Bergische Universität
GH Wuppertal

Steigungsarithmetiken in C – XSC

M. Bräuer, W. Hofschuster, W. Krämer

Preprint 2001/3

Wissenschaftliches Rechnen/
Softwaretechnologie

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster Wissenschaftliches Rechnen/Softwaretechnologie Fachbereich 7 (Mathematik) Bergische Universität GH Wuppertal Gaußstr. 20 D-42097 Wuppertal

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadresse

Dipl.-Math. M. Bräuer
Dr. W. Hofschuster
Prof. Dr. W. Krämer
Bergische Universität GH Wuppertal
Gaußstr. 20
D-42097 Wuppertal

e-mail: `xsc@math.uni-wuppertal.de`
WWW: `http://www.math.uni-wuppertal.de/~xsc/`

C – XSC 2.0 is freely available from
`http://www.math.uni-wuppertal.de/~xsc/xsc/download.html`

The software for slope computations is freely available from
`http://www.math.uni-wuppertal.de/~xsc/xsc/download.html`

Inhaltsverzeichnis

Zusammenfassung

Steigungsarithmetiken in C – XSC: Es werden Steigungsarithmetiken im Vorwärts- und Rückwärtsmodus eingeführt. Weiter werden Implementierungen in C – XSC beschrieben und deren Einsatz an Beispielprogrammen demonstriert. Die Verwendung von Steigungen in Verifikationsalgorithmen führt im Vergleich zur Verwendung der automatischen Differentiation in der Regel zu engeren Einschließungen. Außerdem können im Gegensatz zur automatischen Differentiation beim Einsatz einer Steigungsarithmetik auch nichtdifferenzierbare Funktionen behandelt werden.

Die in der Arbeit vorgestellten Routinen werden derzeit an die neue Version C – XSC 2.0 [8] angepasst.

Abstract

Slope arithmetics in C – XSC: An introduction to slope arithmetics in forward and reverse mode is given. Implementations in C – XSC are described. Sample programs illustrate how to use the software package.

Key words: Automatic slope generation, slopes, forward method, backward method, selfverifying algorithms, C – XSC

1 Motivation und Grundlagen

Es werden die Bezeichnungen der im WEB verfügbaren Arbeit [4] verwendet. Dort findet man insbesondere auch die Definitionen von Begriffen wie Code-Liste und Darstellungsgraph. Auch wird dort im Zusammenhang mit der automatischen Differentiation der allgemeine Ansatz von sogenannten Rückwärtsmethoden vorgestellt.

Zur Motivation wird das folgende nichtlineare Gleichungssystem betrachtet, welches in jeder Gleichung nichtdifferenzierbare Anteile enthält:

$$\left.\begin{array}{rcl} x_1^2 - \mathrm{e}^{x_2} + |x_2| & = & 0 \\ x_1^2 - x_2^2 - |x_1| & = & 0. \end{array}\right\} \tag{1}$$

Herkömmliche Verfahren (z.B. das Newton-Verfahren), welche Ableitungen benutzen, sind hier nicht anwendbar. Dagegen kann eine Lösung unter Verwendung von Intervallsteigungen, deren Berechnung automatisierbar ist, ermittelt werden (vgl. Abschnitt 6). Der vorliegende Abschnitt stellt zunaächst einige grundlegende Definitionen bereit.

Definition 1.1 (Steigung) *Für die Funktion $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ heißt die vektorwertige Funktion $s_f : D \times D \to \mathbb{R}^n$, welche die Gleichung*

$$f(x) = f(c) + s_f(c, x)\ (x - c) \tag{2}$$

erfüllt, Steigung von f bezüglich $x \in D$ und $c \in D$.

Geometrisch interpretiert entspricht $s_f(c, x)$ aus (2) für $x \neq c$ im eindimensionalen Fall ($n = 1$) dem Anstieg der Sekante, die durch die beiden Punkte $(x, f(x))$ und $(c, f(c))$ verläuft.

Definition 1.2 (Intervallsteigung) *Die Menge $S_f \subseteq \mathbb{R}^n$, welche durch*

$$S_f = s_f(C, X) := \{s_f(c, x) \mid x \in X,\, c \in C,\, x \neq c\}$$

definiert ist, wird Intervallsteigung von f bezüglich $\mathbb{IR} \ni C, X \subseteq D$ genannt.

Mit dieser Definition gilt $f(x) \in f(c) + S_f \cdot (x - c) \subseteq f(c) + S_f \cdot (X - c)$, wenn man die Operationen als Potenzmengenoperationen auffaßt und $S_f \neq \emptyset$ gilt. Ist S_f ein Intervall bzw. Intervallvektor, so handelt es sich bei den Operationen um Intervallverknüpfungen.

Definition 1.3 (Faktorisierbare Funktion) *Sei $\pi_i^n : \mathbb{R}^n \to \mathbb{R}$ die Projektion eines reellen n-Tupels auf seine i-te Komponente, d.h. $\pi_i^n(x_1, x_2, \ldots, x_n) = x_i$. Eine Funktion $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ heißt faktorisierbar, falls es eine endliche Folge von Funktionen $f_1, \ldots, f_k$ gibt, für die gilt:*

(1) $f_j : D \to \mathbb{R}$ für alle $j \in \{1, \ldots, k\}$,
(2) $f = f_k$,
(3) $f_1 = \pi_1^n, f_2 = \pi_2^n, \ldots, f_n = \pi_n^n$ und
(4) für $n < j \leq k$: $f_j = g(f_l)$ mit $g \in \mathcal{F}$ und $l < j$ oder
$f_j = f_l \circ f_r$ mit $\circ \in \mathcal{OP}$[1] und $l, r < j$ oder
f_j ist eine reelle Konstante.

Das Problem, das bei Verzweigungen (if-Anweisungen in der Definition der Funktion f) auftreten kann, soll am folgenden Beispiel demonstriert werden (entnommen aus [5], unmittelbar übertragbar auf Steigungsberechnungen):

Beispiel 1.1 *Sei $f : \mathbb{R} \to \mathbb{R}$ durch*

$$f(x) = \begin{cases} x^2 & , \quad x \neq 1 \\ 1 & , \quad x = 1 \end{cases}$$

gegeben. Gesucht sei die erste Ableitung an der Stelle $x_0 = 1$. Programmiertechnisch wird diese Funktion durch eine `if then else`*-Anweisung realisiert. Das Problem, das sich bei der automatischen Differentiation von f in $x_0 = 1$ ergibt, besteht darin, daß hier f fälschlicherweise als konstante Funktion $f \equiv 1$ angesehen wird und sich deshalb für die Ableitung der falsche Wert $f'(1) = 0$ ergibt. Jedoch repräsentiert die Funktion f nichts anderes als die quadratische Funktion $g(x) = x^2$. Man müßte somit $f'(1) = g'(1) = 2$ erhalten.*

Einen allgemein gangbaren Ausweg gibt es hier nicht. Berechnet man jedoch Einschließungen der Ableitungswerte, so schlägt Kearfott in [11] vor, eine spezielle Verzweigungsfunktion χ zu benutzen. Mit Hilfe dieser Verzweigungsfunktion erhält man Einschließungen aller Ableitungswerte. Sie ist wie folgt definiert:

[1] Entspricht $\circ$ dem unären $-$, so wird immer davon ausgegangen, daß nur ein Operand in die Rechnung eingeht.

Definition 1.4 (Verzweigungsfunktion)

$$\chi(x_s,\, x_q,\, x_r) := \begin{cases} x_q & , \quad x_s < 0 \\ x_r & , \quad \textit{sonst} \end{cases} \tag{3}$$

Die in Beispiel 1.1 vorgestellte Funktion f kann dann durch zweimalige Anwendung von (3) durch $f(x) = \chi(x-1,\, x^2,\, \chi(1-x,\, x^2,\, 1))$ ausgedrückt werden.
Mit den Einschließungen X_s, X_q und X_r der Werte x_s, x_q und x_r ergibt sich eine intervallmäßige Erweiterung von (3) mittels

Definition 1.5 (Verzweigungsfunktion, intervallmäßig)

$$\chi(X_s,\, X_q,\, X_r) := \begin{cases} X_q & , \quad X_s < 0 \\ X_r & , \quad X_s > 0 \\ X_q \cup X_r & , \quad \textit{sonst.} \end{cases} \tag{4}$$

Sei nun eine Funktion

$$f(x) = \begin{cases} f_1(x) & , \quad g(x) < 0 \\ f_2(x) & , \quad \text{sonst} \end{cases}$$

mit faktorisierbaren Funktionen $f_1, f_2, g : D \subseteq \mathbb{R} \to \mathbb{R}$ gegeben. Symbolische Differentiation von f nach x liefert

$$f'(x) = \frac{d\chi(g(x),\, f_1(x),\, f_2(x))}{dx} = \chi(g(x),\, {f_1}'(x),\, {f_2}'(x))\,.$$

Existieren dann die Intervallerweiterungen der Funktionen f_1, f_2 und g für eine Einschließung X_0 eines Verzweigungspunktes x_0, so erhält man durch (4) ein Intervall, welches die rechts- und linkseitigen Ableitungen von f an der Stelle x_0 enthält. Für die Funktion aus Beispiel 1.1 bedeutet das:

Beispiel 1.2 (Fortsetzung von Beispiel 1.1) *Für die Ableitung f' gilt:*

$$\begin{aligned} f'(x) &= \frac{d\chi(x-1,\, x^2,\, \chi(1-x,\, x^2,\, 1))}{dx} \\ &= \chi(x-1,\, 2x,\, \frac{d\chi(1-x,\, x^2,\, 1)}{dx}) \\ &= \chi(x-1,\, 2x,\, \chi(1-x,\, 2x,\, 0)). \end{aligned}$$

Eine intervallmäßige Einschließung Y von $f'(x)$ ergibt sich unter Verwendung von (4):

$$Y = \begin{cases} 2X & , \quad X - 1 < 0 \\ \chi(1-X,\, 2X,\, 0) & , \quad X - 1 > 0 \\ 2X \cup \chi(1-X,\, 2X,\, 0) & , \quad \textit{sonst} \end{cases}$$

wobei $x \in X$. Setzt man speziell $X_0 = [1,\, 1]$, so ist $Y_0 = 2X_0 \cup (2X_0 \cup 0) = [0,\, 2]$ eine Einschließung von $f'(1)$.

Nichtdifferenzierbare Funktionen können in das Konzept der faktorisierbaren Funktionen aufgenommen werden, indem man die Menge der elementaren Funktionen erweitert. So können $\max(.,.)$, $\min(.,.)$, $\mathrm{abs}(.)$ und die oben vorgestellte Verzweigungsfunktion $\chi(.,.,.)$ zugelassen werden.

Definition 1.6 *Mit* $\mathcal{F}_e := \{\text{abs}(.),\ \max(.,.),\ \min(.,.),\ \chi(.,.,.)\}$ *wird die Menge der erweiterten elementaren Funktionen* $\mathcal{F}'$ *durch* $\mathcal{F}' = \mathcal{F} \cup \mathcal{F}_e$ *definiert. Eine Funktion heißt dann faktorisierbar, wenn Schritt 4 in Definition 1.3 durch*

(4') für $n < j \leq k$: $f_j = g(f_l)$, $f_j = g(f_l, f_r)$ *bzw.* $f_j = g(f_p, f_l, f_r)$ $(g \in \mathcal{F}'$ *und* $p, l, r < j)$ *oder*
$f_j = f_l \circ f_r$ $(\circ \in \mathcal{OP}$ *und* $l, r < j)$ *oder*
f_j *ist eine reelle Konstante*

ersetzt wird (vgl. S. 2).

Bemerkung 1.1 *Funktionen, die Intervalle als Argumente haben, können in gleicher Art und Weise als faktorisierbar erklärt werden, indem man die elementaren Operationen und Funktionen als Intervalloperationen bzw. Intervallfunktionen auffaßt und ebenfalls mit* $\mathcal{OP}$ *bzw.* $\mathcal{F}'$ *bezeichnet. Dann sind auch Intervalle als Konstanten zugelassen.*

2 Vorwärtsrechnung

Der eindimensionale Fall

Um Intervallsteigungen zu berechnen, kann das Prinzip der Vorwärtsrechnung der automatischen Differentiation übernommen werden. Dabei sind bestimmte Größen beim Durchlaufen des Darstellungsgraphen an jedem Knoten aus den Vorgängerknoten zu berechnen. Es handelt sich hierbei um Intervalltripel, welche die notwendige Informationen für die Neuberechnung der Intervallsteigungen an jedem Knoten beinhalten. Solche Tripel werden nun näher charakterisiert.

Definition 2.1 *Sei* $f : D \subseteq \mathbb{R} \to \mathbb{R}$ *und* $C, X \subseteq D$ *gegeben. Ein Tripel* $(f_x, f_c, f_s) \in I\mathbb{R}^3$ *heißt Steigungstripel bezüglich* f, C *und* X, *falls gilt:*

$$\begin{aligned} \forall x \in X &: f(x) \in f_x \\ \forall c \in C &: f(c) \in f_c \\ \forall x \in X\ \forall c \in C\ \exists s \in f_s &: f(x) = f(c) + s \cdot (x - c). \end{aligned}$$

Bemerkung 2.1 *Im folgenden wird zur Vereinfachung* $C \subseteq X$ *vorausgesetzt. Diese Annahme stellt jedoch keine Beschränkung der Allgemeinheit dar.*

Der folgende Satz zeigt, wie Tripel an den Knoten des Darstellungsgraphen aus den Tripeln der Vorgängerknoten hervorgehen, falls es sich bei den Knotenoperationen um Verknüpfungen aus $\mathcal{OP}$ oder $\mathcal{F}$ handelt.

