch für die verbesserte Näherung $e_l(x) + p_l(x)$ gilt ! $\diamondsuit$

Leider können **Fehlerschätzer** und Fehlerschranken auch versagen; dies bedeutet, daß sie nicht mehr **sinnvoll** sind bzw. keine Schranken mehr sind. Ein Ziel dieser Arbeit ist die Gewinnung einer möglichst robusten/verläßlichen **Fehlerschranke**, die für "möglichst viele" Probleme eine Schranke ist.

Beispiel 3.3.4 (Versagen eines Fehlerschätzers) :

Betrachte i.S. von **Problem D** (siehe 2.2.10S6):

$$y' = \left\{ \begin{array}{rl} 0 \text{ für } x - 0.5 + \delta < & 0 \\ B \text{ für } x - 0.5 < & 0 \\ -B \text{ für } x - 0.5 - \delta < & 0 \\ 0 \text{ für } x - 1 < & 0 \end{array} \right\} \text{ und } y(0) = y_0.$$

Hierbei sind $B, y_0, \delta \in \mathbb{R}^+$ zunächst beliebig, $\delta \ll 1$.

Nun nehmen wir uns ein eingebettetes Runge-Kutta-Verfahren der Ordnungen $q(p)$ mit $q > p$ (siehe Literatur) daher und betrachten den dort üblicherweise benutzten Fehlerschätzer:

Er stellt eine Linearkombination $h * \sum_{i=1}^{s} f_i * k_i$ dar.

Die k_i sind Steigungen an gewissen Stellen x_i.

Nun liegt die rechte Seite in Black-Box-Form vor:

(Wir erhalten als Output nur die Steigung !)

$$y' = \left\{ \begin{array}{rrcl} 0 \text{ für } & 0 \le x & < & 0.5 - \delta \\ B \text{ für } & 0.5 - \delta \le x & < & 0.5 \\ -B \text{ für } & 0.5 \le x & < & 0.5 + \delta \\ 0 \text{ für } & 0.5 + \delta \le x & < & 1 \end{array} \right\}$$

Angenommen, die Schrittweite h sei so gewählt, daß nur x_s im Intervall $[0.5 - \delta, 0.5]$ liegt und alle anderen x_i kleiner als $0.5 - \delta$ sind.

Dann liefert der Fehlerschätzer: $e = h * f_s * k_s$.

Die Runge-Kutta-Näherung ist von der Form:

$$\tilde{y} = y_0 + h * \sum_{i=1}^{s} \tilde{b}_i * k_i \;=\; y_0 + h * \tilde{b}_s * k_s.$$

1.) Nehmen wir an, y_0 sei Null.

Wenn wir ein relatives Fehlerkriterium verwenden, gilt: $e_{rel} = \dfrac{e}{\tilde{y}} = \dfrac{f_s}{\tilde{b}_s}$,

unabhängig von h, B, δ ! Man beachte, daß oft $|f_i|$ klein gegen $|\tilde{b}_i|$ ist.

Falls f_s verschwindet, würde man die Näherung sicher aktzeptieren und sich einen unbekannten Fehler einhandeln.

Ganz wichtig:

Ein ähnliches Beispiel läßt sich auch mit boden Funktionen konstruieren !

2.) Nehmen wir nun an, y_0 sei $\gg hB$.

Dann liefert e_{rel} einen kleinen Wert, der i.w. von hB/y_0 abhängt:

$$e_{rel} = \frac{f_s}{\tilde{b}_s + \dfrac{y_0}{hB}}$$

Angenommen, der Schritt wird aktzeptiert (vergleiche Abschnitt "Schrittweitensteuerung").

Unsere Lösung wird aber mit einem größeren Fehler berechnet, je nach Wahl von hB/y_0 und dem Abstand des Endpunkts $x_0 + h$ von $0.5 - \delta$!

Ein "gutes" Fehlermaß müßte die Form $\dfrac{error}{y_0 + B(h + \delta - 0.5)}$ besitzen.

Der betrachtete Fehlerschätzer besitzt allerdings auch einen ganz wesentlichen Vorteil: Er ist enorm effizient.

Aus diesem "kleinen" Beispiel lernen wir also bzgl. der **automatischen Erkennung von Unstetigkeiten** viererlei:

1. Es kommt auf die Wahl der Zwischenpunkte des Integrators an, insbesondere sollte
2. die Ableitung am Endpunkt des Integrationsschritts in die Berechnung des **Fehlerschätzers** eingehen.
3. Der **Fehlerschätzer** sollte in diesem Beispiel nicht nur von hB/y_0 abhängen.
4. Man sollte zuerst einen Schritt bis kurz vor die **Unstetigkeit** machen und sich dann sehr genau überlegen, wie groß der Fehler beim Überschreiten der **Unstetigkeit** ist.

Bisher wurde nicht festgelegt, welche Normen für **lokale Fehlerschranken** geeignet sind. Aus regelungstheoretischen Gründen sollte man "glatte" Normen verwenden (siehe Abschnitt "Schrittweitenregelung"). Beispielsweise bietet sich eine skalierte euklidische Norm an.

Definition 3.3.5 (Relatives/Absolutes Fehlermaß) :
Wir geben hier eine Skalierung der euklidischen Norm an, die in gewisser Weise einen sehr glatten Übergang vom relativen zum absoluten Fehlerkriterium bietet:
Seien e_i die Komponenten eines sinnvollen Fehlerschätzers und z_i die Komponenten der verbesserten Näherung.
Definiere $e_{rel/abs} := 10 * \sqrt{\sum_{i=1}^{i_{max}} \frac{e_i^2 * (\ 1 + \exp(\frac{1}{2} * (1 - (\frac{z_i}{\tau})^2))\ )}{z_i^2 + \tau^2}}$ mit
10 als Sicherheitsfaktor und τ als geforderter, oberer Fehlergrenze.

In der Praxis steht man vor folgendem Dilemma bei der Wahl einer Skalierung:
Einerseits gilt die Ungleichungskette ($\hat{y}$ exakter Vektor)
$\forall\ i : \frac{|z_i - \hat{y}_i|}{|\hat{y}_i|} \leq 10\frac{|e_i|}{|\hat{y}_i|} \leq 10\|(\frac{e_i}{\hat{y}_i})_{1\leq i\leq imax}\|_\infty \leq 10\|(\frac{e_i}{\hat{y}_i})_i\|_2.$
Man kann also theoretisch eine skalierte euklidische Norm angeben, die den maximalen relativen komponentenweisen Fehler übertrifft, aber andererseits kennt man den exakten Vektor nicht und muß sich daher mit der Form $10\|(\frac{e_i}{z_i})_i\|_2$ begnügen, die keine Schranke ist. Erschwerend kommt hinzu, daß das relative Fehlerkriterium am Nullpunkt nicht definiert ist, weshalb wir das obige "Übergangsfehlermaß" (Maß nicht im Sinne der Maßtheorie) formulierten, das ebenfalls <u>keine</u> **lokale Fehlerschranke** darstellt.

Entweder verwendet man konstante Skalierungsfaktoren und kann dann nicht von einem relativen Kritierium sprechen oder man verzichtet auf die wesentliche Eigenschaft einer Fehlerschranke.

Problem 3.3.6 (Fehlerdilemma) :
Finde eine Skalierung, die das beschriebene Dilemma zumindest "verringert".

Die Verwendung des absoluten Fehlerkriteriums $10\|(e_i)_i\|_2$ ist in der Praxis ebenso wenig problemgerecht wie der Gebrauch von $10\frac{\|(e_i)_i\|_2}{\|(z_i)_i\|_2}$. Betrachte hierzu den exakten Vektor $\begin{pmatrix} 0.1 \\ 100 \end{pmatrix}$ und den Fehlervektor $\begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}$.

