Einführung in die Automatische Differentiation

Hans Joachim Ferreau · 10.05.2004

im Rahmen des Seminars „Automatische Differentiation“
von Dr. Johannes Schlöder und Dr. Ekaterina Kostina,
Sommersemester 2004, Universität Heidelberg

Inhaltsverzeichnis

1. Grundidee der Automatischen Differentiation

1. Grundidee der Automatischen Differentiation

1.1 Motivation

1.1 Motivation

- Viele Anwendungen brauchen Ableitungs-informationen (manchmal 3. Ableitung und höher)
- Funktionen liegen als Programmcode vor
- Symbolische Differentiation und Differenzen-quotienten oft ungeeignet
- Automatische Differentiation kann Ableitungen effizient und im Rahmen der Maschinengenauigkeit *exakt* berechnen

1. Grundidee der Automatischen Differentiation

1.2 Kettenregel und ein Beispiel

1.2 Kettenregel

- Programmcode kann in elementare Operationen zerlegt werden
- Diese können einfach differenziert und mit Kettenregel verknüpft werden
- Satz 1.1 (Kettenregel)

$$U \xrightarrow{g} V \xrightarrow{f} Z$$

$$(f \circ g)'(x) = f'(y) \cdot g'(x)$$

1.2 Kettenregel

- **Korollar 1.2**

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}$$
$$g : \mathbb{R} \longrightarrow \mathbb{R}^m$$

$$\frac{\partial f \circ g}{\partial x}(x) = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i}(y) \cdot g_i'(x)$$

1.2 und ein Beispiel

- **Wir wollen die Funktion**

$$y = F(x_1, x_2) =$$
$$(\sin(x_1/x_2) + x_1/x_2 - \exp(x_2))\,(x_1/x_2 - \exp(x_2))$$

und ihre Ableitung im Punkt $(1.5,0.5)^T$ auswerten

1.2 und ein Beispiel

- **Dazu zerlegen wir sie in Elementaroperationen und werten diese sukzessive aus:**

v_{-1}	$=$	x_1	$=$	1.5000		
v_0	$=$	x_2	$=$	0.5000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
v_6	$=$	$v_5 \cdot v_4$	$=$	$1.4924 \cdot 1.3513$	$=$	2.0167
y	$=$	v_6	$=$	2.0167		

1.2 und ein Beispiel

- Um sie nach x_2 zu differenzieren, müssen alle Zwischenvariablen v_i gemäß der Kettenregel nach x_2 differenziert werden (Vorwärtsmodus):

$$
\begin{aligned}
v_{-1} = x_1 \quad &\Longrightarrow \quad \dot{v}_{-1} = 0 \\
v_0 = x_2 \quad &\Longrightarrow \quad \dot{v}_0 = 1 \\
v_1 = \frac{v_{-1}}{v_0} \quad &\Longrightarrow \quad \dot{v}_1 = \frac{\partial v_1}{\partial v_{-1}} \dot{v}_{-1} + \frac{\partial v_1}{\partial v_0} \dot{v}_0 \\
&\qquad\qquad = \frac{1}{v_0} \dot{v}_{-1} + \frac{-v_{-1}}{v_0^2} \dot{v}_0 = \frac{\dot{v}_{-1} - v_1 \dot{v}_0}{v_0} \\
&\ldots
\end{aligned}
$$