Satz 2.1 *Die Tripel* $(\lambda, \lambda, 0)$ *und* $(X, C, 1)$ *seien die zu den beiden Funktionen* $h_1(x) \equiv \lambda$ $(\lambda \in \mathbb{R})$ *und* $h_2(x) = x$ *gehörenden Steigungstripel bezüglich* C, $X \subseteq \mathbb{R}$. *Weiterhin seien Funktionen* $f, g : D \subseteq \mathbb{R} \to \mathbb{R}$ *sowie deren Steigungstripel* (f_x, f_c, f_s) *und*

(g_x, g_c, g_s) bezüglich C, $X \subseteq D$ gegeben. Dann gilt für das Steigungstripel (h_x, h_c, h_s) von $h = f \circ g$ mit $\circ \in \mathcal{OP}$:

$$\left.\begin{array}{ll} h_x = f_x \circ g_x\,,\, h_c = f_c \circ g_c & \circ \in \{-,+,\cdot,/\} \\ h_s = f_s \circ g_s & \circ \in \{-,+\} \\ h_s = f_s \cdot g_x + f_c \cdot g_s & \circ = \cdot \\ h_s = (f_s - h_c \cdot g_s)/g_x & \circ = / \text{ und } 0 \notin g_x. \end{array}\right\} \tag{5}$$

Existiert $\varphi(f_x)$ ($\varphi \in \mathcal{F}$) und ist $E_\varphi(f_c, f_x)$ eine Einschließung von $s_\varphi(f_c, f_x)$, so erhält man ein Steigungstripel für $h = \varphi(f)$ aus

$$\begin{array}{l} h_x = \varphi(f_x)\,,\, h_c = \varphi(f_c) \quad \text{und} \\ h_s = E_\varphi(f_c, f_x) \cdot f_s. \end{array} \tag{6}$$

Bemerkung 2.2 *Für $\circ \in \{\cdot, /\}$ kann (5) durch*

$$h_s = (f_s \cdot g_x + f_c \cdot g_s) \cap (g_s \cdot f_x + g_c \cdot f_s) \text{ und} \tag{7}$$

$$h_s = ((f_s - h_c \cdot g_s)/g_x) \cap ((f_s - h_x \cdot g_s)/g_c) \tag{8}$$

verbessert werden. Führt man außerdem die Zuordnung

$$h_x := h_x \cap (h_c + h_s(X - C)) \tag{9}$$

durch, so kann möglicherweise auch die Einschließung von h_x verbessert werden.

Beweis: Zum Beweis von Satz 2.1 und Bemerkung 2.2 siehe z.B. [20]. □

Wie gewinnt man möglichst gute Einschließungen für $s_\varphi(f_c, f_x)$ ($\varphi \in \mathcal{F}$)? Falls $\varphi \in \mathcal{F}$ auf f_x stetig differenzierbar ist, gilt zwar $s_\varphi(f_c, f_x) \subseteq \varphi'(f_x)$, das Intervall $\varphi'(f_x)$ liefert jedoch im allgemeinen eine zu grobe Einschließung von $s_\varphi(f_c, f_x)$ ([20]). Spezielle Aussagen lassen sich jedoch für (lokal-)konvexe bzw. (lokal-)konkave Funktionen treffen. Es gilt der folgende

Satz 2.2 *Sei $\varphi : D \subseteq \mathbb{R} \to \mathbb{R}$, $\varphi \in C^{(1)}(D)$ mit $X, C \in \mathbb{IR}$, $C \subseteq X \subset D$. Mit*

$$\widehat{s}_\varphi(c, x) := \begin{cases} s_\varphi(c, x) & x \neq c \\ \varphi'(x) & x = c \end{cases}$$

ergeben sich die folgenden Aussagen.
Ist φ auf X konvex, so gilt:

$$\widehat{s}_\varphi(\underline{c}, \underline{x}) \leq s_\varphi(c, x) \leq \widehat{s}_\varphi(\overline{c}, \overline{x}) \quad c \in C,\ x \in X,\ c \neq x. \tag{10}$$

Ist φ auf X konkav, so gilt:

$$\widehat{s}_\varphi(\overline{c}, \overline{x}) \leq s_\varphi(c, x) \leq \widehat{s}_\varphi(\underline{c}, \underline{x}) \quad c \in C,\ x \in X,\ c \neq x. \tag{11}$$

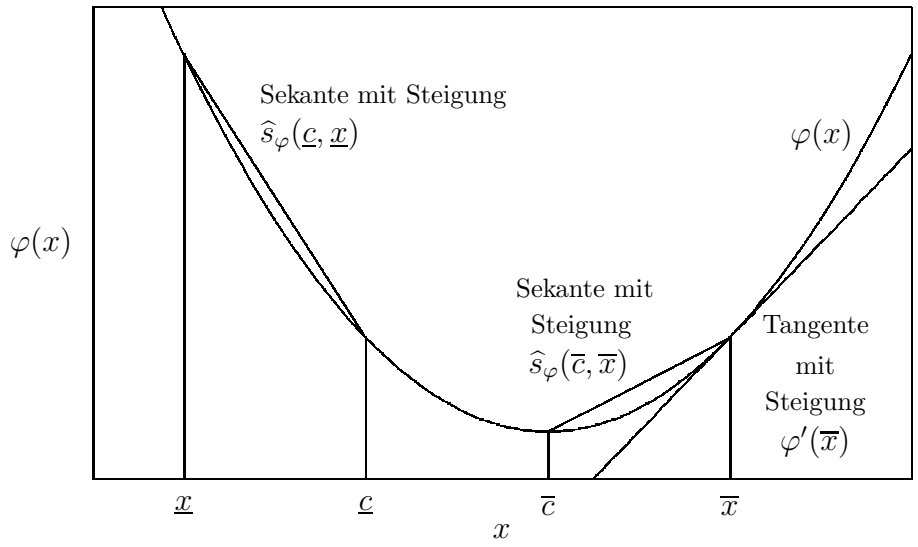

Abbildung 1: Darstellung der Größen aus Satz 2.2 und der Tangente von φ im Punkt $(\overline{x}, \varphi(\overline{x}))$ (konvexer Fall)

Beweis:
Sei φ auf X konvex. Zunächst wird wie in [18] bewiesen, daß die Funktion $s_\varphi(c,x)$ in x und c monoton wächst. Da φ konvex und stetig differenzierbar auf X ist, gilt die Ungleichung

$$\varphi(x_1) \geq \varphi(x_2) + (x_1 - x_2) \cdot \varphi'(x_2)$$

für alle $x_1, x_2 \in X$. Somit erhält man für $x \neq c$

$$\begin{aligned} \frac{\partial}{\partial x}(s_\varphi(c,x)) &= \frac{\partial}{\partial x}\left(\frac{\varphi(x)-\varphi(c)}{x-c}\right) \\ &= \frac{\varphi'(x)(x-c)-\varphi(x)+\varphi(c)}{(x-c)^2} \\ &\geq 0 \end{aligned}$$

und

$$\begin{aligned} \frac{\partial}{\partial c}(s_\varphi(c,x)) &= \frac{\partial}{\partial c}\left(\frac{\varphi(x)-\varphi(c)}{x-c}\right) \\ &= \frac{-\varphi'(c)(x-c)+\varphi(x)-\varphi(c)}{(x-c)^2} \\ &\geq 0. \end{aligned}$$

Daraus folgt (10) für $\underline{x} \neq \underline{c}$ und $\overline{x} \neq \overline{c}$. Hat man zum Beispiel $\underline{x} \neq \underline{c}$ und $\overline{x} = \overline{c}$, so ist die Ungleichung $s_\varphi(c,x) \leq \varphi'(\overline{x})$ erfüllt, da zum einen aufgrund der Konvexität von φ die Funktion φ' auf X monoton wachsend ist (vgl. auch Abb. 1) und zum anderen $s_\varphi(C,X) \subseteq \varphi'(X)$ gilt. Die verbleibenden Fälle verlaufen analog. Ebenso beweist man

den konkaven Fall (11). □

Der Satz ist von besonderer Bedeutung, da viele der Funktionen aus $\mathcal{F}$ auf Teilen des Definitionsbereichs konvex oder konkav und dort ebenfalls stetig differenzierbar sind. Eine Anwendung der beiden obigen Sätze wird am folgenden Beispiel demonstriert. Analog kann auch für andere Funktionen aus $\mathcal{F}$ vorgegangen werden.

Beispiel 2.1 *Betrachtet wird die Funktion* $\varphi(x) = \sinh(x)$*, die für* $x \geq 0$ *konvex und für* $x \leq 0$ *konkav ist. Es sei das Tripel* $F := (f_x, f_c, f_s)$ *gegeben. Gesucht sei das Steigungstripel* $H = (h_x, h_c, h_s) := \sinh(F)$*. Nach Satz 2.1 gilt*

$$\begin{aligned} h_x &= \sinh(f_x) \\ h_c &= \sinh(f_c) \\ h_s &= E_{\sinh}(f_c, f_x) \cdot f_s \supseteq s_{\sinh}(f_c, f_x) \cdot f_s \end{aligned}$$

für eine Einschließung $E_{\sinh}$ *von* $s_{\sinh}(f_c, f_x)$*, welche sich wegen* $\varphi'(x) = \cosh(x)$ $(x \in \mathbb{R})$ *und Satz 2.2 zum einen durch*

$$E_{\sinh} = \begin{cases} \left[\frac{\sinh(\underline{f_x})-\sinh(\underline{f_c})}{\underline{f_x}-\underline{f_c}}, \frac{\sinh(\overline{f_x})-\sinh(\overline{f_c})}{\overline{f_x}-\overline{f_c}}\right] & f_x \geq 0,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \left[\frac{\sinh(\overline{f_x})-\sinh(\overline{f_c})}{\overline{f_x}-\overline{f_c}}, \frac{\sinh(\underline{f_x})-\sinh(\underline{f_c})}{\underline{f_x}-\underline{f_c}}\right] & f_x \leq 0,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \cosh(f_x) & \text{sonst} \end{cases} \tag{12}$$

oder zum anderen mittels

$$E_{\sinh} = \begin{cases} \left[\frac{\sinh(\underline{f_x})-\sinh(\underline{f_c})}{\underline{f_x}-\underline{f_c}}, \frac{\sinh(\overline{f_x})-\sinh(\overline{f_c})}{\overline{f_x}-\overline{f_c}}\right] & f_x \geq 0,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \left[\cosh(\underline{f_x}), \frac{\sinh(\overline{f_x})-\sinh(\overline{f_c})}{\overline{f_x}-\overline{f_c}}\right] & f_x \geq 0,\ \underline{f_x} = \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \left[\frac{\sinh(\underline{f_x})-\sinh(\underline{f_c})}{\underline{f_x}-\underline{f_c}}, \cosh(\overline{f_x})\right] & f_x \geq 0,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} = \overline{f_c} \\ \left[\frac{\sinh(\overline{f_x})-\sinh(\overline{f_c})}{\overline{f_x}-\overline{f_c}}, \frac{\sinh(\underline{f_x})-\sinh(\underline{f_c})}{\underline{f_x}-\underline{f_c}}\right] & f_x \leq 0,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \left[\cosh(\overline{f_x}), \frac{\sinh(\underline{f_x})-\sinh(\underline{f_c})}{\underline{f_x}-\underline{f_c}}\right] & f_x \leq 0,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} = \overline{f_c} \\ \left[\frac{\sinh(\overline{f_x})-\sinh(\overline{f_c})}{\overline{f_x}-\overline{f_c}}, \cosh(\underline{f_x})\right] & f_x \leq 0,\ \underline{f_x} = \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \cosh(f_x) & \text{sonst} \end{cases} \tag{13}$$

bestimmen läßt. Zwar werden bei Anwendung von (13) bessere Einschließungen als bei Benutzung von (12) geliefert, jedoch ist damit ein höherer Aufwand verbunden. Bereits (12) bewirkt im allgemeinen eine Verbesserung gegenüber einer Auswertung gemäß $E_\varphi = \varphi'(f_x)$*.*

Für die Quadrat- und Wurzelfunktion rechnet man Einschließungen der Intervallsteigungen direkt aus. So erhält man zum Beispiel für $\varphi(x) = x^2$ mit $f(x) \in f_x$, $f(c) \in f_c$ und $f(x) \neq f(c)$:

$$s_\varphi(f(c), f(x)) = \frac{\varphi(f(x)) - \varphi(f(c))}{f(x) - f(c)} = \frac{(f(x))^2 - (f(c))^2}{f(x) - f(c)} = f(x) + f(c) \in f_x + f_c.$$

Am Ende dieses Abschnitts wird die Bestimmung der Einschließungen von Steigungen nichtdifferenzierbarer Funktionen aus $\mathcal{F}_e$ vorgestellt.

Satz 2.3 *Gegeben seien die Funktionen $f, u, v : D \subseteq \mathbb{R} \to \mathbb{R}$ mit den zugehörigen Steigungstripeln $F = (f_x, f_c, f_s)$, $U = (u_x, u_c, u_s)$ und $V = (v_x, v_c, v_s)$. Dann kann das Steigungstripel $H = (h_x, h_c, h_s) = \varphi(F)$ der Funktion $\varphi(f) = |f|$ durch*

$$\left.\begin{array}{ll} h_x &= |f_x| \\ h_c &= |f_c| \\ h_s &= E_\varphi \cdot f_s \end{array}\right\} \tag{14}$$

mit

$$E_\varphi = \begin{cases} [1,1] & \underline{f_x} \geq 0 \\ [-1,-1] & \overline{f_x} \leq 0 \\ \left[\frac{|f_x|-|f_c|}{\underline{f_x}-\underline{f_c}}, \frac{|\overline{f_x}|-|\overline{f_c}|}{\overline{f_x}-\overline{f_c}}\right] & 0 \in f_x,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \left[-1, \frac{|\overline{f_x}|-|\overline{f_c}|}{\overline{f_x}-\overline{f_c}}\right] & 0 \in f_x,\ \underline{f_x} = \underline{f_c} \text{ und } \overline{f_x} \neq \overline{f_c} \\ \left[\frac{|f_x|-|f_c|}{\underline{f_x}-\underline{f_c}}, 1\right] & 0 \in f_x,\ \underline{f_x} \neq \underline{f_c} \text{ und } \overline{f_x} = \overline{f_c} \\ [-1,1] & \text{sonst} \end{cases} \tag{15}$$

bestimmt werden. Ferner ist das Tripel $H = (h_x, h_c, h_s) = \varphi(F, U, V)$ ein Steigungstripel der Funktion $\varphi(f, u, v) = \chi(f, u, v)$ (χ sei in den Verzweigungspunkten stetig), wobei H mittels

$$\left.\begin{array}{ll} h_x &= \chi(f_x, u_x, v_x) \\ h_c &= \chi(f_c, u_c, v_c) \\ h_s &= \left\{\begin{array}{ll} u_s & f_x < 0 \\ v_s & f_x > 0 \\ u_s \mathbin{\underline{\cup}} v_s & \text{sonst} \end{array}\right. \end{array}\right\} \tag{16}$$

berechnet wird. Das Tripel $H = (h_x, h_c, h_s) = \varphi(U, V)$ ist für die Funktion $\varphi(u, v) = \max(u, v)$ ein Steigungstripel, wenn man H durch

$$\left.\begin{array}{ll} h_x &= \max(u_x, v_x) \\ h_c &= \max(u_c, v_c) \\ h_s &= \left\{\begin{array}{ll} u_s & v_x < u_x \\ v_s & v_x > u_x \\ u_s \mathbin{\underline{\cup}} v_s & \text{sonst} \end{array}\right. \end{array}\right\} \tag{17}$$

ermittelt.

Beweis: Siehe [18]. □

Bemerkung 2.3 *Sind die Funktionen u und v wie im obigen Satz gegeben, so läßt sich für die Funktion $h(x) = \min(u(x), v(x))$ aufgrund der Identität*

$$\min(u(x), v(x)) = -\max(-u(x), -v(x)), \quad x \in D$$

das zugehörige Steigungstripel H durch $H = -\max(-U, -V)$ berechnen.