3.4 Schrittweitenregelung

Die Grundlagen der Schrittweitensteuerung werden im Buch von [HWN] ausführlich beschrieben. Der Grund dafür, daß wir die Schrittweitensteuerung als Regelung bezeichnen, liegt darin, daß neuere Ansätze (siehe z.B. [Gustafsson]) erfolgreich die Regelungstheorie auf die Wahl der Integrationsschrittweite anwenden. Das Ziel der Regelung besteht in der Gewinnung von Schrittweiten, für die ein Fehlerschätzer oder eine Fehlerschranke dem Toleranzwert nahekommen.
Der in Abschnitt "Einschrittverfahren" definierte Algorithmus legt die Schrittweiten h_l nicht fest. Es ist nun gerade das Kennzeichen einer effizienten Integration, daß angepaßte Schrittweiten verwendet werden. Die Anpassung der Schrittweite erfolgt durch Auswertung einer **lokalen Fehlerschranke** (siehe 3.3.1S26) und einer daraus resultierenden Vorgabe für die zu verwendende Schritweite. Der mittlerweile übliche Regelungsalgorithmus sieht in seiner einfachsten Form so aus (Herleitung siehe [Werner/Arndt]):

Algorithmus 3.4.1 (Schrittweitenregelung) :

$h_{new} \leftarrow h_{start}$
$x \leftarrow a$
$\tilde{y} \leftarrow y_0$
while($x < b$) {
  do { // at least once
    $h \leftarrow h_{new}$
    perform integration step with h, result: $\tilde{y}_{new}$
    calculate smooth error estimator bound e
    offer new stepsize $h_{new} \leftarrow h / \sqrt[r]{\frac{e}{\tau}}$
  } while($e \geq \tau$)
  $x \leftarrow x + h$ //step is accepted
  Output: $\tilde{y} \leftarrow \tilde{y}_{new}$
} //end while

Algoritmus 3.4.1S29 ist in der vorliegenden Form nicht wohldefiniert:

1. Es fehlt die Berücksichtigung des Falls, daß die Fehlerschranke e Null liefert.
   Dies ist auf vielfältige Art und Weise einfach möglich (siehe z.B. Seite 30).

2. Die Schleifen müssen nicht terminieren.
   Beispielsweise ließe sich die Zahl der Schleifendurchläufe begrenzen oder eine minimale Schrittweite festlegen.

3. Die Parameter h_{start}, r und τ sind geeignet festzulegen.
   Hierbei wählt man üblicherweise r größer oder gleich der Fehlerordnung des Integrationsverfahrens. τ bezeichnet die **lokale Fehlertoleranz**, d.i. die geduldete Größe des lokalen Fehlers. Eine sinnvolle Wahl für τ ist problemabhängig und bleibt daher dem Anwender überlassen, der außerdem die zur Verfügung stehende Rechengenauigkeit berücksichtigen muß.

Obiger Algorithmus läßt sich im Prinzip auch dann verwenden, wenn **Unstetigkeiten** im Verlauf des Integrationsprozesses auftreten.

Man muß allerdings darauf achten, daß die **lokale Fehlerschranke** auch in diesem Fall robust ist (vergleiche Abschnitt "Fehlerschranken", insbesondere Beispiel 3.3.4S27).

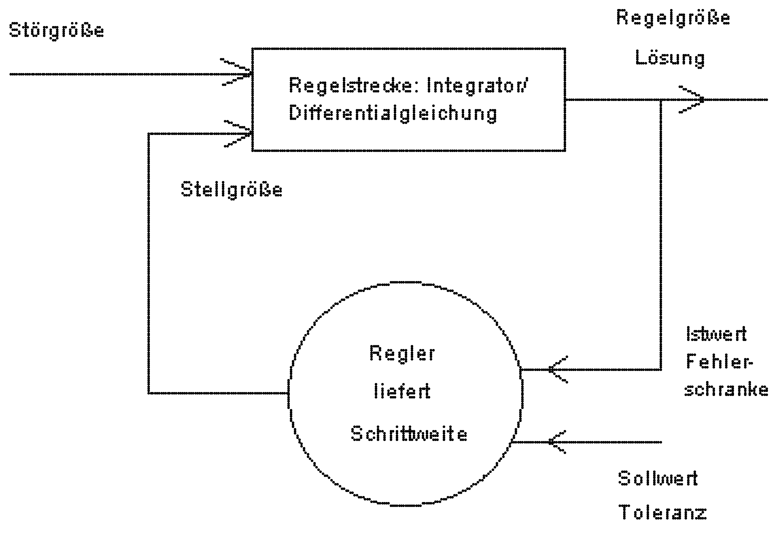
In der Praxis wird die neue Schrittweite meist mittels

$$h_{new} \leftarrow \max(h_{\min}, \min(h_{\max},\ h\ /\ \min(\max(\sqrt[r]{\frac{\max(\rho e, \mu)}{\tau}}, g), k)))$$

berechnet, wobei alle Spezialfälle inbegriffen sind. Wir haben die Begrenzungsfaktoren $g = 0.01$, $k = 2.0$, den Sicherheitsfaktor $\rho = 1.1$ und den Exponenten $r = 8$ gewählt.
μ ist die Maschinengenauigkeit und τ die **Fehlertoleranz**. Die minimale Schrittweite wird auf $20\mu \cdot (|x| + 1)$ gesetzt und $h_{\max}$ ist problemabhängig.
Algorithmus 3.4.1S29 wird außerdem um eine besondere Berücksichtigung des Endpunkts b erweitert, damit man die Lösung genau am Endpunkt auch tatsächlich erhält.
An dieser Stelle wird kurz dargelegt, warum eine glatte Fehlerschranke verwendet wird.

> Zu lösen sei ein **Basisproblem** (siehe 2.2.5S5).
> Zugrundegelegt werde ein einfacher Regelkreis nach Abbildung 3.4.2S30.
> Man erkennt, daß nur eine Steuergröße aus allen Komponenten der Lösung gebildet wird.
> Nun läßt sich aber der Regelkreis selbst wieder als Differentialgleichung auffassen (siehe [Foellinger]).
> Ergo sollte die Regelgröße und mithin die daraus berechnete Fehlerschranke glatt sein, damit die Differentialgleichung des Regelkreises im klassischen Sinn aufgefaßt werden kann und nicht Anlaß zu Überlegungen analog zu **Problem D** o.ä. gibt.

Abbildung 3.4.2 (Regelkreis) :



3.5 Parallel-iterierte Runge-Kutta-Verfahren

Bei den **parallel-iterierten Runge-Kutta-Verfahren** handelt es sich um die mit dem Standardansatz der **Vektoriteration** parallelisierte Variante der Runge-Kutta-Verfahren, die sowohl explizit, als auch implizit sein können (siehe [HouSom]). Wir betrachten hier die einfachste Vaiante:

Definition 3.5.1 (PIRKG(s,m)) :

Sei mit $A \in \mathbb{R}^{s\times s},\ b, c \in \mathbb{R}^s$ das s-stufige, implizite Runge-Kutta-Verfahren vom Gauß-Typ der diskreten Ordnung $2s$ gegeben,

dargestellt in der Notation von Butcher: $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$

Es ergibt sich ein i.a. nichtlineares System für K :

$$y_{l+1} = y_l + hWK \text{ mit } K = \begin{pmatrix} K_1 \\ : \\ K_s \end{pmatrix} \text{ und}$$

$$K_\sigma = f(x_l + c_\sigma h, y_l + h\sum_{\nu=1}^{s} A_{\sigma\nu}K_\nu),\ W = (b_1 I_{n\times n}, \ldots, b_s I_{n\times n}).$$

K ist ein **Super-Vektor**, der selbst aus Vektoren besteht.

Nun iteriert man dieses Gleichungssystem der Stufen m mal zum Startwert 0 und erhält ein <u>explizites</u> Runge-Kutta-Verfahren mit ms Stufen der Ordnung $\min(2s, m)$:

$$\begin{array}{c|ccccccc} c & O & O & O & \ldots & O & O & O \\ c & A & O & O & \ldots & O & O & O \\ c & O & A & O & \ldots & O & O & O \\ & \ldots \\ & \ldots \\ & \ldots \\ c & O & O & O & \ldots & O & A & O \\ \hline & 0_s^T & 0_s^T & 0_s^T & \ldots & 0_s^T & 0_s^T & b^T \end{array}$$

Hierbei steht O für die $s \times s$ Nullmatrix. Dieses Verfahren ist ohne Konvergenzbetrachtungen (bzgl. der Vektoriteration) wohldefiniert, besitzt allerdings <u>nicht</u> die Eigenschaften des impliziten Gaußverfahrens (vergleiche [HouSom]).

Insbesondere ist das Stabilitätsgebiet beschränkt.

A, b und c sind bis $s = 5$ in [Butcher(2)] gegeben.

Den Nachweis für die diskrete Konvergenzordnung $\min(2s, m)$ (bzgl. der Integration) führen [JaKvaeNo] aus; u.a. werden dort auch andere Iterationsverfahren betrachtet.

Im praktischen Teil dieser Arbeit verwenden wir das **PIRKG** mit 4 Stufen und 8 Iterationen; es besitzt also die Ordnung 8. Die Einzelheiten der parallelen Variante werden in Abschnitt "Parallelitätsebenen" besprochen.