1.2 und ein Beispiel

v_{-1}	$=$	x_1	$=$	1.5000		
$\dot{v}_{-1}$	$=$	$\dot{x}_1$	$=$	0.0000		
v_0	$=$	x_2	$=$	0.5000		
$\dot{v}_0$	$=$	$\dot{x}_2$	$=$	1.0000		
v_1	$=$	v_{-1}/v_0	$=$	$1.5000/0.5000$	$=$	3.0000
$\dot{v}_1$	$=$	$(\dot{v}_{-1} - v_1 \cdot \dot{v}_0)/v_0$	$=$	$(0.0000 - 3.0000 \cdot 1.0000)/0.5000$	$=$	-6.0000
v_2	$=$	$\sin(v_1)$	$=$	$\sin(3.0000)$	$=$	0.1411
$\dot{v}_2$	$=$	$\cos(v_1)\dot{v}_1$	$=$	$-0.9900 \cdot (-6.0000)$	$=$	5.9400
v_3	$=$	$\exp(v_0)$	$=$	$\exp(0.5000)$	$=$	1.6487
$\dot{v}_3$	$=$	$v_3 \cdot \dot{v}_0$	$=$	$1.6487 \cdot 1.0000$	$=$	1.6487
v_4	$=$	$v_1 - v_3$	$=$	$3.0000 - 1.6487$	$=$	1.3513
$\dot{v}_4$	$=$	$\dot{v}_1 - \dot{v}_3$	$=$	$-6.0000 - 1.6487$	$=$	-7.6487
v_5	$=$	$v_2 + v_4$	$=$	$0.1411 + 1.3513$	$=$	1.4924
$\dot{v}_5$	$=$	$\dot{v}_2 + \dot{v}_4$	$=$	$5.9400 + (-7.6487)$	$=$	-1.7088
v_6	$=$	$v_5 \cdot v_4$	$=$	$1.4924 \cdot 1.3513$	$=$	2.0167
$\dot{v}_6$	$=$	$\dot{v}_5 \cdot v_4 + v_5 \cdot \dot{v}_4$	$=$	$-1.7088 \cdot 1.3513 + 1.4924 \cdot (-7.6487)$	$=$	-13.7240
y	$=$	v_6	$=$	2.0167		
$\dot{y}$	$=$	$\dot{v}_6$	$=$	-13.7240		

1. Grundidee der Automatischen Differentiation

1.3 Vergleich mit anderen Differentiationstechniken

1.3 Vergleich mit anderen Differentiationstechniken

- Automatische Differentiation ist mit symbolischer Differentiation verwandt
- ABER: mehrfach auftretende Teilausdrücke werden nur *einmal numerisch* ausgewertet
- Dagegen behandelt die symbolische Differentiation jeden Teilausdruck separat
- Dies führt zu sehr langen Ausdrücken

1.3 Vergleich mit anderen Differentiationstechniken

- Automatische Differentiation liefert ausführbaren Programmcode für die Ableitung
- Dieser kann im Rahmen der Maschinengenauigkeit *exakt* ausgewertet werden
- Anders bei finiten Differenzen (einseitige Differenz):

$$\frac{\partial f}{\partial x_i}(x) = \lim_{\eta \to \infty} \frac{f(x + \eta e_i) - f(x)}{\eta}$$

$$= \frac{f(x + \eta e_i) - f(x)}{\eta} + \mathcal{O}(\eta)$$

1.3 Vergleich mit anderen Differentiationstechniken

- **Selbst bei optimaler Wahl gilt nach [5]:**

$$\left| \frac{\partial f}{\partial x_i}(x) - \frac{f(x + \eta_{opt} e_i) - f(x)}{\eta_{opt}} \right| \approx \sqrt{\varepsilon_{RND}}$$

- **Es geht die Hälfte der signifikanten Stellen verloren!**
- **Selbst bei aufwändigerer zentraler Differenz geht ein Drittel der signifikanten Stellen verloren.**

1.3 Vergleich mit anderen Differentiationstechniken

- **Anders bei der Automatischen Differentiation:**

  Regel 0: Automatische Differentiation ist nicht von Abbruchfehlern betroffen.

- **Aber:**

  Regel 1: In manchen Anwendungen ist der Einsatz von Differenzenquotienten sinnvoll.