Die Bestimmung des Steigungstripels einer faktorisierbaren Funktion f bezüglich der Intervalle X und C mit Hilfe der obigen Formeln wird im folgenden Algorithmus zusammengefaßt.

Algorithmus 1: Einfache Steigungsarithmetik
Voraussetzung: $f : D \subseteq \mathbb{R} \to \mathbb{R}$ im Sinne von Def. 1.6 faktorisierbar.

Gegeben: Codeliste $f_1, f_2, \ldots, f_k$ mit
1. $f_1 = x$ (Zuweisung der Unabhängigen) und o.B.d.A.
2. $f_i = \lambda_i$, $\lambda_i \in \mathbb{R}$, $i \in \{2, \ldots, m+1\}$ (Zuweisung der Konstanten),
Intervalle C, X mit $C \subseteq X \subseteq D$.

Gesucht: Steigungstripel (f_x, f_c, f_s).

INIT: Initialisiere Steigungstripel für Unabhängige x und Konstanten λ_i durch
$((f_1)_x, (f_1)_c, (f_1)_s) := (X, C, 1)$ und
$((f_i)_x, (f_i)_c, (f_i)_s) := (\lambda_i, \lambda_i, 0)$, $i \in \{2, \ldots, m+1\}$.

PROZEDUR: FOR $j := m+2$ TO k BEGIN

falls $f_j = f_l \circ f_r$ mit $\circ \in \mathcal{OP}$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (5), (7) und
(8) aus $((f_l)_x, (f_l)_c, (f_l)_s)$ und $((f_r)_x, (f_r)_c, (f_r)_s)$,

falls $f_j = \varphi(f_l)$ mit $\varphi \in \mathcal{F}$:
$((f_j)_x, (f_j)_c, (f_j)_s) = \varphi((f_l)_x, (f_l)_c, (f_l)_s)$
gemäß (6),

falls $f_j = |f_l|$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (14) und (15)
aus $((f_l)_x, (f_l)_c, (f_l)_s)$,

falls $f_j = \chi(f_b, f_l, f_r)$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (16)
aus $((f_b)_x, (f_b)_c, (f_b)_s)$, $((f_l)_x, (f_l)_c, (f_l)_s)$ und
$((f_r)_x, (f_r)_c, (f_r)_s)$,

falls $f_j = \max(f_l, f_r)$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (17)
aus $((f_l)_x, (f_l)_c, (f_l)_s)$ und $((f_r)_x, (f_r)_c, (f_r)_s)$,

falls $f_j = \min(f_l, f_r)$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß Bem. 2.3
aus $((f_l)_x, (f_l)_c, (f_l)_s)$ und $((f_r)_x, (f_r)_c, (f_r)_s)$

END

AUSGABE: $(f_x, f_c, f_s) := ((f_k)_x, (f_k)_c, (f_k)_s)$.

Implementierung

Bei der Implementierung von Algorithmus 1 wird wieder das Konzept der Operatorenüberladung ausgenutzt. Dazu ist zunächst eine Klasse `slope_type` zu definieren, deren Objekte Steigungstripel darstellen. Ein kurzer Ausschnitt des Headerfiles `slope.h`, das im Abschnitt A (S. 27) ausführlich aufgelistet ist, zeigt die wichtigsten Komponenten und Funktionen der Klasse:

```
class slope_type{

  private:
    interval f_x; //f(X)
    interval f_c; //f(C)
    interval f_s; //zugehoerige Intervallsteigung

    static int calc_slope_type;

  public:

    //Konstruktoren u. Destruktor
    ...
    //Initialiserungsfunktionen
    ...
    //Zugriffs- und Ausgabefunktionen
    ...
    //Ueberladen der Operatoren
    friend slope_type operator-(slope_type& );

    friend slope_type operator-(slope_type& , slope_type& );
    friend slope_type operator+(slope_type& , slope_type& );
    friend slope_type operator*(slope_type& , slope_type& );
    friend slope_type operator/(slope_type& , slope_type& );

    friend slope_type operator-(slope_type& , interval& );
    ...
    //Ueberladen der Standardfunktionen
    friend slope_type sqr(slope_type&);
    ...
    friend slope_type abs(slope_type&);
    friend slope_type max(slope_type&, slope_type&);
    ...
};
```

Für zwei Objekte der Klasse `slope_type` werden die elementaren Operationen so überladen, daß für das Steigungstripel des Ergebnisobjekts die zugehörige Gleichung in (5) (S. 5) gilt. Für die Multiplikation geschieht dies zum Beispiel durch den Operator

```
slope_type operator*(slope_type& s, slope_type& t)
```

dessen Quellcode im Abschnitt A (S. 29) abgedruckt ist.
Für Funktionen $\varphi \in \mathcal{F}$ und einer Variablen `t` vom Typ `slope_type` ist φ(`t`) gemäß (6) (S. 5) implementiert. Dabei werden Einschließungen von s_φ(`t.f_c`, `t.f_x`) mit Hilfe von Satz 10 berechnet. Wie in Beispiel 2.1 wurden für jedes $\varphi \in \mathcal{F}$ notwendige Fallunterscheidungen gemäß (12) bzw. (13) durchgeführt. Nur für die trigonometrischen Funktionen wird die Ableitung φ'(`t.f_x`) als Einschließung von s_φ(`t.f_c`, `t.f_x`)

verwendet. Es ist zu beachten, daß untere Schranken der Größen

$$\widehat{s}_\varphi(\texttt{Inf(t.f\_c)},\ \texttt{Inf(t.f\_x)}),\ \widehat{s}_\varphi(\texttt{Sup(t.f\_c)},\ \texttt{Sup(t.f\_x)})$$

sowie obere Schranken der Werte

$$\widehat{s}_\varphi(\texttt{Inf(t.f\_c)},\ \texttt{Inf(t.f\_x)})\ \text{ und }\ \widehat{s}_\varphi(\texttt{Sup(t.f\_c)},\ \texttt{Sup(t.f\_x)})$$

berechnet werden müssen. Dies kann am einfachsten durch intervallmäßige Auswertung geschehen und wird durch die Hilfsfunktionen

```
int s_1_down(interval&, interval&, real&, interval(*f)(interval& ))
int s_2_down(interval&, interval&, real&, interval(*f)(interval& ))

int s_1_up(interval&, interval&, real&, interval(*f)(interval& ))
int s_2_up(interval&, interval&, real&, interval(*f)(interval& ))
```

realisiert (vgl. dazu Abschnitte D und D, S. 39f). Beim Überladen der verbleibenden nichtdifferenzierbaren Funktionen verknüpft man die Objektkomponenten gemäß (14) und (15), (16) bzw. (17). Als Beispiele für das Überladen der Funktionen aus $\mathcal{F}'$ sind

```
slope_type exp(slope_type& s)
```

und

```
slope_type chi(slope_type& s, slope_type& p, slope_type& r)
```

ebenfalls im Abschnitt A (S. 29) zu finden.

Der mehrdimensionale Fall

Der mehrdimensionale Fall unterscheidet sich in der Vorgehensweise zur Berechnung der Steigungstripel nicht wesentlich vom eindimensionalen Fall. Zunächst muß geklärt werden, was man hier unter einem Steigungstripel versteht.

Definition 2.2 *Sei $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ und $\mathbb{IR}^n \ni C, X \subseteq D$ gegeben. Ein Tripel $(f_x, f_c, f_s) = (f_x, f_c, (f_s^i)_{1\le i\le n}) \in (\mathbb{IR} \times \mathbb{IR} \times \mathbb{IR}^n)$ heißt Steigungstripel bezüglich f, C und X, falls gilt:*

$$\begin{aligned} \forall x \in X &: f(x) \in f_x \\ \forall c \in C &: f(c) \in f_c \\ \forall x \in X\ \forall c \in C\ \exists s \in f_s &: f(x) = f(c) + s \cdot (x - c). \end{aligned}$$

Die Komponente $(f_s^i)_{1\le i\le n}$ ist dabei als Zeilenvektor zu interpretieren.

Bemerkung 2.4 *Auch hier wird wieder o.B.d.A. $C_i \subseteq X_i$ $(1 \le i \le n)$ vorausgesetzt.*

Die Art und Weise, wie solche Tripel für elementare Operationen und Funktionen verknüpft werden, beantwortet

Satz 2.4 *Für die Funktionen $h_1, h_2 : \mathbb{R}^n \to \mathbb{R}$ mit $h_1(x) \equiv \lambda$ ($\lambda \in \mathbb{R}$) und $h_2(x) = x_i$ ($1 \le i \le n$) seien die Tripel $(\lambda, \lambda, 0_{IR^n})$ sowie (X_i, C_i, e_i) die zu h_1 bzw. h_2 gehörigen Steigungstripel bezüglich der Intervallvektoren C, X. Seien weiterhin Funktionen $f, g : D \subseteq \mathbb{R}^n \to \mathbb{R}$ sowie deren Steigungstripel (f_x, f_c, f_s) und (g_x, g_c, g_s) bezüglich der Intervallvektoren C, X gegeben. Dann gilt für das Steigungstripel (h_x, h_c, h_s) der Funktion $h = f \circ g$, wobei $\circ \in \mathcal{OP}$:*

$$\left.\begin{array}{ll} h_x = f_x \circ g_x\,,\, h_c = f_c \circ g_c & \circ \in \{-,+,\cdot,/\} \\ h_s = f_s \circ g_s & \circ \in \{-,+\} \\ h_s = g_x \cdot f_s + f_c \cdot g_s & \circ = \cdot \\ h_s = (f_s - h_c \cdot g_s)/g_x & \circ = / \text{ und } 0 \notin g_x. \end{array}\right\} \tag{18}$$

Für $h = \varphi(f)$ gilt, falls $\varphi(f_x)$ existiert und $E_\varphi(f_c, f_x)$ eine Einschließung von $s_\varphi(f_c, f_x)$ für $\varphi \in \mathcal{F}$ darstellt,

$$\begin{array}{l} h_x = \varphi(f_x)\,,\, h_c = \varphi(f_c) \quad \text{und} \\ h_s = E_\varphi(f_c, f_x) \cdot f_s. \end{array} \tag{19}$$

Beweis: [18]. □

Bemerkung 2.5 *Wie im eindimensionalen Fall kann man zur Bestimmung einer Einschließung von $s_\varphi(f_c, f_x)$ Satz 2.2 heranziehen.*

Gemäß [18] gelten (14)-(17) auch im Mehrdimensionalen, wenn die Operationen zur Berechnung von h_s komponentenweise interpretiert werden:

Satz 2.5 *Gegeben seien die Funktionen $f, u, v : D \subseteq \mathbb{R}^n \to \mathbb{R}$ mit zugehörigen Steigungstripeln $F = (f_x, f_c, f_s)$, $U = (u_x, u_c, u_s)$ und $V = (v_x, v_c, v_s)$. Dann kann das Steigungstripel $H = (h_x, h_c, h_s) = \varphi(F)$ der Funktion $\varphi(f) = |f|$ durch*

$$\left.\begin{array}{ll} h_x & = |f_x| \\ h_c & = |f_c| \\ h_s^i & = E_\varphi \cdot f_s^i, \quad 1 \le i \le n \end{array}\right\} \tag{20}$$

bestimmt werden. E_φ wird dabei analog zu (15) berechnet. Ferner ist das Tripel $H = (h_x, h_c, h_s) = \varphi(F, U, V)$ ein Steigungstripel der Funktion $\varphi(f, u, v) = \chi(f, u, v)$ (χ in den Verzweigungspunkten stetig), welches mit Hilfe von

$$\left.\begin{array}{ll} h_x & = \chi(f_x, u_x, v_x) \\ h_c & = \chi(f_c, u_c, v_c) \\ h_s^i & = \left\{\begin{array}{ll} u_s^i & f_x < 0 \\ v_s^i & f_x > 0 \\ u_s^i \mathbin{\underline{\cup}} v_s^i & \text{sonst} \end{array}\right. \end{array}\right\} \tag{21}$$

für $1 \le i \le n$ ermittelt wird. Das Tripel $H = (h_x, h_c, h_s) = \varphi(U, V)$ ist für die Funktion $\varphi(u, v) = \max(u, v)$ ein Steigungstripel, wenn man H mittels

$$\left.\begin{array}{ll} h_x & = \max(u_x, v_x) \\ h_c & = \max(u_c, v_c) \\ h_s^i & = \left\{\begin{array}{ll} u_s^i & v_x < u_x \\ v_s^i & v_x > u_x \\ u_s^i \mathbin{\underline{\cup}} v_s^i & \text{sonst} \end{array}\right. \end{array}\right\} \tag{22}$$

für $1 \le i \le n$ berechnet.

Allgemein kann die Bestimmung des Steigungstripels einer faktorisierbaren Funktion f bezüglich der Intervallvektoren X und C durch folgenden Algorithmus zusammengefaßt werden:

Algorithmus 2: Eine mehrdimensionale Steigungsarithmetik (Vorwärtmethode)

Voraussetzung: $f: D \subseteq \mathbb{R}^n \to \mathbb{R}$ im Sinne von Def. 1.6 faktorisierbar.

Gegeben: Codeliste $f_1, f_2, \ldots, f_k$ mit
1. $f_i = x_i$, $i \in \{1, \ldots, n\}$ (Zuweisung der Unabhängigen) und o.B.d.A.
2. $f_i = \lambda_i$, $\lambda_i \in \mathbb{R}$, $i \in \{n+1, \ldots, n+m\}$ (Zuweisung der Konstanten),
Intervallvektoren C, X mit $C_i \subseteq X_i \subseteq D_i$ $(1 \leq i \leq n)$.

Gesucht: Steigungstripel (f_x, f_c, f_s).

INIT: Initialisiere Steigungstripel für die Unabhängigen x_i und Konstanten λ_i durch
$((f_i)_x, (f_i)_c, (f_i)_s) := (X_i, C_i, e_i)$, $i \in \{1, \ldots, n\}$ und
$((f_i)_x, (f_i)_c, (f_i)_s) := (\lambda_i, \lambda_i, 0_{IR^n})$, $i \in \{n+1, \ldots, n+m\}$.