3.6 Dichte, parallel-iterierte Runge-Kutta-Verfahren

Kombiniert man das **PIRKG** mit einem Interpolationspolynom, so erhält man ein **dichtes Einschrittverfahren** (siehe 3.2.1S25).

Definition 3.6.1 (DPIRKG(s,m,t)) :

Vorgelegt sei ein **PIRKG**(s, m) nach Def. 3.5.1S31.
Verwendet man die Zwischenstufen als Näherungen für Ableitungen an den Zwischenpunkten $x + c_i h$ $(1 \leq i \leq s)$, so besitzt man mit den berechneten Argumenten für die rechte Seite insgesamt $2(s+1)$ Daten, die allerdings z.T. eine geringere Ordnung (siehe 3.2.2S25) aufweisen.
Wir verwenden den Anfangs- und Endpunkt, sowie die t $(1 \leq t \leq s-2)$ Punkte, die durch c_2 bis c_{t+1} dargestellt werden, um ein Hermitepolynom des Grades $2t+3$ zu konstruieren. Dies liefert eine kontinuierliche Approximation im Integrationsinterval $[x, x+h]$.

Zur Genauigkeit des konstruierten Verfahrens betrachte

Satz 3.6.2 (Dichte Ordnung des DPIRKG) :

Vor.: Sei ein **DPIRKG**(s,m,t) nach Def. 3.6.1S32 gegeben.
Beh.: Es besitzt die **dichte Ordnung** (siehe 3.2.1S25) $\min(s, m, 2t+3)$.
Bew.: Skizze:

stage-order $\longrightarrow$ Datenordnung $\longrightarrow$ dichte Ordnung.

1.) Anwendung von Theorem 7.9 aus [HWN]:
ergibt die stage-order für Runge-Kutta-Gauß-Verfahren
2.) Anwendung von Theorem 1 aus [JaKvaeNo]:
Iterationszahl $\wedge$ Startwertordnung $\Longrightarrow$
Ordnung der iterierten Stufen
3.) einfache Dreiecksungleichung:
Ordnung der iterierten Stufen $\wedge$ stage-order $\Longrightarrow$ Datenordnung
(Datenordnung siehe Definition 3.2.2S25)
4.) Verwendung der Herleitung aus [Dopri(4)]:
Datenordnung $\wedge$ Datenanzahl $\wedge$ ...(*) $\Longrightarrow$ dichte Ordnung.

Die zusätzlichen technischen Vor. (*) sind für explizite Runge-Kutta-Verfahren erfüllt.$_{\text{qed}}$

Man beachte, daß man im Beweis von Satz 3.6.2S32 auch auf die Eigenschaften des ursprünglich zugrundeliegenden impliziten Runge-Kutta-Verfahrens zurückgreift. Deshalb ist die Ausnutzung der kontinuierlichen Approximation des **DPIRKGs** nur sinnvoll, wenn die Fixpunt-Jacobi-Iteration konvergiert. Dies steht im Gegensatz zu der Aussage in Definition 3.5.1S31 und impliziert eine begrenzte Anwendbarkeit des **DPIRKGs**. Glücklicherweise ist die Begrenzung bei mittleren bis hohen Genauigkeitsanforderungen nicht so tiefgreifend, genauer:
Die Integrationsschrittweite h wird eingeschränkt:

$$h < \frac{1}{NL} \text{ mit } N := \max_{\sigma} \sum_{\nu} |A_{\sigma\nu}| \text{ und } L := \max_{x \in [x_{l-1}, x_l]} \max_{\sigma} \sum_{\nu} |A(x)_{\sigma\nu}|,$$

wobei A die Koeffizientenmatrix des **PIRKGs** und $A(x)$ die Koeffizientenmatrix des **Basisproblem** ist (siehe [Schwarz]); L ist die lokal gültige Lipschitzkonstante. Für das implizite Verfahren vom Gauß-Typ gilt: $N < 1$.
Bei den **Testbeispielen** muß diese Einschränkung der Schrittweite nicht explizit berücksichtigt werden; die von uns verwendete **Fehlerschranke** reagiert automatisch auf eine Verletzung der obigen Voraussetzung.
Beispielsweise besitzt das **DPIRKG**(4,8,2) die Ordnung 8 und die **dichte Ordnung** 4. Wünschenswert wäre die Konstruktion eines Verfahrens mit der **dichten Ordnung** 8. Da wir jedoch nur die einfache **Schrittweitenregelung** nach 3.4.1S29 betrachten, genügt uns die erzielte Genauigkeit; eine genauere Begründung liefert der nächste Abschnitt.

3.7 Lösen der Defektdifferentialgleichung

Zur Gewinnung eines robusten/verläßlichen **Fehlerschätzers** kann man die Defektdifferentialgleichung heranziehen.

Definition 3.7.1 (Defektdifferentialgleichung) :

Seien die autonome Differentialgleichung (DE) $y' = f(y)$ und ein dichtes Einschrittverfahren nach Def. 3.2.1S25 gegeben.
Die Defektdifferentialgleichung stellt sich mit der kontinuierlichen Approximation $p(x)$ so dar: (SC) $e'(x) = f(p(x) + e(x)) - p'(x)$.
(DE) und (SC) sind im Sinne von **Problem EU** (siehe 2.5.28S22) aufzufassen, falls f linear und unstetig ist.
Eine sinnvolle Approximation $p(x)$ ist automatisch differenzierbar.
[Skeel] bezeichnet dieses Vorgehen als "solving for the correction".
Allgemein bekannt ist der Ausdruck "defect correction".

Wir verwenden die **Lösung der Defektdifferentialgleichung** lokal, d.h. nur im jeweiligen Integrationsteilinterval $[x, x+h]$. Dadurch erhalten wir die folgende **lokale Fehlerschranke**:

Definition 3.7.2 (Sftc-error) :

Sei die Defektdifferentialgleichung (SC) nach 3.7.1S34 im Teilinterval $[x_{l-1}, x_l]$ gegeben.
Löst man diese zum Anfangswert 0 mitttels eines beliebigen Einschrittverfahrens, so erhält man eine diskrete Näherung $e_{l,\tilde{l}}$, passend zur lokalen, kontinuierlichen Approximation p_l.
Der zweite Index bezeichnet die Teilergebnisse, wenn mehrere Integrationsschritte im Teilinterval $[x_{l-1}, x_l]$ durchgeführt werden.

Als **lokale Fehlerschranke** kann man $e_l := 10 * \sqrt[2]{\sum_{\tilde{l}=1}^{\tilde{l}\max} e_{l,\tilde{l}}^2}$ benutzen.

Man beachte, daß keine **dichte Fehlerschranke** (siehe 3.3.1S26) vorliegt.
Der **spezielle Sftc-error** liegt genau dann vor, wenn zwei Schritte mit einem **PIRKG**(s, m) (siehe 3.5.1S31) zur Integration von (SC) ausgeführt werden. Der Zusatz "speziell" wird je nach Zusammenhang auch weggelassen.

Zu zeigen bleiben die Punkte

1. Unter welchen Voraussetzungen ist der **Sftc-error** tatsächlich eine wohldefinierte **lokale Fehlerschranke** ?

2. Der **Sftc-error** stellt im Sinne des Beispiels 3.3.4S27 eine "Verbesserung" gegenüber "einfacheren" Fehlerschranken dar.

Ad 1) beachte "Sätzchen"

Lemma 3.7.3 (Sftc-error ist wohldefiniert) :

Vor.: wie in Definition 3.7.2S34.
Zusätzlich gelte: $\|e_{l,\tilde{l}\max} - \hat{y}_l(x_l) + p_l(x_l)\| < 0.9\|\hat{y}_l(x_l) - p_l(x_l)\|$;
hierbei steht $\hat{y}_l$ für die Lösung eines Basisproblems zum Anfangswert y_{l-1}.

Beh.: $\|\hat{y}_l(x_l) - p_l(x_l)\| < e_l$.

Bew.: analog zu Lemma 3.3.3S26. $\diamondsuit$

Die zusätzliche Bedingung in Lemma 3.7.3S35 quantifiziert die Genauigkeit, mit der die Defektdifferentialgleichung "global im Lokalen" zu lösen ist. Leider kann man sie selten nachprüfen. Insofern können wir nicht von einer exakten Fehlerschranke sprechen, aber wir könnten eine "approximative Fehlerschranke" einführen, die mit hoher Wahrscheinlichkeit eine **lokale Fehlerschranke** ist.

Stattdessen untersuchen wir lieber den Punkt 2) auf Seite 34 genauer. Geht man zunächst von der in praxi unmöglichen exakten Lösung der **Defektdifferentialgleichung** aus, so ergibt Beispiel 3.3.4S27 im Interval $[0, h]$ mit $h \in (\omega, 0.5]$ mit $\omega := 0.5 - \delta$:

$$\begin{aligned} \hat{e}_1' &= 0 - p' = -p' \text{ für } x \in [0\,,\,\omega] \text{ mit } \hat{e}_1(0) = 0 \text{ und} \\ \hat{e}_2' &= B - p' \qquad\quad \text{für } x \in [\omega,\, h] \text{ mit } \hat{e}_2(\omega) = \hat{e}_1(\omega). \end{aligned}$$

Ergo ist $e_1(x) = p(0) - p(x) = y_0 - p(x) = -p(x)$ für $y_0 = 0$.
e_2 ist ein Polynom, dessen Grad gleich dem von p ist, falls Grad p größer 0 gilt:
$e_2(x) = C + Bx - p(x)$ mit $C := y_0 - B\omega$.
(Hieraus liest man auch die exakten Lösungen der **Basisprobleme** ab.)
Das praxisrelevante, relative Fehlerkriterium liefert am Endpunkt:
$e_{rel} = |\frac{e_2(h)}{e_2(h) + p(h)}| = |1 - \frac{p(h)}{C + Bh}|$.

4 Parallelität

4.1 Ebenen der Parallelität

Algorithmen sind oft hierarchisch aufgebaut, die Hierarchie läßt sich an den verschachtelten Blockstrukturen ablesen. Darüberhinaus können weitere Parallelitätsebenen innerhalb der elementaren (rechnerorientierten) Operationen unterschieden werden (z.B. bei Vektorrechnern). Der Gesamtalgorithmus zur Behandlung eines Problems vom Typ EU gliedert sich in die nachstehend aufgeführten Teile:

Algorithmus 4.1.1 (Behandlung von Problem EU) :

Äußere Integrationsschleife
- Schleife zur Bestimmung einer geeigneten Schrittweite
  - Ausführung eines Integrationsschritts
  - Aufbau der kontinuierlichen Lösung
  - Fehlerschranken und Schrittweitenvorhersage
    - Innere Integrationsschleife für die Defektdifferentialgleichung
      - Einzelschritte der inneren Integration

Die einzelnen Integrationsschritte, sowohl in der äußeren, als auch in der inneren Schleife, bestehen ihrerseits aus Teilalgorithmen (siehe 3.5.1S31). Hervorzuheben ist hierbei insbesondere die Auswertung der rechten Seite.

Dieser Gesamtalgorithmus ist bis zur Ebene der Einzelintegrationschritte seriell. Nacheinander wird das Integrationsintervall durchschritten. Eine geeignete Schrittweite wird in aufeinanderfolgenden Versuchen bestimmt, wobei die Zahl der Versuche aufgrund des Erfolgskriteriums "Fehler kleiner Toleranz" schwankt. Da die Schrittweite variiert, läßt sich auch die Zahl der erfolgreichen äußeren Integrationsschritte nicht a priori festlegen. In der inneren Integrationsschleife hingegen werden immer zwei aufeinanderfolgende Schritte (vergleiche 3.7.2S34) ausgeführt, entsprechend der Wahl des **speziellen Sftc-error**.

Jeder Integrationsschritt besteht aus seriellen und parallelen Elementen:

Algorithmus 4.1.2 (PIRKG parallel) :

Serielle Iterationsschleife mit m Schritten
- Parallele Auswertung der rechten Seite entsprechend den ursprünglich seriellen Runge-Kutta-Stufen (Anzahl: s) des impliziten Gaußverfahrens(siehe 3.5.1S31)
  - Parallelität innerhalb der rechten Seite

Die **Testprobleme** aus dem gleichnamigen Abschnitt sind nicht geeignet, um die untere Ebene der Parallelität auszuschöpfen, hierzu müßten hochdimensionale Gleichungen konstruiert werden.

Desweiteren läßt sich der <u>Aufbau der kontinuierlichen Lösung</u> auf der gleichen Ebene parallelisieren wie die Auswertung der rechten Seite. Die einzelnen Polynome einer Vektorlösung lassen sich parallel bzgl. der Komponenten aufbauen und auswerten.

Da in dieser Arbeit die Eigenarten des **Problems EU** im Vordergrund stehen, wird die eigentlich interessante Parallelisierung bzgl. der rechten Seite nicht weiter analysiert. Insofern ist der beschriebene Ansatz zunächst nicht massiv-parallel, weil zur parallelen Berechnung der Runge-Kutta-Stufen nur s Prozessoren nötig sind. Da man üblicherweise bei Verwendung einer doppeltgenauen Arithmetik (Rechengenauigkeit ca. 10^{-15}) nur Ordnungen bis maximal 20 betrachtet, benötigt man also maximal 10 Prozessoren (vergleiche 4.1.2S36). Die Stufen lassen sich parallel berechnen, weil die Iterationsschleife gerade die Jacobiiteration widerspiegelt, die keine Verknüpfungen zwischen einzelnen **Super-Vektorkomponenten** (die Komponenten eines **Super-Vektors** sind Näherungen der mehrdimensionalen Stufen im Falle einer unendlichen Iteration) benötigt. Man erkennt die Parallelität sehr einfach am Aufbau des Butchertableaus:
Die Koeffizientenmatrix ist selbst eine subdiagonale Blockmatrix. Die Blöcke werden in sich parallel, nacheinander abgearbeitet.
Wie bereits angedeutet, wird man also durch Blockbildung innerhalb der Runge-Kutta-Koeffizientenmatrix nur geringe Parallelisierungserfolge erzielen (vergleiche auch mit [JaNo]). Dennoch lohnt es sich, die "Effizienzuntersuchungen" im nächsten Abschnitt genauer zu betrachten.

Wünschenswert wären eine Parallelisierung der Iterationsschleife ($m > s$!) sowie die Entwicklung paralleler Fehlerschranken und Schrittweitenbestimmungsverfahren. Die Idee der eingebetteten Runge-Kutta-Verfahren oder die parallele Berechnung eines Verfahrens mit einer niedrigeren Ordnung erlaubt zwar eine größere Effizienz, diese wird aber auf Kosten der Verläßlichkeit erkauft (angedeutet im Abschnitt "Fehlerschranken").

Abgesehen von der parallelen Organisation der Berechnungen muß man auch den Speicherzugriff auf die Datenstrukturen parallelisieren. Stellvertretend wird hier der Lesevorgang beschrieben, der der Berechnung der Argumente für die rechte Seite vorausgeht. Ein typisches Argument besitzt die Gestalt einer Runge-Kutta-Summe

$$y_l + h \sum_{\nu=1}^{s} A_{\sigma\nu} K_\nu.$$

s Prozessoren können wie folgt auf die s Stufen K_ν zugreifen:
Im ersten Schritt liest der erste Prozessor die erste Stufe, der zweite die zweite, ... und der s. Prozessor liest die Stufe K_s. Im zweiten Schritt liest der erste Prozessor die zweite Stufe, der zweite die dritte, ... und der s. Prozessor die erste Stufe. Dies setzt sich analog bis zum letzten Schritt fort; in dem der erste Prozessor die Stufe K_s, der zweite die erste Stufe K_1, ... und der s. Prozessor die Stufe K_{s-1} liest.
Die kann man sich noch anhand der folgenden Matrix klarmachen, bei der auf je eine Spalte pro Schritt zugegriffen wird.

$$\begin{array}{lllll}
K_1 & K_2 & \dots & K_{s-1} & K_s \\
K_2 & K_3 & \dots & K_s & K_1 \\
. & . & \dots & . & . \\
. & . & \dots & . & . \\
. & . & \dots & . & . \\
K_s & K_1 & \dots & K_{s-2} & K_{s-1}
\end{array}$$

Die Elemente einer Spalte lassen sich jeweils mit den lokal auf jedem Prozessor bereitstehenden Koeffizienten $A_{\sigma\nu}$ multiplizieren und dann aufsummieren.