PROZEDUR: FOR $j := n + m + 1$ TO k BEGIN

falls $f_j = f_l \circ f_r$ mit $\circ \in \mathcal{OP}$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (18) aus
$((f_l)_x, (f_l)_c, (f_l)_s)$ und $((f_r)_x, (f_r)_c, (f_r)_s)$,

falls $f_j = \varphi(f_l)$ mit $\varphi \in \mathcal{F}$:
$((f_j)_x, (f_j)_c, (f_j)_s) := \varphi((f_l)_x, (f_l)_c, (f_l)_s)$
gemäß (19),

falls $f_j = |f_l|$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (20) und (15)
aus $((f_l)_x, (f_l)_c, (f_l)_s)$,

falls $f_j = \chi(f_b, f_l, f_r)$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (21)
aus $((f_b)_x, (f_b)_c, (f_b)_s)$, $((f_l)_x, (f_l)_c, (f_l)_s)$ und
$((f_r)_x, (f_r)_c, (f_r)_s)$,

falls $f_j = \max(f_l, f_r)$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß (22)
aus $((f_l)_x, (f_l)_c, (f_l)_s)$ und $((f_r)_x, (f_r)_c, (f_r)_s)$,

falls $f_j = \min(f_l, f_r)$:
berechne $((f_j)_x, (f_j)_c, (f_j)_s)$ gemäß Bem. 2.3
aus $((f_l)_x, (f_l)_c, (f_l)_s)$ und $((f_r)_x, (f_r)_c, (f_r)_s)$

END

AUSGABE: $(f_x, f_c, f_s) := ((f_k)_x, (f_k)_c, (f_k)_s)$.

Implementierung

Die Formeln zur Berechnung der Steigungstripel bei Verknüpfung durch Elementaroperationen und elementaren Funktionen unterscheiden sich im Vergleich zum eindimensionalen Fall nur dadurch, das sie komponentenweise ausgeführt werden. Deshalb

liegt eine analoge Vorgehensweise zur Implementierung aus Abschnitt 2 nahe. Der einzige Unterschied besteht in der zugrundeliegenden Datenstruktur. Die dritte Intervallkomponente der Klasse `slope_type` wird durch einen n-dimensionalen Intervallvektor ersetzt:

```
class slope{
  private:
    int dim; //Dimension des Grundraums (entspricht n)
    interval f_x; //f(X)
    interval f_c; //f(C)
    ivector  f_s; //Intervallvektor mit Lb(f_s)=1 Ub(f_s)=dim

    static int calc_slope;

  public:
    //Konstruktoren u. Destruktor, Initialisierungsfunktionen...
    //Zugriffs- und Ausgabefunktionen...
    //Ueberladen der Operatoren und Funktionen...
};
```

Das Überladen der Operatoren und Elementarfunktionen erfolgt ähnlich zum eindimensionalen Fall. Für die Funktionen

```
slope operator*(slope& s, slope& t)

slope exp(slope& s)

slope chi(slope& s, slope& q, slope& r)
```

ist der Quelltext im Anhang (S. 33f) zu finden. Wie bei der Klasse `slope_type` kann man sich zur Berechnung der Einschließungen von $s_\varphi($`t.f_c`$,$ `t.f_x`$)$ eines Argumentobjekts `t` für $\varphi \in \mathcal{F}$ den dort vorgestellten Hilfsfunktionen bedienen.

Um in natürlicher Weise mit Objekten der Klasse `slope` rechnen zu können, wurde zusätzlich eine Klasse `slope_vector` implementiert.

3 Schnelle Berechnung von Intervallsteigungen

Verfahren

Wenn f durch Operationen aus $\mathcal{OP}$ und $\mathcal{F} \cup \{\text{abs}(.)\}$ zusammengesetzt ist, kann Korollar 3.1 in [4] angewendet werden. Denn für eine Codeliste f_1, $f_2, \ldots,$ f_k von f erfüllt der Vektor $((f_i)_s^j)_{1 \le i \le k}$ (j fest, $1 \le j \le n$) aufgrund von

$$(f_i)_s^j = \begin{cases} \delta_{ij} & 1 \le i \le n+m \\ \displaystyle\sum_{q=1}^{i-1} \alpha_{iq} \cdot (f_q)_s^j & n+m+1 \le i \le k \end{cases} \tag{23}$$

die Voraussetzungen des genannten Korollars. Die Größen α_{iq} sind entweder 0 oder entsprechen den zuvor berechneten Werten aller Teilfunktionen bzw. den Einschließungen $E_\varphi((f_q)_c, (f_q)_x)$ der Steigungen $s_\varphi((f_q)_c, (f_q)_x)$. Die Rückwärtsmethode kann

demnach zur schnellen Berechnung von Steigungen eingesetzt werden.

Aber auch die restlichen Verknüpfungen aus $\mathcal{F}_e$ können in eine Rückwärtsrechnung einbezogen werden. Grundlage dafür bildet

Lemma 3.1 *Seien $A \in \mathbb{IR}, B \in \mathbb{IR}$ gegeben. Dann gilt:*

$$A \underline{\cup} B \subseteq [0,1] \cdot A + [0,1] \cdot B =: C. \tag{24}$$

Beweis: Mit $a \in A$ gilt auch $a \in [1,1] \cdot A + [0,0] \cdot B$ und wegen der Inklusionsisotonie ist $a \in [0,1] \cdot A + [0,1] \cdot B$. Mit $b \in B$ gilt $b \in [0,0] \cdot A + [1,1] \cdot B$ und wegen der Inklusionsisotonie $b \in [0,1] \cdot A + [0,1] \cdot B$.
Insgesamt ergibt sich $A \cup B \subseteq [0,1] \cdot A + [0,1] \cdot B$. Da $C \in \mathbb{IR}$, ist auch die konvexe Hülle von $A \cup B$ in C enthalten. □

Sind die Voraussetzungen von Satz 2.5 erfüllt, so kann in (21) und (22) die Größe $u_s^i \underline{\cup} v_s^i$ durch $[0,1] \cdot u_s^i + [0,1] \cdot v_s^i$ ersetzt werden. Dann ist auch hier Korollar Korollar 3.1 in [4] anwendbar.

Im nun folgenden Algorithmus zur schnellen Berechnung von Intervallsteigungen werden alle Operationen ausführlich dargestellt.

Algorithmus 3: Mehrdimensionale Steigungsarithmetik (Rückwärtsrechnung)

Voraussetzung: $f: D \subseteq \mathbb{R}^n \to \mathbb{R}$ im Sinne von Def. 1.6 faktorisierbar.

Gegeben: Codeliste $f_1, f_2, \ldots, f_k$ mit
1. $f_i = x_i$, $i \in \{1, \ldots, n\}$ (Zuweisung der Unabhängigen) und o.B.d.A.
2. $f_i = \lambda_i$, $\lambda_i \in \mathbb{R}$, $i \in \{n+1, \ldots, n+m\}$ (Zuweisung der Konstanten),
Intervallvektoren C, X mit $C_i \subseteq X_i \subseteq D_i$ $(1 \le i \le n)$.

Gesucht: Steigungstripel (f_x, f_c, f_s).

SCHRITT 1: FOR $j := 1$ TO n: speichere $(f_j)_x := X_j$, $(f_j)_c := C_j$
FOR $j := n+1$ TO $n+m$: speichere $(f_j)_x := \lambda_j$, $(f_j)_c := \lambda_j$
FOR $j := n+m+1$ TO k: falls $f_j = \min(f_l, f_r)$:
setze $f_j = -\max(-f_l, -f_r)$,
berechne und speichere $(f_j)_x$ und $(f_j)_c$

SCHRITT 2: FOR $j := 1$ TO $k-1$: $u_j := 0$
$u_k := 1$

FOR $j := k$ DOWN TO $n+m+1$ BEGIN
falls $f_j = -f_l$: $u_l := u_l - u_j$,

falls $f_j = f_l \circ f_r$ mit $\circ \in \mathcal{OP}$
SWITCH($\circ$)
CASE + : $u_l := u_l + u_j$

$$
\begin{array}{lll}
 & & u_r := u_r + u_j \\
 & \text{CASE } - : & u_l := u_l + u_j \\
 & & u_r := u_r - u_j \\
 & \text{CASE } \cdot : & u_l := u_l + (f_r)_x \; u_j \\
 & & u_r := u_r + (f_l)_c \; u_j \\
 & \text{CASE } / : & u_l := u_l + u_j/(f_r)_x \\
 & & u_r := u_r - u_j \; (f_j)_c/(f_r)_x,
\end{array}
$$

falls $f_j = \varphi(f_l)$ mit $\varphi \in \mathcal{F} \cup \{\text{abs}(.)\}$
$u_l := u_l + u_j \; E_\varphi((f_l)_c, (f_l)_x)$,

falls $f_j = \max(f_l, f_r)$
falls $(f_l)_x > (f_r)_x$: $u_l := u_l + u_j$,
falls $(f_l)_x < (f_r)_x$: $u_r := u_r + u_j$
sonst: $\left\{ \begin{array}{l} u_l := u_l + [0,1]\, u_j \\ u_r := u_r + [0,1]\, u_j \end{array} \right.$,

falls $f_j = \chi(f_b, f_l, f_r)$
falls $(f_b)_x < 0$: $u_l := u_l + u_j$,
falls $(f_b)_x > 0$: $u_r := u_r + u_j$
sonst: $\left\{ \begin{array}{l} u_l := u_l + [0,1]\, u_j \\ u_r := u_r + [0,1]\, u_j \end{array} \right.$.

END

AUSGABE: $(f_x,\ f_c,\ f_s)$ mit $f_x := (f_k)_x$, $f_c := (f_k)_c$ und $f_s := (u_j)_{1 \le j \le n}$.

Implementierung

Grundsätzlich gleichen sich die Implementierungen zur schnellen Berechnung des Gradienten und der von Algorithmus 3. Schritt 1 entspricht auch hier dem Aufzeichnen eines Tapes. Im Gegensatz zur Implementierung der Klasse `rivall` wird hier kein Intervallvektor, sondern es werden zwei Komponenten des Typs `interval` mitgeführt, die den ersten beiden Bestandteilen des Steigungstripels entsprechen. Die so entstandene Klasse trägt den Namen `rslope`.

Schritt 1 ist mit verschiedenen Versionen der Funktion `forward_sweep` durchführbar (vgl. Abschnitt C, S. 35). Diese unterscheiden sich in der Art und Weise der Initialisierung der zweiten Komponente des Steigungstripels und in der Gestalt der zugrundeliegenden Funktion, für welche die Steigungen berechnet werden sollen. So können zum Beispiel auch Steigungsmatrizen für vektorwertige Funktionen bestimmt werden. Nachdem Schritt 1 durchgeführt worden ist, kann mit einer der Funktionen

```
void reverse_sweep(STAPE& wt, ivector& slopes, int y_c)

void reverse_sweep(STAPE& wt, imatrix& slopematrix)
```

Schritt 2 ausgeführt werden. Die Größe `y_c` gibt an, daß der Steigungsvektor der `y_c`-ten Komponentenfunktion einer vektorwertigen Funktion $f : \mathbb{R}^n \to \mathbb{R}^m$ bestimmt wird. Ist $m = 1$, so muß `y_c` auf 1 gesetzt werden.

4 Komponentenweise Berechnung

Verfahren

In [20] wird eine dritte Möglichkeit für die Berechnung von Steigungsvektoren dargestellt. In diesem Abschnitt wird ähnlich vorgegangen. Zunächst muß Definition 2.1 verallgemeinert werden (vgl. [20]):

Definition 4.1 *Seien $F : D \subseteq \mathbb{R} \to I\mathbb{R}$ und $C \subseteq X \subseteq D$ gegeben. Ein Tripel $(F_x, F_c, F_s) \in I\mathbb{R}^3$ heißt Steigungstripel bezüglich F, C und X, falls gilt:*

$$\begin{aligned}
&\forall x \in X : F(x) \subseteq F_x \\
&\forall c \in C : F(c) \subseteq F_c \\
&\forall x \in X\ \forall c \in C\ (x \neq c) \quad \forall y \in F(x)\ \forall \widetilde{y} \in F(c)\ \exists S \in F_s : y - \widetilde{y} = S \cdot (x - c).
\end{aligned}$$

Außerdem ist die Steigung $s(F, C, X)$ von F bezüglich X und C durch

$$s(F, C, X) := \left\{ \frac{y - \widetilde{y}}{x - c} \mid y \in F(X),\ \widetilde{y} \in F(C),\ x \in X,\ c \in C,\ x \neq c \right\}$$

definiert.

Bemerkung 4.1

1. *Es wird wieder o.B.d.A. $C \subseteq X$ vorausgesetzt.*

2. *Zur Berechnung eines Steigungstripels bei Verknüpfung durch $\circ \in \mathcal{OP}$ und $\varphi \in \mathcal{F} \cup \{\mathrm{abs}(.)\}$ können die Sätze 2.1, 2.2 und 2.3 benutzt werden. Es sei z.B. das Steigungstripel $F_T := (F_x, F_c, F_s)$ bezüglich $F : D \subseteq \mathbb{R} \to I\mathbb{R}$, C und X gegeben. Möchte man dann das Tripel $W_T := (W_x, W_c, W_s)$ bezüglich $W : D \subseteq \mathbb{R} \to I\mathbb{R}$, C und X für $W_T = \varphi(F_T)$ bestimmen, so kann man wie folgt vorgehen:*
   Seien $x \in X$ und $c \in C$ $(x \neq c)$. Zu $\widetilde{z} \in W(c)$ und $z \in W(x)$ gibt es $\widetilde{y} \in F(c)$ und $y \in F(x)$, sodaß gilt:

$$\begin{aligned}
z - \widetilde{z} = \varphi(y) - \varphi(\widetilde{y}) &= s_\varphi(\widetilde{y}, y) \cdot (y - \widetilde{y}) \\
&= s_\varphi(\widetilde{y}, y) \cdot S \cdot (x - c) \text{ für } S \in F_s \\
&= \underbrace{s_\varphi(\widetilde{y}, y) \cdot S}_{\in s_\varphi(F_c, F_s) \cdot F_s} \cdot (x - c)
\end{aligned}$$

   Ist $E_\varphi(F_c, F_s)$ eine Einschließung von $s_\varphi(F_c, F_s)$, so erhält man mittels $W_T := (\varphi(F_x), \varphi(F_c), E_\varphi(F_c, F_s) \cdot F_s)$ ein Steigungstripel der Funktion W.

Seien nun für stetiges $f : D = (D_1 \times \ldots \times D_n) \subseteq \mathbb{R}^n \to \mathbb{R}$ und $X, C \in \mathbb{IR}^n$ die Funktion $F^j : D_j \subseteq \mathbb{R} \to \mathbb{IR}$ durch

$$F^j(z) := f(X_1, \ldots, X_{j-1}, z, C_{j+1}, \ldots, C_n) \text{ für } 1 \leq j \leq n \tag{25}$$

definiert. Mit einer Einschließung $S(F^j,\ C_j,\ X_j)$ von $s(F^j,\ C_j,\ X_j)$ gilt für $x \in X$, $c \in C$, $1 \leq j \leq n$ (vgl. [20]):

$$f(x_1, \ldots, x_j, c_{j+1}, \ldots, c_n) \subseteq f(c) + \sum_{i=1}^{j} S(F^j,\ C_j,\ X_j) \cdot (X_j - C_j) \tag{26}$$

und speziell

$$f(x) \subseteq f(c) + \sum_{i=1}^{n} S(F^j,\ C_j,\ X_j) \cdot (X_j - C_j).$$

Im Gegensatz zur im Abschnitt 2 vorgestellten Arithmetik müssen zusätzlich zum Steigungsvektor alle Werte der Teilfunktionen $f(X_1, \ldots, X_j, C_{j+1}, \ldots, C_n)$ $(0 \leq j \leq n)$ in einem $(n+1)$-dimensionalen Intervallvektor mitgeführt werden. Durch (9) und (26) können diese Funktionswerte verbessert werden.