4.2 Effizienzuntersuchungen

Die Wahl eines geeigneten Effizienzmaßes ist durchaus umstritten (vergleiche [DonGen]). In jedem Fall müßte man sich nun eigentlich in die langweiligen Niederungen des Auszählens von Rechen- und Kommunikationsoperationen begeben. Stattdessen schließen wir uns aus reiner Faulheit der Gepflogenheit an, davon auszugehen, daß der Aufwand i.w. durch die Komplexität der rechten Seite der Differentialgleichung gegeben ist. Wir vernachlässigen hierbei sowohl den overhead des Integrations- und Schrittweitenauswahlverfahrens als auch die Kommunikation. Letzteres ist gerechtfertigt, wenn der Rechner passend zum Algorithmus gewählt wird und die Rechenoperationen überwiegen. Ersteres wird experimentell abgesichert:
Hierzu verwendet man Testbeispiel 2.3.3S12 mit unterschiedlichen Zählschleifenparametern (siehe "serielle bzw. parallele Ergebnisse ").
Außerdem geht man natürlich davon aus, daß jede f-Auswertung i.w. die gleiche Zeit benötigt.

Die Anwendung von Algorithmus 4.1.2S36 stellt unter der angegebenen vereinfachenden Annahme den wesentlichen algorithmischen Teil dar. Es werden s Prozessoren benötigt, um die Runge-Kutta-Stufen zu berechnen. Alles übrige läßt sich von einem Prozessor erledigen, zwangsläufig verbunden mit einem Auslastungsungleichgewicht (load imbalance). Theoretisch könnte man dieses auf Kosten der Kommunikation verringern, praktisch würde sich dies nicht effizienzsteigernd auswirken.
Wenn wir den zeitlichen Aufwand in f-Auswertungen messen, ergeben sich m Zeitschritte (für die Iteration bis zur gewünschten Ordnung) plus zweimal $\tilde{m}$ Zeitschritte (für die Integration der Defektdifferentialgleichung). Wir haben $m = 8$ und $\tilde{m} = 4$ gewählt, benötigen also 16 Zeitschritte insgesamt.

Satz 4.2.1 (Komplexität) :

Vor.: Sei ein **DPIRKG**$(\frac{p}{2}, p, 2)$ (siehe Def. 3.6.1S32) für gerade Ordnung p gegeben.
Ferner werde der **Sftc-error** (siehe Definition 3.7.2S34) als **Fehlerschranke** verwendet.
Zur Integration der **Defektdifferentialgleichung** werden 2 Integrationsschritte mit einem **PIRKG**$(\frac{p}{2}, \frac{p}{2})$ ausgeführt.

Beh.: Algorithmus 4.1.1S36 benötigt i.w. $\frac{p}{2}$ Prozessoren und $2p$ parallele bzw. p^2 serielle Zeitschritte.
Dies ergibt ein Speedup von $\frac{p}{2}$ und eine Effizienz(siehe [Schendel(2)]) von 1 !

Bew.: klar.$_{\text{qed}}$

In anderen Worten besagt Satz 4.2.1S38, daß die beschriebene parallele Variante von Algorithmus 4.1.1S36 zumindest theoretisch für jede vorgegebene gerade Ordnung p auf $\frac{p}{2}$ Prozessoren tatsächlich $\frac{p}{2}$-mal so schnell abläuft wie die serielle Variante. Leider ist die praktisch verwendbare Ordnung nach oben hin begrenzt (siehe Seite 37).

Es ist klar, daß die oben beschriebene load imbalance in der Praxis zu drastischen Effizienzeinbußen führen kann (beachte Amdahls Gesetz), insbesondere dann, wenn die Kommunikation zu langsam erfolgt oder der Aufwand für die rechte Seite pro Komponente zu klein ist und mithin obige Annahme nicht mehr gerechtfertigt ist. Im letzeren Fall müßte man Verfahren mit möglichst geringem overhead pro Komponente verwenden.

Um eine möglichst rasche Kommunikation zu gewährleisten, sollte Algorithmus 4.1.1S36 auf einem shared-memory System bestehend aus unabhängigen Vektorprozessoren ausgeführt werden. Hierbei wird vorausgesetzt, daß sich die rechte Seite vektorisieren läßt; dies ist bei den **Basisproblemen** der Fall.

Wir betonen noch einmal, daß das betrachtete Effizienzmaß ausschließlich den wesentlichen Parallelisierungsgrad des parallelen **DPIRKG** im Vergleich zum seriellen **DPIRKG** beschreibt. Man könnte das Ergebnis von Satz 4.2.1S38 auch so formulieren:
Es ist nicht möglich, Algorithmus 4.1.1S36 so seriell zu organisieren, daß sich redundante Rechenoperationen einsparen lasssen.

Die theoretisch sehr guten Ergebnisse der Effizienzuntersuchungen werden durch einen Vergleich mit einem anderen denkbaren Verfahren relativiert. Der Einfachheit halber soll sich dieses Verfahren nur im verwendeten Integrationsverfahren unterscheiden.

Verwendet man zur seriellen Integration der **Basisprobleme** ein Extrapolationsverfahren vom Gragg-Bulirsch-Stoer Typ (vergleiche mit "ODEX" aus [HWN]), so werden bei Wahl der aufwandsoptimalen Schrittweitenfolge $\{1, 2, 3, \ldots\}$ ca. $\frac{1}{2}\frac{p}{2}(\frac{p}{2}+1)$ f- Auswertungen (= Zeitschritte) benötigt, um die konstant vorgegebene Ordnung p zu erreichen. Wenn wir den Aufwand zur Lösung der **Defektdifferentialgleichung** ebenso hoch ansetzen, dann ergeben sich ca. $\frac{p^2}{4}$ Zeitschritte insgesamt. In diesem Fall würde ein Vergleich des seriellen Verfahrens mit unserem parallelen Verfahren nur eine Effizienz von $\frac{1}{4}$ ergeben !
Man beachte, daß es sich hierbei um ein Kriterium zur Effizienzmessung handelt, bei dem ein möglichst schneller serieller Algorithmus mit dem zu untersuchenden parallelen Algorithmus verglichen wird. Dieses Kriterium ist sehr variabel, wenn nicht bewiesen werden kann, welcher serielle Algorithmus der schnellste ist. Solche Vergleiche sind natürlich immer nur dann sinnvoll, wenn das gleiche zu lösende Problem zugrunde gelegt wird.

Man müßte Algorithmus 4.1.1S36 zusammen mit den **Testproblemen** nun noch mit einem anderen parallelen Algorithmus für **EU-Probleme** vergleichen, um weitergehende Aussagen zur Effizienz zu erhalten. Bei derartigen Effizienzvergleichen ist jedoch immer darauf zu achten, daß die Ergebnisse mit einer reproduzierbaren, vergleichbaren Genauigkeit vorliegen.

Abschließend bemerken wir noch, daß Algorithmus 4.1.1S36 nicht skalierbar ist, weil er in der beschriebenen Variante nur auf einer a priori festgelegten Zahl von Prozessoren lauffähig ist.

5 Adaptivität

5.1 Ebenen der Adaptivität

Neben der hierarchischen Anordnung der parallelen Teilalgorithmen kann man Algorithmus 4.1.1S36 auch nach dem Gesichtspunkt des Adaptivitätsgrads gliedern. Hierzu werden zunächst informelle Definitionen der Adaptivität aufgestellt. Generell ist **adaptiv** als "**sich selbst anpaßend an**" zu lesen.

Definition 5.1.1 (Adaptivität) :

> Ein Algorithmus soll **benutzer-adaptiv** genannt werden genau dann, wenn er ein von den Benutzereingaben abhängiges Verhalten zeigt.
>
> Ein Algorithmus soll **problem-adaptiv** genannt werden genau dann, wenn er ein aus einer zulässigen Problemklasse gestelltes Problem analysiert und für die einzelnen Teilprobleme Teilalgorithmen nach festgelegten Kriterien auswählt und die Gesamtlösung aus den Teilproblemlösungen zusammensetzt. Hierbei wird Wert darauf gelegt, daß sowohl an den Teilproblemen als auch am Gesamtproblem möglichst wenig reduziert wird.
>
> Ein Algorithmus soll **ressourcen-adaptiv** genannt werden genau dann, wenn er abhängig vom Problem und/oder vom Lösungsprozeß selbständig die erforderliche Hardware anfordert und die daraufhin zur Verfügung gestellte in Beschlag nimmt.
>
> Ein Algorithmus soll **lösungs-adaptiv** genannt werden genau dann, wenn er von gewissen Parametern abhängt und diese sich durch Eigenschaften der Näherungslösung des zu lösenden Problems aufgrund festgelegter Kriterien während des Lösungsprozesses ändern. Hierbei können die Parameter zu Beginn des Lösungsprozesses explizit vorgegeben werden (**fremdstartender** Algorithmus) oder werden vom Algorithmus auch zu Beginn selbst gewählt (**selbststartender** Algorithmus).

Die meisten Algorithmen sind **benutzer-adaptiv**.
Man beachte, daß sich **problem-adaptive** Algorithmen von lernfähigen Algorithmen aufgrund der Festlegung der Auswahlkriterien unterscheiden.
Ressourcen-adaptive Algorithmen wählen beispielsweise den Hauptspeicherbedarf oder die Prozessoranzahl selbständig. Fast alle gängigen Algorithmen sind **ressourcen-adaptiv** bzgl. der Dateiein-/ausgabe.
Das Beispiel für **lösungs-adaptive** Algorithmen ist natürlich die adaptive Lösung von gewöhnlichen oder partiellen Differentialgleichungen, weil dort in offensichtlicher Art und Weise eine von der Schritt-/Gitterweite abhängige Diskretisierung benutzt wird, die ihrerseits den algorithmischen Verlauf wesentlich diktiert.

Algorithmus 4.1.1S36 ist **benutzer-adaptiv**, weil der Anwender explizit die geduldete **Fehlertoleranz** vorgibt.

Algorithmus 4.1.1S36 wäre **ressourcen-adaptiv**, wenn der angeforderte Hauptspeicher sich nach Dimension des Systems richten würde oder wenn die Ordnung des Verfahrens (und mithin auch die Prozessoranzahl) aufgrund der **Fehlertoleranz** gewählt werden würde.

Es ist nicht ganz klar (aufgrund der sprachlichen Definition), ob der Algorithmus 4.1.1S36 **problem-adaptiv** ist. Dafür spricht das von der rechten Seite abhängige Verhalten des Lösungsalgorithmus. Dagegen spricht die fehlende Aufteilung in Teilprobleme und die Zusammensetzung der Einzellösungen zur Gesamtlösung; es sei denn man wollte das rekursive Vorgehen und sequentielle Abarbeiten der **Basisprobleme** "künstlich" in diesem Sinne interpretieren.

Algorithmus 4.1.1S36 ist in mehreren Ebenen **lösungs-adaptiv**. Zunächst ist die von der Näherungslösung indirekt über die **Fehlerschranke** abhängige Wahl der Schrittweite offensichtlich. Dies bedeutet strenggenommen, daß der Integrationsteilalgorithmus **lösungs-adaptiv** ist !
Wird die rechte Seite nur als Black-Box betrachtet, so kann man nicht erkennen, daß eigentlich eine zweite <u>Ebene der Adaptivität</u> vorliegt. Die Formulierung von **Problem D** zeigt deutlich, daß das aktuell zu lösende **Basisproblem** erst im Verlauf des Lösungsprozesses feststeht und sich folglich ein Parameter ändert: die Basisproblemnummer. Hier ist es der rechte-Seiten-Algorithmus, der adaptiv ist.
Der Gesamtalgorithmus ist strenggenommen nicht **lösungs-adaptiv**, weil er nicht von den beschriebenen Parametern explizit abhängt.
Der Gesamtalgorithmus hängt zwar von der gewählten **Fehlertoleranz** explizit ab, wählt diese aber gerade <u>nicht</u> selbständig.

Man erkennt, wo hier eine sprachliche Schwierigkeit liegt:
Teilalgorithmen hängen von Parametern ab, diese werden im übergeordneten Algorithmus bestimmt und dadurch wird der Teilalgorithmus adaptiv genannt, obwohl er allein sich nicht selbständig anpassen kann. Würde man den jeweils übergeordneten Algorithmus als "adaptiv" bezeichnen, so ergeben sich Schwierigkeiten bei der Beschreibung der Eigenschaften der Parameter.

5.2 Ausblick

Gegenstand weiterer Forschungsarbeiten sollen insbesondere folgende Probleme sein:

Problem 5.2.1 (Adaptivität) :

Die im vorherigen Abschnitt aufgeführten informellen Definitionen sind durch abstrakte, formale Definitionen zu ersetzen.

Problem 5.2.2 (Adaptivität/Parallelität) :

Die Zuammenhänge zwischen Adaptivität und Parallelität sind zu untersuchen.

Problem 5.2.3 (Effizienzmessung) :

Wie läßt sich <u>Effizienz</u> bzgl. der Adaptivität adäquat messen ?

Problem 5.2.4 (Algorithmen) :

Entwicklung von <u>effizienteren</u> parallelen Algorithmen zur Lösung von **Problem D**. Insbesondere sollten neue Algorithmen "stärker" parallelisierbar sein.

Desweiteren sind natürlich auch die in den übrigen Abschnitten aufgeführten Probleme interessant.

6 Testrechnung

6.1 Serielle Ergebnisse

Algorithmus 4.1.1S36 wurde zusammen mit dem Integrationsverfahren 3.6.1S32 und der Fehlerschranke 3.7.2S34 in Borland C Version 5.02 (Modell Win32 Konsole) programmiert und auf einem IBM-kompatiblen Personal-Computer mit Pentium II-CPU bei 266 Mhz, 66 Mhz-Board, ca. 20 Mflops Peak (eigene Messungen) und 64 MByte SD-Ram getestet. Alle Optimierungsoptionen des Compilers wurden ausgeschaltet. Der PC wurde unter Windows 98 im Single-User-Mode betrieben. Da es sich um ein Multitasking-Betriebssystem handelt, sind die aufgelisteten Rechenzeiten mit Vorsicht zu genießen; zur Kontrolle wurden die Messungen sechsmal durchgeführt. Man beachte, daß der Pentium II selbst schon Merkmale eines Parallelrechners (z.B. Pipelines) besitzt.

Die Maschinengenauigkeit beträgt ca. $2 * 10^{-16}$. Die verwendete **Fehlertoleranz** wurde relativ groß gewählt, weil dadurch die Robustheit der **Fehlerschranke** besser getestet werden kann: 10^{-5}. Für kleinere Toleranzen ergeben sich entsprechend genauere Ergebnisse.

Das verwendete **DPIRKG** besitzt die Ordnung 8 und die dichte Ordnung 4. Der Interpolationsgrad beträgt maximal 7. Die Defektdifferentialgleichung wird mit zwei Schritten eines **PIRKG(4,4)** "gelöst".

Die Parameter $c_0, c, d, \hat{s}_{-1}$ von Beispiel 2.3.1S10 sind jeweils angegeben; ebenso i_{max} in Beispiel 2.3.3S12.

Die maximal zuläßige Schrittweite h_{max} ist ein wesentlicher Parameter des Algorithmus; es ist klar, daß durch eine kleinere Wahl von h_{max} automatisch bessere Ergebnisse erzielt werden, unabhängig von der Wahl des Verfahrens !
Dies trifft allerdings nur dann zu, wenn nicht zu viele Integrationsschritte durchgeführt werden, weil der in jedem Schritt eingeführte Rundungsfehler schließlich dominieren kann (vergleiche Seite 127 ff in [Werner/Arndt]).
Andererseits wird jedes Verfahren bei einer zu großen Wahl von h_{max} versagen, weil dann mehrere **Unstetigkeiten** innerhalb eines Integrationsschrittes auftreten können, die sich quasi gegenseitig "ausgleichen" können; siehe hierzu Seite 45.
In allen gerechneten Tests wird die minimale Schrittweite (siehe S30) niemals erreicht.
Um den Rundungsfehlereinfluß gering zu halten, sollte der zuläßige Bereich für die Schrittweite eigentlich noch stärker eingeschränkt werden; alternativ ließe sich auch die **Fehlertoleranz** nach unten hin begrenzen.

Die Spalten "steps" und "fails" enthalten die Anzahlen der erfolgreich ausgeführten Integrationsschritte und der wiederholten Integrationsschritte.

Tabelle 6.1.1 (Beispiel 2.3.1S10) :

c_0	c	d	$\hat{s}_{-1}$	h_{max}	y(1)	steps	fails
10.0	10.0	0.0039	0.0	0.01	1.4e-07	1972	7555
1000.0	1000.0	0.0039	0.0	0.01	1.6e-05	2462	8826
10.0	10.0	0.0039	1000.0	0.01	9.99997e+02	330	860
1000.0	1000.0	0.0039	1000.0	0.01	9.99992e+02	664	3500

Trotz der zu groß gewählten Schrittweite $h_{\max}$ (der minimale Abstand zwischen 2 **Übergangspunkten** beträgt $\frac{0.0039}{4} \approx 0.001$) werden alle **Unstetigkeiten** erkannt (nicht aus der Tabelle ersichtlich).