Die folgenden Größen werden im zusammenfassenden Algorithmus benötigt: für $f : \mathbb{R}^n \to \mathbb{R}$ und $X, C \in \mathbb{IR}^n$ sei r der $(n+1)$-dimensionale Intervallvektor, dessen Komponenten die Werte der Teilfunktionen $F^j(X_j)$ umfassen, also $r_j \supseteq f(X_1, \ldots, X_j, C_{j+1}, \ldots, C_n)$ für $j \in \{0, \ldots, n\}$ gilt[2]. Mit s sei der n-dimensionale Intervallvektor bezeichnet, für den $s(F^j,\ C_j,\ X_j) \subseteq s_j$ ist.

Algorithmus 4: Eine mehrdimensionale Steigungsarithmetik durch komponentenweise Berechnung

Voraussetzung: $f :\ D \subseteq \mathbb{R}^n \to \mathbb{R}$, im Sinne von Def. 1.6 faktorisierbar.

Gegeben: Codeliste $f_1,\ f_2, \ldots,\ f_k$ mit
1. $f_i = x_i\ \forall i \in \{1, \ldots, n\}$ (Zuweisung der Unabhängigen) und o.B.d.A.
2. $f_i = \lambda_i,\ \lambda_i \in \mathbb{R}\ \forall i \in \{n+1, \ldots, n+m\}$ (Zuweisung der Konstanten)
Intervallvektoren C, X mit $C_i \subseteq X_i \subseteq D_i$ $(1 \leq i \leq n)$ für welche die intervallmäßige Auswertung von $f_1, \ldots, f_k$ existiert.

Gesucht: Einschließungen für $f(X)$, $f(C)$ und Steigungsvektor S mit
$f(x) \subseteq f(c) + \sum_{i=1}^{n} S_j \cdot (X_j - C_j)\ \forall x \in X$ und $\forall c \in C$.

INIT: Initialisiere Vektoren r und s für Unabhängige x_i und Konstanten λ_i durch

$$\forall i \in \{1, \ldots, n\}\text{: } f_i.s := e_i \text{ und } f_i.r[j] := \begin{cases} C_i & \text{für } 0 \leq j < i \\ X_i & \text{für } i \leq j \leq n \end{cases}$$
$$\forall i \in \{n+1, \ldots, n+m\}\text{: } f_i.s := 0_{IR^n} \text{ und } f_i.r := (\lambda_i, \lambda_i, \ldots, \lambda_i).$$

[2]Beachte: Aus (25) ergibt sich $F^j(X_j) = F^{j+1}(C_{j+1})$. Einschließungen von $F^j(C_j)$ brauchen also nicht gesondert abgespeichert werden.

<u>*PROZEDUR:*</u> FOR $j := n+m+1$ TO k BEGIN

falls $f_j = f_l + f_r$:
$R := f_l.r[0] + f_r.r[0]$, $f_j.r[0] := R$
FOR $q := 1$ TO n BEGIN
$f_j.s[q] := f_l.s[q] + f_r.s[q]$
$R := R + f_j.s[q] \cdot (X_q - C_q)$
$f_j.r[q] := R \cap (f_l.r[q] + f_r.r[q])$
END

falls $f_j = f_l - f_r$:
$R := f_l.r[0] - f_r.r[0]$, $f_j.r[0] := R$
FOR $q := 1$ TO n BEGIN
$f_j.s[q] := f_l.s[q] - f_r.s[q]$
$R := R + f_j.s[q] \cdot (X_q - C_q)$
$f_j.r[q] := R \cap (f_l.r[q] - f_r.r[q])$
END

falls $f_j = f_l \cdot f_r$:
$R := f_l.r[0] \cdot f_r.r[0]$, $f_j.r[0] := R$
FOR $q := 1$ TO n BEGIN
$f_j.s[q] := (f_l.r[q] \cdot f_r.s[q] + f_r.r[q-1] \cdot f_l.s[q])$
$\cap\ (f_r.r[q] \cdot f_l.s[q] + f_l.r[q-1] \cdot f_r.s[q])$
$R := R + f_j.s[q] \cdot (X_q - C_q)$
$f_j.r[q] := R \cap (f_l.r[q] \cdot f_r.r[q])$
END

falls $f_j = f_l\ /\ f_r$ und $0 \notin f_r.r[q]\ \forall q \in \{0, \ldots, n\}$:
$R := f_l.r[0]/f_r.r[0]$, $f_j.r[0] := R$
FOR $q := 1$ TO n BEGIN
$f_j.s[q] := ((f_l.s[q] - f_j.r[q-1] \cdot f_r.s[q])/f_r.r[q])$
$\cap\ ((f_l.s[q] - f_l.r[q]/f_r.r[q] \cdot f_r.s[q])/f_r.r[q-1])$
$R := R + f_j.s[q] \cdot (X_q - C_q)$
$f_j.r[q] := R \cap (f_l.r[q]/f_r.r[q])$
END

falls $f_j = \varphi(f_l)$ mit $\varphi \in \mathcal{F} \cup \{\mathrm{abs}(.)\}$:
$R := \varphi(f_l.r[0])$, $f_j.r[0] := R$
FOR $q := 1$ TO n BEGIN
$f_j.s[q] := E_\varphi(f_l.r[q-1], f_l.r[q]) \cdot f_l.s[q]$
$R := R + f_j.s[q] \cdot (X_q - C_q)$
$f_j.r[q] := R \cap (\varphi(f_l.r[q]))$
END

END

<u>*AUSGABE:*</u> $f_k.r[n] \supseteq f(X)$, $f_k.r[0] \supseteq f(C)$ und $S := f_k.s$.

Implementierung

Es muß vorangestellt werden, daß in dieser Arbeit nur eine Implementierung für Testzwecke realisiert wurde. Dabei sind die elementaren Operationen, einige Standardfunktionen sowie die Betragsfunktion für Objekte einer Klasse `expansion` gemäß Algorithmus 4 überladen. Jedes solche Objekt besitzt zwei Intervallvektoren, welche die obigen Vektoren s und r repräsentieren. Wie in Algorithmus 4 zu sehen ist, werden ebenfalls die Argumentvektoren C und X beim Überladen der einzelnen Operationen und Funktionen benötigt. Das bedeutet, daß diese beiden Vektoren global zur Verfügung stehen müssen. Um die Flexibilität nicht zu beeinträchtigen (z. B. bei Berechnungen mit verschiedenen Argumentvektoren in einem Programmabschnitt), werden diese Vektoren zusätzlich als Komponenten eines jeden `expansion`-Objekts aufgenommen.

```
class expansion{

  private:
    int n;

    ivector x; //Argumentvektor X, sonst global
    ivector c; //Argumentvektor C, sonst global

    ivector f; //Funktionswertevektor dim=n+1
    ivector s; //Steigungsvektor      dim=n

  public:
    //Konstruktoren u. Destruktor
    ...
    //Initialisierungsfunktionen
    ...
    //Zugriffs- und Ausgabefunktionen
    ...
    //Ueberladen der Operatoren und Funktionen
    friend expansion operator+(expansion& g, expansion& h);
    friend expansion operator-(expansion& g, expansion& h);
    ...
    friend expansion sqr(expansion& g);
    ...
    friend expansion abs(expansion& g);
};
```

5 Vergleich der Verfahren

Zunächst werden die Implementierungen der Algorithmen 2, 3 und 4 hinsichtlich ihres Laufzeitverhaltens an der Funktion $f(x_1, \ldots, x_n) = \mathrm{e}^{x_1+\cdots+x_n}$ für verschiedene Dimensionen n verglichen. Dabei wurde $X_i = [-1, 1]$ und $C_i = [0, 0]$ $(1 \leq i \leq n)$ gesetzt. In Abbildung 2 (S. 21) ist zu erkennen, daß unter den beiden Vorwärtsberechnungen Algorithmus 2 besser als Algorithmus 10 abschneidet. Die Ursache liegt in der Vielzahl der komponentenweisen Zwischenberechnungen bei Algorithmus 4. Außerdem ist die Rückwärtsmethode deutlich überlegen. Vergleicht man die zentrierten Formen $z_f := f(C) + s \cdot (X - C)$ mit $s_f(C, X) \subseteq s$ für verschiedene n miteinander, so stellt

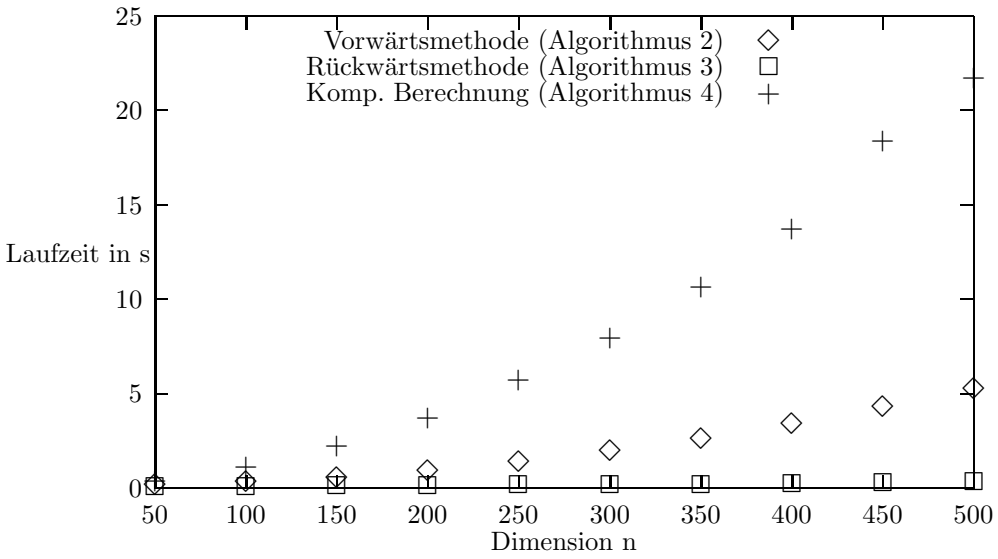

Abbildung 2: Verschiedene Laufzeiten der Algorithmen 2, 3 und 4 bei wachsender Dimension

man fest, daß durch komponentenweise Berechnung trotz schlechteren Laufzeitverhaltens keine besseren Einschließungen des Wertebereichs durch zentrierte Formen zu verzeichnen sind (vgl. Tabellen 1 und 2).

n	z_f
50	$[-5.184706E+021, 5.184706E+021]$
100	$[-2.688118E+043, 2.688118E+043]$
150	$[-1.393710E+065, 1.393710E+065]$
200	$[-7.225974E+086, 7.225974E+086]$
250	$[-3.746455E+108, 3.746455E+108]$
300	$[-1.942427E+130, 1.942427E+130]$

Tabelle 1: Zentrierte Form bei verschiedenen Dimensionen n der Funktion $f(x_1, \ldots, x_n) = \mathrm{e}^{x_1+\cdots+x_n}$, Berechnung der Steigungsvektoren mit Algorithmus 2

n	z_f
50	$[-5.184706E+021, 5.184706E+021]$
100	$[-2.688118E+043, 2.688118E+043]$
150	$[-1.393710E+065, 1.393710E+065]$
200	$[-7.225974E+086, 7.225974E+086]$
250	$[-3.746455E+108, 3.746455E+108]$
300	$[-1.942427E+130, 1.942427E+130]$

Tabelle 2: Zentrierte Form bei verschiedenen Dimensionen n der Funktion $f(x_1, \ldots, x_n) = \mathrm{e}^{x_1+\ldots+x_n}$, Berechnung der Steigungsvektoren mit Algorithmus 4

Für einige (zufällig gewählte) Funktionen wurden ebenfalls Steigungsvektoren und zugehörige zentrierte Formen berechnet. Dabei handelt es sich um die Funktionen

$$\begin{array}{lcll} f_1(x_1,x_2) & = & 100\cdot(x_1-x_2^2)^2+(x_1-1)^2 & \text{(entnommen aus [18])} \\ f_2(x_1,x_2) & = & 4x_1^2+x_1\cdot x_2+4x_2^2-x_1 & \\ f_3(x_1,x_2) & = & \mathrm{e}^{x_1\cdot x_2}-x_1 & \text{(entnommen aus [20])} \\ f_4(x_1,x_2) & = & x_1-\mathrm{e}^{x_2}+\sin^2(x_2) & \\ f_5(x_1,x_2) & = & |x_1-\mathrm{e}^{x_2}+\sin^2(x_2)| & \end{array}$$

mit den folgenden Argumentwerten:

$$\begin{array}{lllll} f_1, f_2: & X_1 := [4.0, 4.25] & X_2 := X_1 & C_1 := \mathrm{m}(X_1) & C_2 := \mathrm{m}(X_2) \\ f_3: & X_1 := [-1.0, 1.0] & X_2 := [1.0, 1.25] & C_1 := \mathrm{m}(X_1) & C_2 := \mathrm{m}(X_2) \\ f_4, f_5: & X_1 := [-0.25, 0.25] & X_2 := [1.0, 1.25] & C_1 := \mathrm{m}(X_1) & C_2 := \mathrm{m}(X_2) \end{array}$$

Die berechneten Steigungen bei Anwendung der Vorwärts- und Rückwärtsmethode sind (bei gewählter Ausgabe von sechs Nachkommastellen) gleich. Sie betragen:

```
f_1 Slopes: [2.002663E+004,2.257962E+004] [-2.695313E+003,-2.464062E+003]
zentrische Form: [1.346722E+004,1.978596E+004]

f_2 Slopes: [ 35.500000, 36.750000] [ 36.625000, 37.625000]
zentrische Form: [1.397187E+002,1.583125E+002]

f_3 Slopes: [ -1.000000,  5.389057] [ -0.000000,  0.000000]
zentrische Form:[ -4.389057,  6.389057]


f_4 Slopes: [  1.000000,  1.000000] [ -2.731170, -1.895244]
zentrische Form: [ -2.857527, -1.674733]

f_5 Slopes: [ -1.000000, -1.000000] [  1.895244,  2.731170]
zentrische Form: [  1.674733,  2.857527]
```