Die Ergebnisse zu Ellisons Beispiel sind in Tabelle 6.1.2S44 und Tabelle 6.1.3S44 enthalten.

Tabelle 6.1.2 (Beispiel 2.3.2S11) :

h_{max}	y(12.5)	steps	fails
0.9	-1.1269268e+00	86	377

Tabelle 6.1.3 (Beispiel 2.3.2S11 (Übergangspunkte)) :

x_k
1.570796327
3.701322073
4.938115475
7.193558464
8.369355454
9.765111831
11.10419835

Die CPU-intensive Testserie mit Beispiel 2.3.3S12 läuft ebenfalls ohne Komplikationen ab und liefert die folgenden Rechenzeiten:

Tabelle 6.1.4 (Beispiel 2.3.3S12 (Zeit in Sekunden)) :

	Gesamt		nur f-Auswert.			
i_{max}	min	max	min	max	mittel	overhead
30	11	12	3	7	5	7
300	13	14	5	10	8	6
3000	29	30	19	25	21	9
30000	193	194	183	192	187	7
90000	559	-	550	-	-	9

Für $i_{max} = 90000$ wurde nur eine Messung durchgeführt.
Man erkennt die prozentuale Abnahme des Anteils des overheads.
Für alle Werte von i_{max} lauten die Ergebnisse:

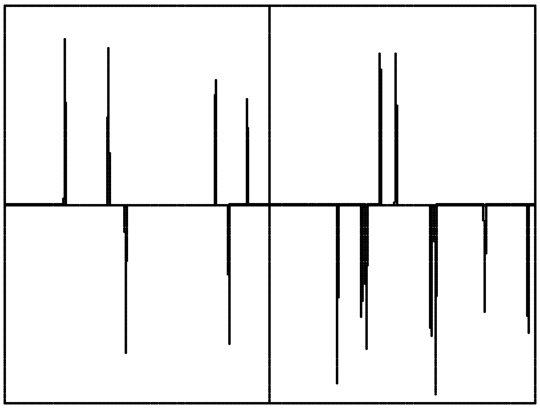
Tabelle 6.1.5 (Beispiel 2.3.3S12) :

Dimension	h_{max}	$y(\ln(1 + 25 * \pi))$	steps	fails
3	0.9	5.3e-06	319	738

Wiederum werden trotz zu groß gewählter Schrittweite alle Nullstellen (=**Unstetigkeiten=Übergangspunkte**) gefunden.

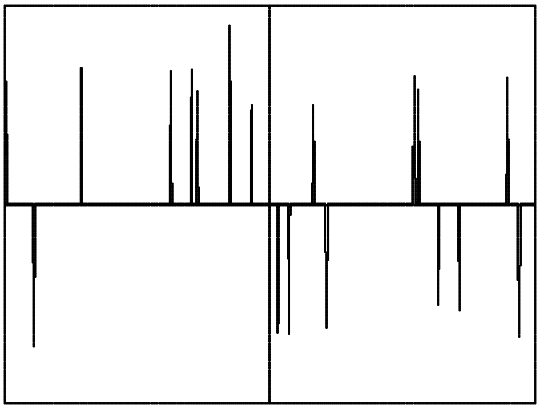
Die Lösung von Beispiel 2.3.1S10 kann so aussehen:

Abbildung 6.1.6 (Interval (0,0.5) (15 Spitzen)) :



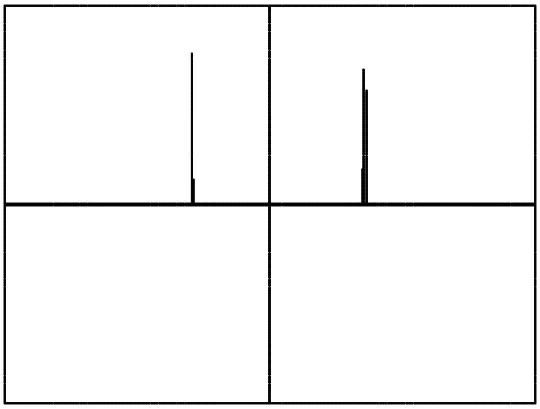
Die unterschiedliche Höhe rührt vom gewollten Zufallseinfluß her. Die Spitzenhöhe ist mindestens 1% genau !

Abbildung 6.1.7 (Interval (0.5,1) (18 Spitzen)) :



Sie kann aber auch so aussehen, wenn $h_{max} = 0.2$ gewählt wird !

Abbildung 6.1.8 (Interval (0,1) (fehlerhaft)) :



Auswertung

Auf den ersten Blick ist die verwendete Schrittweitensteuerung furchtbar ineffizient. Allerdings ist zu bedenken, daß alle Unstetigkeiten trotz zu großer Maximalschrittweite automatisch erkannt werden, allein durch Ausnutzung von Black-Box-f-Information !

Die Belastung der CPU zeigt, daß ab ca. 1000 Additionen/ Divisionen pro Komponente der rechten Seite diese den wesentlichen Anteil am Gesamtaufwand stellt.

Zur allgemeinen Methodik läßt sich sagen, daß man aus einer "großen" Abweichung der exakten Lösung am Endpunkt von der Näherungslösung direkt auf ein Versagen des benutzten Verfahrens schließen kann. Umgekehrt können auch noch so viele erfolgreiche Tests nicht die Güte eines Vefahrens beweisen.

6.2 Parallele Ergebnisse

Algorithmus 4.1.1S36 wurde zusammen mit dem Integrationsverfahren 3.6.1S32 und der Fehlerschranke 3.7.2S34 auf dem Cray Multi-Vektorprozessor-System J932/16-8192 getestet. Zu Vergleichszwecken lief das Programm sowohl auf vier als auch auf nur einem Prozessor.
Weil die erzielte Genauigkeit nicht wesentlich von den seriellen Testergebnissen abweicht (insbesondere werden alle **Übergangspunkte** erkannt), werden nur die Rechenzeiten zu Test 2.3.3S12 angegeben.

Das Programm wurde mit den folgenden **C**-Compileroptionen übersetzt:

```
cc -DJCRAY -O0 -h msglevel_0 -h report=istv -o super s.c -lcraylm 2> err
```

Insbesondere sind also alle Optimierungen ausgeschaltet. Die Parallelisierung wurde mit Hilfe der Compilerdirektiven

```
#pragma _CRI parallel shared(np1a,np1b,np1c,np1d,BIGY,RJ,RJM1,gsmuell),
#pragma _CRI case,
#pragma _CRI endcase,
#pragma _CRI guard,
gsmuell = 0.0;
#pragma _CRI endguard,
und
#pragma _CRI endparallel
```

durchgeführt.
Die Zahl der angeforderten Prozessoren wird mit Hilfe von

```
setenv NCPUS 1
bzw.
setenv NCPUS 4
```

in der Shell oder der Batchdatei/dem Shellscript festgelegt.

Erläuterung der Tabellen:

Der Parameter i_{max} steht für die Komplexität des Problems (siehe 2.3.3S12).
Die Rechenzeit ist ein Mittelwert aus drei Messungen und wird pro Prozessor angegeben.
Der Overhead wird bzgl. der Gesamtzeit angegeben.

Tabelle 6.2.1 (Paralleles Beispiel 2.3.3S12) :

		Rechenzeit in s			
i_{max}	Prozessoren	Gesamt	nur f-Auswert.	overhead in %	Speeedup
9000	1	771	743	3,6	
9000	4	428	278	35,1	1.8

Verwendet man die folgenden **C**-Compileroptionen

```
cc -DJCRAY -O3 -h msglevel_0 -h report=istv -o super s.c -lcraylm 2> err,
```

so ist der höchste Optimierungslevel eingeschaltet. Dies bedeutet u.a., daß aggressiv vektorisiert wird.
Weil die Rechenzeiten pro Benutzer im Dialogbetrieb beschränkt sind, muß man das Programm "super" mit Hilfe eines Batchjobs ausführen:

```
#!/usr/local/bin/tcsh
#QSUB -q crassus
#QSUB -lM 60MB
#QSUB -lT 3300
#QSUB -eo
#QSUB -r super
setenv NCPUS 4
cd
cd super1
super<i90k>o90k.txt
```

Man erkennt, welchen Vorteil die Vektorisierung im Vergleich zur Parallelisierung einbringt:

Tabelle 6.2.2 (Optimiertes Beispiel 2.3.3S12) :

		Rechenzeit in s			
i_{max}	Prozessoren	Gesamt	nur f-Auswert.	overhead in %	Speeedup
90000	1	624	585	6,25	
90000	4	486	190	60,9	1,3

Obwohl der Parameter i_{max} zehnmal so groß ist, wird annähernd die gleiche Zeit benötigt. Es ist ersichtlich, wie wesentlich die Annahme ist, daß der Aufwand für die f-Auswertungen dominiert. Nur in diesem Fall (siehe Tabelle 6.2.1S48) läßt sich ein "erträgliches" Speedup erzielen.