Berechnet man die Steigungen mittels Algorithmus 4, so unterscheiden sich zum Teil die auf diese Art und Weise berechneten Komponenten der Steigungsvektoren von den oben bestimmten Intervallsteigungen:

```
f_1 Slopes: [2.012819E+004,2.247493E+004] [-2.800000E+003,-2.362500E+003]
zentrische Form: [1.346722E+004,1.978596E+004]

f_2 Slopes: [ 35.625000, 36.625000] [ 36.500000, 37.750000]
zentrische Form: [1.397187E+002,1.583125E+002]

f_3 Slopes: [ -0.367880,  0.718282] [ -4.670775,  4.670775]
zentrische Form: [ -4.389057,  6.389057]

f_4 Slopes: [  1.000000,  1.000000] [ -2.731170, -1.895244]
zentrische Form: [ -2.857527, -1.674733]

f_5 Slopes: [ -1.000000, -1.000000]  [  1.895244,  2.731170]
zentrische Form: [  1.674733,  2.857527]
```

6 Beispiele: Verifikation von Nullstellen

Eine Möglichkeit zur Verifikation von Lösungen des Gleichungssystems (1)

$$\left.\begin{array}{rcl} x_1^2 - \mathrm{e}^{x_2} + |x_2| & = & 0 \\ x_1^2 - x_2^2 - |x_1| & = & 0 \end{array}\right\} (*)$$

bietet der folgende Satz aus [21]:

Satz 6.1 *Seien $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ stetig und eine zu f gehörige intervallwertige Funktion $F : D \to \mathbb{IR}^n$ mit $f(x) \in F(x)$ $(x \in D)$ gegeben. Weiterhin seien $\emptyset \neq X \subseteq D$ kompakt und konvex, $c \in D$ und $R \in \mathbb{R}^{n\times n}$. Falls dann*

$$c - R \cdot F(c) + \{I - R \cdot s_f(c, X)\} \cdot (X - c) \subseteq \mathrm{int}(X) \tag{27}$$

gilt[3]*, so existiert ein $\widehat{x} \in X$ mit $f(\widehat{x}) = 0$.*

Um eine Inklusion gemäß (27) zu erhalten, empfiehlt Rump in [21] die Größe c als Näherung einer Nullstelle $\widehat{x}$ von f und die Matrix R als approximative Inverse einer Matrix aus $s_f(c, X)$ zu wählen. X sollte eine möglichst gute Einschließung von c und $\widehat{x}$ sein.
(27) kann dazu verwendet werden, um eine mögliche Einschließung $X_0 \in \mathbb{IR}$ einer Nullstelle iterativ zu verbessern. Setzt man

$$X_{i+1} := c_i - R_i \cdot F(c_i) + \{I - R_i \cdot S_f(c_i, X_i)\} \cdot (X_i - c_i), \quad i \geq 0$$

und ist $X_{i+1} \subseteq \mathrm{int}(X_i)$ erfüllt, so befindet sich in X_i eine Nullstelle. Dabei ist $S_f(c_i, X_i) \in \mathbb{IR}^{n\times n}$ eine Einschließung der Steigung $s_f(c_i, X_i)$. R_i kann z.B. als Näherung der Inversen der Mittelspunktsmatrix von $S_f(c_i, X_i)$ und c_i als Mittelpunkt von X_i gewählt werden. Das Verfahren ist ohne Ergebnis abzubrechen, wenn $X_{i+1} \subseteq \mathrm{int}(X_i)$ nach endlich vielen Schritten (z.B. $i = 15$) nicht verifiziert werden kann. Dann ist mit der

[3]Ist nur $c - R \cdot F(c) + \{I - R \cdot s_f(c, X)\} \cdot (X - c) \subseteq X$ erfüllt, kann die Regularität von R nicht automatisch verifiziert werden.

vorgestellten Methode keine Existenzaussage möglich.

Eine solche Iteration wurde für das obige Gleichungssystem $(*)$ durchgeführt. Vermutet wird eine Nullstelle im Intervall[4] $X_0 = [-3.0, -2.9] \times [2.4, 2.5]$. Durch vier Iterationen konnte ein Intervall X_4 berechnet werden, welches eine Lösung von (1) enthält und für das $\mathrm{d}(X_4) \leq 1.e-14$ gilt. Dabei konnte $X_{i+1} \subseteq \mathrm{int}(X_i)$ in jedem Schritt verifiziert werden.

i	X_i	verif.
1	$[-2.978216, -2.966369] \times [2.416369, 2.426175]$	ja
2	$[-2.971545, -2.971414] \times [2.420319, 2.420429]$	ja
3	$[-2.971479, -2.971478] \times [2.420373, 2.420374]$	ja
4	$[-2.971479, -2.971478] \times [2.420373, 2.420374]$	ja

Tabelle 3: Einschließungen einer Lösung des nichtlinearen Gleichungssystems (1)

Ein zweites Beispiel geht auf Rump [21] zurück. Dort werden Einschließungen von Nullstellen einer vektorwertigen Funktion nachgewiesen, deren Komponentenfunktionen nirgends differenzierbare Anteile enthalten. Bei dieser Funktion handelt es sich um $f : \mathbb{R}^2 \to \mathbb{R}^2$, $f(x,y) = (f_1(x,y), f_2(x,y))^T$, die durch

$$\begin{aligned} f_1(x,y) &:= \mathrm{e}^{w(x)} - w(y) - 1 \\ f_2(x,y) &:= w(x)^2 + w(y)^2 - \cos(20x) \cdot \cos(20y)/\sqrt{2} \end{aligned}$$

definiert ist. Beide Komponenten enthalten eine stetige, aber nirgends differenzierbare

[4]Für die Berechnung auf der Maschine müssen die Zahlen -2.9 und 2.4 so gerundet werden, daß man eine Einschließung von X_0 auf der Machine erhält.

Funktion $w : \mathbb{R} \to \mathbb{R}$ mit

$$w(x) := \lim_{n\to\infty} w_n(x),\ w_n(x) := \sum_{v=1}^{n} \alpha^v \cdot \sin(b^v \cdot \pi x) \tag{28}$$
$$(0 < \alpha < 1,\ b \in \mathbb{N} \text{ und gerade},\ \alpha \cdot b > 1 + \frac{3}{2}\pi).$$

Zur Darstellung auf dem Rechner kann für festes m der Reihenrest aus (28) unter Verwendung der geometrischen Reihe wie folgt abgeschätzt werden (vgl. [21]):

$$\left| \sum_{v=m+1}^{\infty} \alpha^v \cdot \sin(b^v \cdot \pi x) \right| \leq \sum_{v=m+1}^{\infty} \alpha^v = \frac{\alpha^{m+1}}{1-\alpha}.$$

Somit erhält man eine Intervallfunktion $W(x)$

$$W(x) := \sum_{v=1}^{m} \alpha^v \cdot \sin(b^v \cdot \pi x) + \left[-\frac{\alpha^{m+1}}{1-\alpha},\ \frac{\alpha^{m+1}}{1-\alpha} \right],$$

für die $w(x) \in W(x)$ $(x \in D)$ gilt. Dadurch sind die Voraussetzungen von Satz 6.1 erfüllt, und Rump weist unter Anwendung von (27) für eine Reihe von Intervallen nach, daß diese Nullstellen enthalten. Für $m \in \{2, 6, 8, 10\}$, $b = 60$ und $\alpha = 0.1$ wurde versucht, die Rechnung für fünf dieser Bereiche X_i $(1 \leq i \leq 5)$ nachzuvollziehen. Im einzelnen handelt es sich um

$X_1 := 0.5487^{50009}_{49998} \times 0.2343841^{9}_{7}$ $\quad X_2 := 0.5021204500^{7}_{5} \times 0.2354628035^{9}_{7}$
$X_3 := 0.514782868610^{5}_{0} \times 0.2354432680^{9}_{6}$ $\quad X_4 := 0.535027699^{2}_{0} \times 0.235032401^{8}_{6}$
$X_5 := 0.549779186^{15}_{09} \times 0.249772912^{23}_{17}$.

Abbildung 3 (S. 26) zeigt weitere Bereiche, in denen Nullstellen vermutet werden. Mit Hilfe von Algorithmus 8 (Vorwärtsrechnung) konnte mit den oben erwähnten Parametern in keinem der Bereiche $X_1, \ldots, X_5$ die Existenz von Nullstellen nachgewiesen werden. Es treten für $m \in \{2, 8, 10\}$ erhebliche Abweichungen der Größe der Eingangsintervalle zur Größe der durch einmalige Anwendung von (27) erhaltenen Ergebnisintervalle auf[5]. Nur für $m = 6$ ist die Größe der Ausgangs- und Ergebnisintervalle vergleichbar. Die Ergebnisse findet man in den Tabellen 4-7 (S. 26f). Durch Berechnung der Steigungsmatrix mittels Algorithmus 10 konnten die zuvor berechneten Ergebnisse bestätigt werden. Da in [21] keine Angabe über die verwendeten Parameter m, b und α gemacht wird, ist davon auszugehen, daß dort andere als die hier verwendeten Parameter benutzt wurden.

[5] Bei der einmaligen Iteration wurden Einschließungen y des Defekts $\widehat{x} - c$ verwendet. Mit $y := X - c$ muß dann überprüft werden, ob $-R \cdot F(c) + \{I - R \cdot s_f(c, X)\} \cdot y =: y_{iter}$ in das Innere von y abgebildet wird.

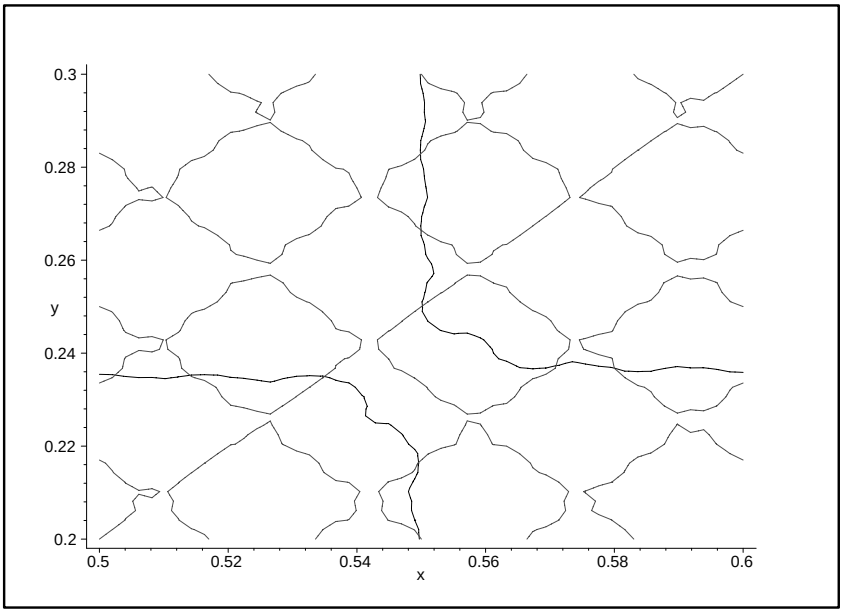

Abbildung 3: Nullstellen der Funktion f bei bei Verwendung von $w_2(x)$

	y	y_{iter}	verif.
X_1	[-5.500001E-009,5.500001E-009]	[-1.797003E-004,1.850084E-004]	n
	[-1.000001E-008,1.000001E-008]	[-1.930529E-005,1.943127E-005]	n
X_2	[-1.000012E-011,1.000012E-011]	[-3.300704E-005,3.627344E-005]	n
	[-1.000003E-011,1.000003E-011]	[-1.391951E-005,1.218944E-005]	n
X_3	[-2.500112E-012,2.500112E-012]	[-2.168140E-005,2.024335E-005]	n
	[-1.500023E-012,1.500051E-012]	[-2.204183E-005,1.181883E-005]	n
X_4	[-1.000012E-011,1.000012E-011]	[-1.785060E-005,1.706792E-005]	n
	[-1.000003E-011,1.000003E-011]	[-1.483952E-005,1.428322E-005]	n
X_5	[-3.000156E-012,3.000045E-012]	[-1.458588E-006,5.653497E-006]	n
	[-3.000045E-012,3.000017E-012]	[-1.476927E-005,3.104631E-005]	n

Tabelle 4: y, y_{iter} für $m = 2$

7 Zusammenfassung

Treten in den Komponenten nichtlinearer Gleichungssysteme nichtdifferenzierbare Funktionen auf, so kann man mit Hilfe von Intervallsteigungen mögliche Lösungen verifizieren. Die Berechnung der Intervallsteigungen kann automatisiert werden. In dieser Arbeit wurden dazu verschiedene Verfahren vorgestellt und in C-XSC implementiert. Der Quellcode der Implementierungen steht frei zur Verfügung.

Berechnungen mit Intervallsteigungen liefern im Vergleich zu entsprechenden Berechnungen mit intervallmäßiger automatischer Differentiation engere Einschließungen.

	y	y_{iter}	verif.
X_1	[-5.500001E-009,5.500001E-009]	[-5.771511E-007,5.734997E-007]	n
	[-1.000001E-008,1.000001E-008]	[-6.042639E-007,6.047535E-007]	n
X_2	[-1.000012E-011,1.000012E-011]	[-1.858792E-011,1.900936E-011]	n
	[-1.000003E-011,1.000003E-011]	[-3.007051E-011,2.405468E-011]	n
X_3	[-2.500112E-012,2.500112E-012]	[-1.668138E-012,1.982581E-012]	j
	[-1.500023E-012,1.500051E-012]	[-1.509517E-012,1.532737E-012]	n
X_4	[-1.000012E-011,1.000012E-011]	[-1.569289E-011,1.634213E-011]	n
	[-1.000003E-011,1.000003E-011]	[-1.607722E-011,1.711316E-011]	n
X_5	[-3.000156E-012,3.000045E-012]	[-3.003406E-012,3.181738E-012]	n
	[-3.000045E-012,3.000017E-012]	[-3.116019E-012,2.696676E-012]	n

Tabelle 5: y, y_{iter} für $m = 6$

	y	y_{iter}	verif.
X_1	[-5.500001E-009,5.500001E-009]	[-2.126241E-005,2.125879E-005]	n
	[-1.000001E-008,1.000001E-008]	[-2.233872E-005,2.233924E-005]	n
X_2	[-1.000012E-011,1.000012E-011]	[-1.221060E-009,1.221877E-009]	n
	[-1.000003E-011,1.000003E-011]	[-1.760383E-009,1.756312E-009]	n
X_3	[-2.500112E-012,2.500112E-012]	[-1.833713E-010,1.842100E-010]	n
	[-1.500023E-012,1.500051E-012]	[-1.537086E-010,1.525942E-010]	n
X_4	[-1.000012E-011,1.000012E-011]	[-1.249673E-009,1.252179E-009]	n
	[-1.000003E-011,1.000003E-011]	[-1.296284E-009,1.295802E-009]	n
X_5	[-3.000156E-012,3.000045E-012]	[-2.944332E-010,2.935739E-010]	n
	[-3.000045E-012,3.000017E-012]	[-2.767603E-010,2.758371E-010]	n

Tabelle 6: y, y_{iter} für $m = 8$

A Implementierung der Klasse slope_type

Diese Klasse wurde in Abschnitt 2 (S. 10f) beschrieben. Zu ihr gehören die Dateien

`slope.h` `slope.cpp` `helpfuncs.h` `helpfuncs.cpp`

Headerfile `slope.h`

```
#ifndef _SLOPE_H
#define _SLOPE_H

#include "helpfuncs.h"
/*------------------------------------------------------------------------
class slope_type  zur automatischen Bestimmung der Intervallsteigung einer
                  gegebenen Funktion
                  angelehnt an PASCAL-XSC Version von Dr. D. Ratz (slopes.p)

  Elemente:
            interval  f_x ............... 1.Intervalkomponente (Fkt.wert)
```

	y	y_{iter}	verif.
X_1	[-5.500001E-009,5.500001E-009]	[-7.659337E-004,7.659301E-004]	n
	[-1.000001E-008,1.000001E-008]	[-8.047813E-004,8.047818E-004]	n
X_2	[-1.000012E-011,1.000012E-011]	[-4.459435E-008,4.459517E-008]	n
	[-1.000003E-011,1.000003E-011]	[-6.419773E-008,6.419366E-008]	n
X_3	[-2.500112E-012,2.500112E-012]	[-6.792733E-009,6.793571E-009]	n
	[-1.500023E-012,1.500051E-012]	[-5.661240E-009,5.660126E-009]	n
X_4	[-1.000012E-011,1.000012E-011]	[-4.578529E-008,4.578779E-008]	n
	[-1.000003E-011,1.000003E-011]	[-4.743814E-008,4.743766E-008]	n
X_5	[-3.000156E-012,3.000045E-012]	[-1.082954E-008,1.082868E-008]	n
	[-3.000045E-012,3.000017E-012]	[-1.017745E-008,1.017652E-008]	n

Tabelle 7: y, y_{iter} für $m = 10$

```
            interval  f_c ............... 2.Intervalkomponente (Fkt.wert)
            interval  f_s ............... Steigung

            static int calc_slope_type ...falls 1 werden Steigungen  be-
                                          rechnet; default ist 1

  Elementfunktionen:

            siehe Implementation

Implementationsfile: slope.cpp
------------------------------------------------------------------------*/

class slope_type{

  private:
   interval f_x; //f(X)
   interval f_c; //f(C)
   interval f_s; //zugehoerige Steigungskomponente

   static int calc_slope_type;

  public:
   static void set_calc_slope_type();
   static void unset_calc_slope_type();
   static int get_calc_slope_type();

   slope_type();;
   slope_type(real& );
   slope_type(interval& );
   slope_type(interval& , interval&);
   slope_type(interval& , real&);

   friend slope_type var_slope_type(interval&, interval& );

   friend slope_type const_slope_type(real& );
   friend slope_type const_slope_type(interval& );
   friend slope_type const_slope_type(interval&, interval& );
   friend slope_type const_slope_type(interval&, real& );

   friend slope_type init_mid_var(interval& );
   friend slope_type init_mid_const(interval& );

   friend interval get_fx(slope_type& );
   friend interval get_fc(slope_type& );
```

```
   friend interval get_fs(slope_type& );

   friend void print_slope_type(slope_type& );

   friend slope_type operator-(slope_type& );

   friend slope_type operator-(slope_type& , slope_type& );
   friend slope_type operator+(slope_type& , slope_type& );
   friend slope_type operator*(slope_type& , slope_type& );
   friend slope_type operator/(slope_type& , slope_type& );

   friend slope_type operator-(slope_type& , interval& );
   //Es folgen weiter Operatoren

   friend slope_type sqr(slope_type&);
   friend slope_type sqrt(slope_type&);
   friend slope_type sqrt(slope_type&, int);

   friend slope_type power(slope_type&, int);
   //Es folgen weiter elementare Funktionen
};

void feval(slope_type (*f)(slope_type& ), interval& , interval& );
void fseval(slope_type (*f)(slope_type& ), interval&,
           interval& , interval& );
#endif
```

Auszüge aus `slope.cpp`

```
#include "slope.h"
...
/*------------------------------------------------------------------------

operator *

------------------------------------------------------------------------*/
slope_type operator*(slope_type& s, slope_type& t)
{
  slope_type res;
  res.f_x=s.f_x*t.f_x;
  if(slope_type::calc_slope_type)
  {
    res.f_c=s.f_c*t.f_c;
    res.f_s=s.f_s*t.f_x+s.f_c*t.f_s; //Symetrie ausnutzen
    res.f_s=res.f_s & (s.f_x*t.f_s+s.f_s*t.f_c); //Durchschnitt
   }
   return res;
}
...
/*------------------------------------------------------------------------

exp  konvex fuer alle x aus R, streng monoton wachsend

------------------------------------------------------------------------*/
slope_type exp(slope_type& s)
{
 slope_type res;
 res.f_x=exp(s.f_x);

 if(slope_type::calc_slope_type)
  {
   res.f_c=exp(s.f_c);

   real h_d, h_u;

   if(s_1_down(s.f_x, s.f_c, h_d, exp)
```

```
      &&
      s_2_up(s.f_x, s.f_c, h_u ,exp))
     {
       //konvex u. streng wachsend
       res.f_s=s.f_s*_interval(h_d,h_u);
       return res;
     }

   if( (!s_1_down(s.f_x, s.f_c, h_d, exp)) //==0
      &&
        s_2_up(s.f_x, s.f_c, h_u, exp) )
     {
       //konvex u. streng wachsend
       h_d=Inf(res.f_x);
       res.f_s=s.f_s*_interval(h_d,h_u);
       return res;
     }

   if( s_1_down(s.f_x, s.f_c, h_d, exp)
      &&
       (!s_2_up(s.f_x, s.f_c,h_u, exp)) ) //==0
     {
       //konvex u. streng wachsend
       h_u=Sup(res.f_x);
       res.f_s=s.f_s*_interval(h_d,h_u);
       return res;
     }

     res.f_s=s.f_s*res.f_x;
  }
 return res;
}
...
/*------------------------------------------------------------------------

chi : Verzweigungsfunktion

------------------------------------------------------------------------*/
slope_type chi(slope_type& s, slope_type& q, slope_type& r)
{
 slope_type res;
 res.f_x=chi(s.f_x, q.f_x, r.f_x);
 if(slope_type::calc_slope_type)
  {
    res.f_c=chi(s.f_c, q.f_c, r.f_c);
    res.f_s=chi(s.f_x, q.f_s, r.f_s);
  }
 return res;
}
```

B Implementierung der Klasse slope

Diese Klasse wurde in Abschnitt 2 (S. 13f) beschrieben. Zu ihr gehören die Dateien

`multi_dim_slope.h` `multi_dim_slope.cpp` `helpfuncs.h` `helpfuncs.cpp`

Headerfile `multi_dim_slope.h`

```
////////////////////////////////////////////////////////////////////////
//
// Headerfile multi_dim_slope.h fuer mehrdimensionale
// Intervallsteigungsarithmetik
```

```
//
//////////////////////////////////////////////////////////////////////////

#ifndef _MULTIDIMSLOPE_H
#define _MULTIDIMSLOPE_H

//cxsc headers
#include "interval.hpp"
#include "imath.hpp"
#include "ivector.hpp"

//C++ standard headers
#include <iostream.h>
#include <stdlib.h>

//einbinden der globalen Hilfsfunktionen
#include "helpfuncs.h"

//debug flag
#define debug

/*------------------------------------------------------------------------
class slope  automatische Berechnung mehrdimensionaler Intervallsteigungen
             angelehnt an PASCAL-XSC Version von Dr. D. Ratz (slp_ari.p)

  Elemente:
            int dim ..................... Dimension der Unabhaeng.
            interval  f_x ............... 1.Intervalkomponente (Fkt.wert)
            interval  f_c ............... 2.Intervalkomponente (Fkt.wert)
            ivector   f_s ............... Steigungvektor

            static int calc_slope ....... falls 1 werden Steigungen  be-
                                          rechnet; default ist 1

  Elementfunktionen:

            siehe Implementation

Implementationsfile: multi_dim_slope.cpp
------------------------------------------------------------------------*/

class slope{

 private:
  int dim;
  interval f_x; //f(X)
  interval f_c; //f(C)
  ivector  f_s; //Intervallvektor mit Lb(f_s)=1 Ub(f_s)=dim

  static int calc_slope;

 public:

  static void set_calc_slope();
  static void unset_calc_slope();
  static int get_calc_slope();

  slope();
  slope(int );
  slope(slope& );

  slope& operator=(slope&);

  friend slope var_slope(int, int, real& );
  friend slope var_slope(int, int, interval& );
  friend slope var_slope(int, int, interval&, interval& );
  friend slope var_slope(int, int, interval&, real& );
```

```
  friend slope const_slope(int, real& );
  friend slope const_slope(int, interval& );

  friend slope init_mid_var(int, int, interval& );

  friend void print_slope(slope&);

  friend int get_dim(slope& );
  friend interval get_fx(slope& );
  friend interval get_fc(slope& );
  friend ivector get_fs(slope& );

  friend slope operator-(slope& );

  friend slope operator-(slope& ,slope& );
  friend slope operator+(slope& ,slope& );
  friend slope operator*(slope& ,slope& );
  friend slope operator/(slope& ,slope& );

  friend slope operator-(slope& ,interval& );
  Es folgen weiter Operatoren...

  friend slope sqr(slope&);
  friend slope sqrt(slope&);
  friend slope sqrt(slope&, int);

  friend slope power(slope&, int);
  //Es folgen weiter Funktionen...
};

/*------------------------------------------------------------------------
Klasse slope_vector  Vektor fuer Objekte vom Typ slope
  Elemente:
            int      dim ............... Dimension des Vektors vom Typ
                                         slope_vector
            int      lb ................ lower bound
            int      dim ............... upper bound
            int      dim_slopes ........ Dimension der Vektors f_s der
                                         slopes
            slope*   sv ................ Zeiger auf Vektor von slopes

  Elementfunktionen:

            siehe Implementation

Implementationsfile: multi_dim_slope.cpp
------------------------------------------------------------------------*/

class slope_vector{

 private:
  int dim;
  int lb;
  int ub;
  int dim_slopes;
  slope* sv;

 public:

  slope_vector();
  slope_vector(int, int, int );

  slope_vector(slope_vector& );
  ~slope_vector();

  slope_vector& operator=(slope_vector& s);
  slope& operator[](int n);
```

```
  friend int Lb(slope_vector& x);
  friend int Ub(slope_vector& x);

  friend slope_vector InitVar(ivector& );
  friend slope_vector InitVar(rvector& );
  friend slope_vector InitVar(ivector& , ivector& );
  friend slope_vector InitVar(ivector& , rvector& );

  friend slope_vector Init_mid_Var(ivector& );

#ifdef debug
  friend void print_slope_vector(slope_vector& s);
#endif
};

void feval(slope (*f)(slope_vector& ), ivector& , interval& );
void fseval(slope (*f)(slope_vector& ), ivector&,
            interval& , ivector& );
#endif
```

Auszüge aus `multi_dim_slope.cpp`

```
#include " multi_dim_slope.h"
...
/*------------------------------------------------------------------------

operator *

------------------------------------------------------------------------*/
slope operator*(slope& s, slope& t)
{
 int dim1=s.dim;
 int dim2=t.dim;
 if(dim1!=dim2)
   {
    cout << "Fehler in operator*(slope& ,slope& )";
    cout << endl; exit(1);
   }

 slope res(dim1);
 res.f_x=s.f_x*t.f_x;
 if(slope::calc_slope)
  {
    res.f_c=s.f_c*t.f_c;
    for(int i=1; i<=dim1; i++)
      {
        res.f_s[i]=s.f_x*t.f_s[i]+t.f_c*s.f_s[i];
      }
  }
 return res;
}
...
/*------------------------------------------------------------------------

exp  konvex fuer alle x aus R, streng monoton wachsend

------------------------------------------------------------------------*/
slope exp(slope& s)
{
 slope res(s.dim);
 int i;
 res.f_x=exp(s.f_x);

 if(slope::calc_slope)
  {
   res.f_c=exp(s.f_c);
```

```
   real h_d, h_u;

   if(s_1_down(s.f_x, s.f_c, h_d, exp)
      &&
      s_2_up(s.f_x, s.f_c, h_u, exp))
     {
       //konvex u. streng wachsend
       interval sl=_interval(h_d,h_u);
       for(i=1; i<=s.dim; i++)
        {
          res.f_s[i]=s.f_s[i]*sl;
        }
       return res;
     }

   if( (!s_1_down(s.f_x, s.f_c, h_d, exp)) //==0
      &&
        s_2_up(s.f_x, s.f_c, h_u, exp) )
     {
       //konvex u. streng wachsend
       h_d=Inf(res.f_x);

       interval sl=_interval(h_d,h_u);
       for(i=1; i<=s.dim; i++)
        {
          res.f_s[i]=s.f_s[i]*sl;
        }
       return res;
     }

   if( s_1_down(s.f_x, s.f_c, h_d, exp)
      &&
       (!s_2_up(s.f_x, s.f_c, h_u, exp)) ) //==0
     {
       //konvex u. streng wachsend
       h_u=Sup(res.f_x);

       interval sl=_interval(h_d,h_u);
       for(i=1; i<=s.dim; i++)
        {
          res.f_s[i]=s.f_s[i]*sl;
        }
       return res;
     }
   //sonst Ableitung
     for(i=1; i<=s.dim; i++)
        {
          res.f_s[i]=s.f_s[i]*res.f_x;
        }
  }
 return res;
}
...
/*------------------------------------------------------------------------

chi : Verzweigungsfunktion

------------------------------------------------------------------------*/
slope chi(slope& s, slope& q, slope& r)
{
 if(q.dim!=r.dim)
  {cout << "Fehler in chi(slope& , slope& )" << endl; exit(1);}
 slope res(r.dim);
 int i;

 res.f_x=chi(s.f_x, q.f_x, r.f_x);
 if(slope::calc_slope)
```

```
  {
    res.f_c=chi(s.f_c, q.f_c, r.f_c);
    for(i=1; i<=s.dim; i++) res.f_s[i]=chi(s.f_x, q.f_s[i], r.f_s[i]);
  }
 return res;
}
```

C Implementierung der Klasse rslope

Diese Klasse wurde in Abschnitt 3 (S. 16) beschrieben. Zu ihr gehören die Dateien