Wenn man die Zeit für die f-Auswertungen auf vier Prozessoren in den obigen Tabellen mit vier multipliziert, dann ergeben sich deutlich größere Werte als auf einem Prozessor:

$$4 * 278 = 1112 > 743 \text{ und } 4 * 190 = 760 > 585.$$

Dies deutet daraufhin, daß ein bedeutender, zusätzlicher Aufwand durch das Laden der einzelnen Prozessoren, die alle im Time-Sharing betrieben werden, entsteht.

Zusammenfassend kann man sagen, daß zumindest in den hier betrachteten Beispielen die Vektorisierung mehr einbringt, als die Parallelisierung.

6.3 Vergleich

Die numerische Qualität der Ergebnisse von PC und Vektorrechner ist vergleichbar gut.

Erwartet wurde eine Geschwindigkeitssteigerung des parallelen Programms in der Größenordnung Faktor 3, d.h. das Programm läuft auf vier(!) Prozessoren dreimal so schnell wie auf einem.
Tatsächlich ergab sich ein geringeres Speedup, weil:

1. der Aufwand für die f-Auswertungen nicht stark genug dominiert,
2. aufgrund des Time-sharing der Cray-Multi-Vektorrechner aus Benutzersicht nicht als idealer Parallelrechner aufgefaßt werden kann und
3. die parallele Programmierung des Algorithmus 4.1.1S36 nicht weit genug vorangetrieben wurde.

Wenn man dem erzielten Speedup den Entwicklungs- und Programmieraufwand gegenüberstellt, dann gelangt man zu den Erkenntnissen:

1. Auf Vektorrechnern sollte man vektorisieren (oh - ha !).
2. "Parallelisierungen im Kleinen" (mit wenigen Prozessoren) lohnen sich nicht.

Bemerkenswert ist außerdem, daß der PC im Single-user-mode genauso schnell ist, wie der Vektorrechner im Time-sharing-Betrieb (vergleiche die Zeiten für $i_{max} = 90000$) !

Literatur

[Aubin/Cellina] J.P. Aubin, A. Cellina.
Differential inclusions.
Springer Berlin 1984.

[Berz(1)] Edgar Berz.
Infinitesimalrechnung Band 1.
Viewig uni-text.Braunschweig 1977.

[Berz(2)] Edgar Berz.
Infinitesimalrechnung Band 2.
Viewig uni-text.Braunschweig 1977.

[Bogacki/Shampine] P. Bogacki, L.F. Shampine.
Interpolating high-order Runge-Kutta formulas.
Computers Math. Applic. 20(3), pp 15-24, 1990.

[Butcher(1)] John Charles Butcher.
The numerical analysis of ordinary differential equations. Runge-Kutta and general linear methods.
John Wiley & sons Chichester 1987.

[Butcher(2)] John Charles Butcher.
Implicit Runge-Kutta Integration Processes.
Math. Comp. 18, pp 50-64, 1964.

[DonGen] Jack J. Dongarra, Wolfgang Gentzsch.
Computer Benchmarks.
series: Advances in parallel computing 8
edited by Gerhard R. Joubert, Udo Schendel.
North-Holland New York 1993.

[Dopri(1)] John R. Dormand, Pete J. Prince.
A family of embedded Runge-Kutta fomulae.
Journal of Computational and Applied Mathematics 6(1), pp 19-26, 1980.

[Dopri(2)] John R. Dormand, Pete J. Prince.
High order embedded Runge-Kutta fomulae.
Journal of Computational and Applied Mathematics 7(1), pp 67-75, 1981.

[Dopri(3)] John R. Dormand, Pete J. Prince.
A reconsideration of some embedded Runge-Kutta fomulae.
Journal of Computational and Applied Mathematics 15(2), pp 203-211, 1986.

[Dopri(4)] John R. Dormand, Pete J. Prince.
Runge-Kutta-triples.
Comp. & Maths. with Appls. 12A(9), pp 1007-1017, 1986.

[Dopri(5)] John R. Dormand, Pete J. Prince.
Practical Runge-Kutta processes.
SIAM J. Sci. Stat. Comput. 10(5), pp 977-989, 1989.

[Ellison] D. Ellison.
Efficient automatic integration of ODEs with discontinuities.
Mathematics and Computers in Simulation 23, pp 12 - 20, 1981.

[EnJaNoTho] W.H. Enright, K.R. Jackson, S.P. Nørsett, P.G. Thomsen.
Effective Solution of Discontinous IVPs Using a Runge-Kutta Formula Pair with Interpolants.
Applied Mathematics and Computations 27, pp 313-335, 1988.

[Foellinger] O. Foellinger.
Einführung in die Dynamik von Regelkreisen. Teil I.
AEG Institut für Automation IAZ/INF/3004.

[Gear/Osterby] C.W. Gear, O. Østerby.
Solving ordinary differential equations with discontinuities.
ACM Transactions on Mathematical Software 10(1), pp 23-44, 1984.

[Gustafsson] Kjell Gustafsson.
Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods.
ACM Transactions on Mathematical Software 17(4), pp 533-554, 1991.

[HWN] Ernst Hairer, Gerhard Wanner, Syvert Nørsett.
Solving ordinary differential equations 1. Nonstiff problems.
Springer. Berlin 1987.

[Heuser] Harro Heuser.
Gewöhnliche Differentialgleichungen.
B.G. Teubner. Stuttgart 1989.

[Heuser(2)] Harro Heuser.
Funktionalanalysis.
B.G. Teubner. Stuttgart 1986.

[HouSom] P.J. van der Houwen; B.P. Sommeijer.
Parallel iteration of high-order Runge-Kutta methods with stepsize control.
Journal of Computational and Applied Mathematics 29, pp 111-127, 1990.

[JaNo] K.R. Jackson, S.P. Nørsett.
The potential for parallelism in Runge-Kutta methods.
Part 1: RK formulas in standard form.
SIAM Journal on Numerical Analysis 32(1), pp 49-82, 1995.

[JaKvaeNo] K.R. Jackson, A. Kværnø, S.P. Nørsett.
The use of Butcher series in the analysis of Newton-like iterations in Runge-Kutta formulas.
Applied Numerical Mathematics 15, pp 341-356, 1994.
und
An analysis of the order of Runge-Kutta methods that use an iterative scheme to compute their internal stage values.
BIT 36(4), pp 713-765, 1996.

[Mannshardt] Reinhold Mannshardt.
One-Step Methods of Any Order for Ordinary Differential Equations with Discontinuous Right-Hand Sides.
Numerische Mathematik 31, pp 131-152, 1978.

[Schaefer] Peter Schäfer.
Numerische Lösungsverfahren für modulare Differentialgleichungssysteme mit Unstetigkeiten zur Simulation technischer Systeme.
Dissertation. Technische Universität Clausthal 1992.

[Schendel(1)] Udo Schendel.
Einführung in die Numerische Mathematik Teil I.
Vorlesungsskript der Freien Universität Berlin 1993.

[Schendel(2)] Udo Schendel.
Einführung in die parallele Numerik.
Vorlesungsskript der Freien Universität Berlin 1996.

[Schwarz] Hans Rudolf Schwarz.
Numerische Mathematik.
B.G. Teubner. Stuttgart 1988.

[ShaGladBra] L.F. Shampine, I. Gladwell, R.W. Brankin.
Reliable Solution of Special Event Location Problems for ODEs.
ACM Transactions on Mathematical Software 17(1), pp 11-25, 1991.

[Skeel] Robert D. Skeel.
Thirteen Ways to Estimate Global Error.
Numerische Mathematik 48, pp 1-20, 1986.

[Tews] Joachim Tews.
Diplomarbeit.
Fachbereich Mathematik und Informatik. Freie Universität Berlin 1995.

[Werner/Arndt] Helmut Werner, Herbert Arndt.
Gewöhnliche Differentialgleichungen. Eine Einführung in Theorie und Praxis.
Springer Hochschultext. Berlin 1986.