`s_reverse.h` `s_reverse.cpp` `helpfuncs.h` `helpfuncs.cpp`

Headerfile `s_reverse.h`

```
#ifndef __SREVERSE_H
#define __SREVERSE_H
//cxsc headers
#include "interval.hpp"
//Es folgen weiter Headers...

//gnu g++ headers
#include <iostream.h>
#include <stdlib.h>

//Headerfile fuer Einbinden der Hilfsfunktionen
#include "helpfuncs.h"

enum{
// zum Ueberladen der Verzweigungs- und Standardfunktionen
__sqrt, __sqrt_n,__sqr,__sin,__cos,__tan,__cot,__asin,__acos,__atan,
__acot,__exp,__ln,__sinh,__cosh,__tanh,__coth,__asinh,__acosh,__atanh,
__acoth,__power_n,__abs,__max,__chi,__x = 50,__c,__unminus,__plus,
__minus,__mul,__div,__erg
};

class STAPE;

/*------------------------------------------------------------------------
class rslope  Klasse fuer Objekte zur automatische Berechnung mehrdim-
              ensionaler Intervallsteigungen, werden auf Tape geschrieben

  Elemente:
            STAPE* tapezeiger .... Zeiger auf aktuelles Tape
            int k ................ Nummer der Operation
            interval  f_x ........ 1.Intervalkomponente (Fkt.wert)
            interval  f_c ........ 2.Intervalkomponente (Fkt.wert)
            int op ............... im k-ten Schritt durchzufuehrende Op.
            int l ................ erster Vorgaenger
            int r ................ zweiter Vorgaenger
            int bed .............. Kontrolleintrag fuer Bedingung

  Elementfunktionen:

            siehe Implementation

Implementationsfile: s_reverse.cpp
------------------------------------------------------------------------*/

class rslope{
 //friend class STAPE; waere besser
 private:
```

```
  STAPE* tapezeiger;
  int k ;
  interval f_x;
  interval f_c;
  int  op ;
  int l;
  int r;
  int bed;

 public:

  rslope();
  rslope(STAPE& );
  rslope(STAPE& , interval& );
  rslope(STAPE& , real& );
  rslope(STAPE& , interval& , interval& );
  rslope(const rslope& );

  rslope& operator=(const rslope& );

  friend void print_rslope(rslope& );

  interval get_f_x(){return f_x;};
  interval get_f_c(){return f_c;};
  int get_k(){return k;};
  int get_op(){return op;};
  int get_l(){return l;};
  int get_r(){return r;};
  int get_bed(){return bed;};

  friend void rslope_anfuegen(STAPE& working_tape,
                              rslope& s,
                              interval& fx,
                              interval& fc,
                              int op,
                              int l,
                              int r,
                              int bed );

  friend rslope result(rslope& u);

  friend rslope operator-(rslope& u);

  friend rslope operator+(rslope& u, rslope& v);
  friend rslope operator-(rslope& u, rslope& v);
  friend rslope operator*(rslope& u, rslope& v);
  friend rslope operator/(rslope& u, rslope& v);

  friend rslope operator+(interval& xc, rslope& u);
  //Es folgen weiter Operatoren

  friend rslope sqrt(rslope& u);
  friend rslope sqrt(rslope& u, int n);
  friend rslope sqr(rslope& u);
  friend rslope power(rslope& u, int n);
  //Es folgen weiter Funktionen...
};

/*------------------------------------------------------------------------
class rslope_vec: Vektor mit Elementen der Klasse rslope
                  zum Initialisieren der Unabhaengigen und Abhaeng.
------------------------------------------------------------------------*/

class rslope_vec{
 private:
  int dim;
  int lb; //=1 !
  int ub;
```

```
  rslope* rslope_vector;

 public:
  rslope_vec(int n=1);

  rslope_vec(const rslope_vec& );
  ~rslope_vec();
  rslope_vec& operator=(const rslope_vec& );
  rslope& operator[](int );
  int size();

  friend int Lb(rslope_vec& );
  friend int Ub(rslope_vec& );
};

/*------------------------------------------------------------------------
class list_rslope: fuer liste mit Elementen der Klasse rslope
------------------------------------------------------------------------*/

class list_rslope{
 class list_rslope_element{
  public:

  list_rslope_element* suc;
  int count; //fuer Indexzugriff
  rslope element;
 };
 typedef list_rslope_element* node;
 node head;
 node back;

 public:
  list_rslope();
  list_rslope(rslope& );
  ~list_rslope();

  rslope& operator[](int );

  void push__back(rslope& e);
  void clean_up_list_rslope();
  void print_list_rslope();
  int size();
};

/*------------------------------------------------------------------------
class STAPE: beinhaltet Liste vom Type list_rslope zum Aufzeichnen der
             Codeliste
------------------------------------------------------------------------*/

class STAPE{
 private:
  list_rslope tape; //Liste von Objekten des Typs rslope
  int laenge_tape;  //Laenge des Tapes/Liste

  int stop_writing; // falls 1: es wird kein rslope mehr angefuegt

  int counter;      //Zaehler fuer _erg's
  intvector y_comp; //Stellen in tape, wo _erg geschrieben wird

  int dimension_x; //Dimension Unabhaengige
  int dimension_y; //Dimension Abhaengige

  static int writing; //Flag fuer Ausschliessen von Verschachtelungen
                      //ist 1 falls irgendwo ein Tape geschrieben wird,
                      //sonst 0

 public:
```

```
  friend void init_tape(STAPE&,
                        rslope_vec& ,ivector& ,
                        rslope_vec& );


  friend void init_tape(STAPE& ,
                        rslope_vec& , ivector& ,ivector& ,
                        rslope_vec& );

  friend void init_tape(STAPE& ,
                        rslope_vec& , ivector& , rvector&,
                        rslope_vec& );


  friend void stop_writing_tape(STAPE& working_tape);

  friend void rslopeVar(STAPE& working_tape, rslope_vec& ein_vec,
                        ivector& x, ivector& c);

  friend void rslopeVar(STAPE& working_tape, rslope_vec& ein_vec,
                        ivector& x, rvector& c);

  friend void rslopeVar_mid(STAPE& working_tape, rslope_vec& ein_vec,
                           ivector& x);

  friend void rslope_anfuegen(STAPE& working_tape,
                              rslope& s,
                              interval& fx,
                              interval& fc,
                              int op,
                              int l,
                              int r,
                              int bed );

  int get_laenge_tape();         //gibt Laenge+1 des Tapes aus
  void set_laenge_tape(int n=0); //setzt Laenge des Tapes auf 0

  void set_stop_writing(int& n=1);
  int  get_stop_writing();

  int get_counter();
  void set_counter(int );

  int get_y_comp(int );
  void set_y_comp(int ,int );

  int get_size(); //gibt wirkliche Laenge der Liste zurueck

  int get_dimension_x();
  int get_dimension_y();

  friend void print_tape(STAPE& working_tape);

  friend void clean_up(STAPE& working_tape);

  friend void reverse_sweep(STAPE& wt, ivector& slopes, int y_c);
  friend void reverse_sweep(STAPE& wt, imatrix& slopematrix);
};

//verschiedene Funktionen fuer forward-sweep

void forward_sweep(STAPE& working_tape,
                   rslope (*f)(STAPE& working_tape, rslope_vec& x),
                   ivector& x_in,
                   interval& fx_val,
                   interval& fc_val);

void forward_sweep(STAPE& working_tape,
```

```
                  rslope (*f)(STAPE& working_tape, rslope_vec& x),
                  ivector& x_in, ivector& c_in,
                  interval& fx_val,
                  interval& fc_val);

void forward_sweep(STAPE& working_tape,
                  rslope (*f)(STAPE& working_tape, rslope_vec& x),
                  ivector& x_in, rvector& c_in,
                  interval& fx_val,
                  interval& fc_val);

 //fuer vektorwertige Funktionen

void forward_sweep(STAPE& working_tape,
                  rslope_vec (*f)(STAPE& working_tape, rslope_vec& x),
                  ivector& x_in,
                  ivector& fx_val,
                  ivector& fc_val);

void forward_sweep(STAPE& working_tape,
                  rslope_vec (*f)(STAPE& working_tape, rslope_vec& x),
                  ivector& x_in, ivector& c_in,
                  ivector& fx_val,
                  ivector& fc_val);

void forward_sweep(STAPE& working_tape,
                  rslope_vec (*f)(STAPE& working_tape, rslope_vec& x),
                  ivector& x_in, rvector& c_in,
                  ivector& fx_val,
                  ivector& fc_val);

#endif
```

D Hilfsfunktionen zur Steigungsberechnung

Die Dateien `helpfuncs.h` und `helpfuncs.cpp` definieren und implementieren die in den Steigungsarithmetiken benötigten Hilfsfunktionen.

Headerfile `helpfuncs.h`

```
////////////////////////////////////////////////////////////////////////
//
// Headerfile mit Hilfsfunktionen fuer eindim. und mehrdim.
// Intervallsteigungsarithmetik
//
////////////////////////////////////////////////////////////////////////

#ifndef _HELPFUNCS_H
#define _HELPFUNCS_H

//cxsc headers
#include "interval.hpp"
#include "imath.hpp"

//C++ standard headers
#include <iostream.h>
#include <stdlib.h>

////////////////////////////////////////////////////////////////////////
//
//Definitionen der globalen Hilfsfunktionen
//
```

```
//////////////////////////////////////////////////////////////////////
int even(int n);
interval h_abs(real&, interval& );
interval h_max(interval&, interval&);
interval chi(interval&, interval&, interval&);

int s_1_down(interval&, interval&, real&, interval(*f)(interval& ));
int s_1_up(interval&, interval&, real&, interval(*f)(interval& ));
int s_2_down(interval&, interval&, real&, interval(*f)(interval& ));
int s_2_up(interval&, interval&, real&, interval(*f)(interval& ));


int s_1_down(interval&, interval&, int ,real&,
             interval(*f)(interval& , const int ));
int s_1_up(interval&, interval&, int, real&,
           interval(*f)(interval& , const int));
int s_2_down(interval&, interval&, int, real&,
             interval(*f)(interval& , const int));
int s_2_up(interval&, interval&, int, real&,
           interval(*f)(interval& , const int));
#endif
```

Auszüge aus `helpfuncs.cpp`

```
#include "helpfuncs.h"
...
int s_1_down(interval& x, interval& c, real& res, interval(*f)(interval& ))
{
 interval _x=_interval(Inf(x));
 interval _c=_interval(Inf(c));

 interval h= _x-_c;

 if(_interval(0.0)<=h) return 0;
  else
   {
     interval _fx=f(_x);
     interval _fc=f(_c);
     interval i_res=(_fx-_fc)/h;
     res = Inf(i_res);
     return 1;
   }
}

int s_1_up(interval& x, interval& c, real& res, interval(*f)(interval& ))
{
 interval _x=_interval(Inf(x));
 interval _c=_interval(Inf(c));

 interval h= _x-_c;

 if(_interval(0.0)<=h) return 0;
  else
   {
     interval _fx=f(_x);
     interval _fc=f(_c);
     interval i_res=(_fx-_fc)/h;
     res = Sup(i_res);
     return 1;
   }
}

//es folgen weiter Hilfsfunktionen...
```

Literatur

[1] Berz, M., Bischof, C., Corliss, G. and Griewank, A. (eds.): *Computational Differentiation: Techniques, Applications, and Tools*, SIAM, Philadelphia, Penn., 1996

[2] Bliek, C.: *Fast evaluation of partial derivatives and interval slopes* Reliable Computing, 3:259-268, 1997

[3] Bräuer, M.: *Berechnungsmethoden für Ableitungen und Steigungen und deren Realisierung in C-XSC*, Diplomarbeit, Universität Karlsruhe, 1999.

[4] Bräuer, M., Krämer, W : *Rückwärtsmethode zur automatischen Berechnung von worst-case Fehlerschranken*, Bericht 03/1999 aus dem Forschungsschwerpunkt Komputerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation (CAVN) an der Universität Karlsruhe, 1999.

[5] Fischer, H.: *Special Problems in Automatic Differentiation*, in [5], pp. 43-50

[6] Griewank, A., Corliss, G. F. (eds.): *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM, Philadelphia, Penn., 1991

[7] Hammer, R., Hocks, M., Kulisch, U. und Ratz, D.: *C++ Toolbox for Verified Computing I - Basic Numerical Problems* Springer Verlag, Berlin, 1995

[8] Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0 - A C++ Class Library for Extended Scientific Computing* Preprint BUGHW-WRSWT 2001/1, Universität Wuppertal, 2001

[9] Iri, M., Tsuchiya, T. and Hoshi, M.: *Automatic computation of partial derivatives and rounding error estimates with application to large-scale systems of nonlinear equations* Journal of Computational and Applied Mathematics 24, 1988, pp. 365-392

[10] Kearfott, R.: *Interval Extensions of Non-Smooth Functions for Global Optimization and Nonlinear System Solvers* Computing 57, pp. 149-162, 1996

[11] Kearfott, R.: *Automatic Differentiation of Conditional Branches in an Operator Overloading Context,* in [1], pp. 75-81

[12] Kedem, G.: *Automatic Differentiation of Computer Programs* ACM Trans. Math. Software, 6, pp. 150-165, 1980

[13] Klatte, R., Kulisch, U., Lawo, C., Rauch, M., Wiethoff, A.: *C-XSC, A C++ Class Library for Extended Scientific Computing* Springer-Verlag Berlin, Heidelberg, New York, 1993

[14] Kulisch, U., Lohner, R., Facius, A.(eds): Perspectives on Enclosure Methods, Springer, 2001

[15] Lang, B.: *A Comparison of Techniques for Evaluating Centered Forms*, in [14], 2001

[16] Neumeier, A.: *Interval Methods for Systems of Equations* Cambridge University Press, Cambridge, UK, 1990

[17] Oliveira, J. B.: *New slope methods for sharper interval functions and a note on Fisher's acceleration method.* Reliable Comput., 2(3):299-320, 1996

[18] Ratz, D.: *Automatic Slope Computation and its Application in Nonsmooth Global Optimization* Shaker Verlag, Aachen, Habilitationsschrift, 1998

[19] Ratz, D.: *Globale Optimierung nichtdiffernzierbarer Funktionen unter Einsatz von Intervallsteigungen* ZAMM 79, S1, 247-250, 1999.

[20] Rump, S. M.: *Expansion and Estimation of the Range of Nonlinear Functions* Mathematics of Computation, Vol. 65, No. 216, pp. 1503-1512, 1996

[21] Rump, S. M.: *Inclusion of Zeros of Nowhere Differentiable n-Dimensional Functions* Reliable Computing, Vol. 3, pp. 5-16, 1997

[22] Stroustrup, B.: *The C++ Programming Language*. Special Edition, Addison-Wesley, Reading, Mass., 2000.
Deutsche Übersetzung: *Die C++ Programmiersprache*, 4. Auflage, Addison-Wesley, München, 2000