2. Konzepte rund um die Funktionsauswertung

2. Konzepte rund um die Funktionsauswertung

2.1 Programmebenen

2.1 Programmebenen

- Funktionen liegen als Computerprogramm vor
- Wir unterscheiden drei Programmebenen:

  1. *Auswertungsprogramm (evaluation program)*
     – alle Programmiermethoden zulässig

  2. *Auswertungsprozedur (evaluation procedure)*
     – keine Verzweigungen, Rekursionen, Schleifen
     – Unterprogramme dürfen mit Parametern aufgerufen werden

2.1 Programmebenen

- Jedes Auswertungsprogramm lässt sich durch *bedingtes Compilieren (instantiation)* in eine Auswertungsprozedur umwandeln
- Dazu werden Verzweigungsbedingungen ausgewertet, Rekursionen aufgelöst und Schleifen ersetzt (loop unrolling)
- Durch inlining der Unterprozeduren erhält man einen

  3. *Auswertungspfad (evaluation trace)*

2.1 Programmebenen

- Wir erhalten folgende Beziehung:

2. Konzepte rund um die Funktionsauswertung

2.2 Dreiteilige Funktionsauswertung und Datenabhängigkeitsrelation

2.2 Dreiteilige Funktionsauswertung

- **Die Funktionsauswertung ist dreigeteilt:**

1. Eingabevariablen x_i werden in interne Variablen v_{i-n}, $1 \leq i \leq n$, kopiert

$$\begin{array}{lcll} v_{-1} & = & x_1 & = & 1.5000 \\ v_0 & = & x_2 & = & 0.5000 \end{array}$$

2. Zwischenvariablen v_i, $1 \leq i \leq l$, werden berechnet

$$\begin{array}{lcllcl} v_1 & = & v_{-1}/v_0 & = & 1.5000/0.5000 & = & 3.0000 \\ v_2 & = & \sin(v_1) & = & \sin(3.0000) & = & 0.1411 \\ v_3 & = & \exp(v_0) & = & \exp(0.5000) & = & 1.6487 \\ v_4 & = & v_1 - v_3 & = & 3.0000 - 1.6487 & = & 1.3513 \\ v_5 & = & v_2 + v_4 & = & 0.1411 + 1.3513 & = & 1.4924 \\ v_6 & = & v_5 \cdot v_4 & = & 1.4924 \cdot 1.3513 & = & 2.0167 \end{array}$$

2.2 Dreiteilige Funktionsauswertung

3. Das Ergebnis v_{l-m+i} wird in die Ergebnisvariablen y_i, $1 \leq i \leq m$, kopiert

$$y = v_6 = 2.0167$$

- **Die Variablen v_i genügen folgendem Schema:**

$$v = (\overbrace{v_{1-n}, v_{2-n}, \ldots, v_{-1}, v_0}^{x \in \mathbb{R}^n},$$
$$v_1, v_2, \ldots, v_{l-m-1}, v_{l-m},$$
$$\underbrace{v_{l-m+1}, v_{l-m+2}, \ldots, v_{l-1}, v_l}_{y \in \mathbb{R}^m}) \in \mathbb{R}^{n+l}$$

2.2 und Datenabhängigkeitsrelation

- Jedem v_i, $1 \leq i \leq l$, wird das Ergebnis einer Elementarfunktion $\varphi_i \in \Phi$ zugewiesen
- Dabei kann φ_i von jeder Variable v_j, $j < i$, abhängen
- **Definition 2.1 (Datenabhängigkeitsrelation)**

Auf der Indexmenge $I = \{1-n, \ldots, l\}$ *des Variablenvektors* v *definieren wir die antisymmetrische Relation*

$$j \prec i \quad :\Longleftrightarrow \quad v_i = \varphi_i(v_j) \text{ hängt direkt von } v_j \text{ ab}$$

Sie heißt Datenabhängigkeitsrelation (data dependence relation).

2.2 und Datenabhängigkeitsrelation

Ihr transitiver Abschluss

$$j \prec^* i \quad :\Longleftrightarrow \quad \exists\, p_1, \ldots, p_r \in I :$$

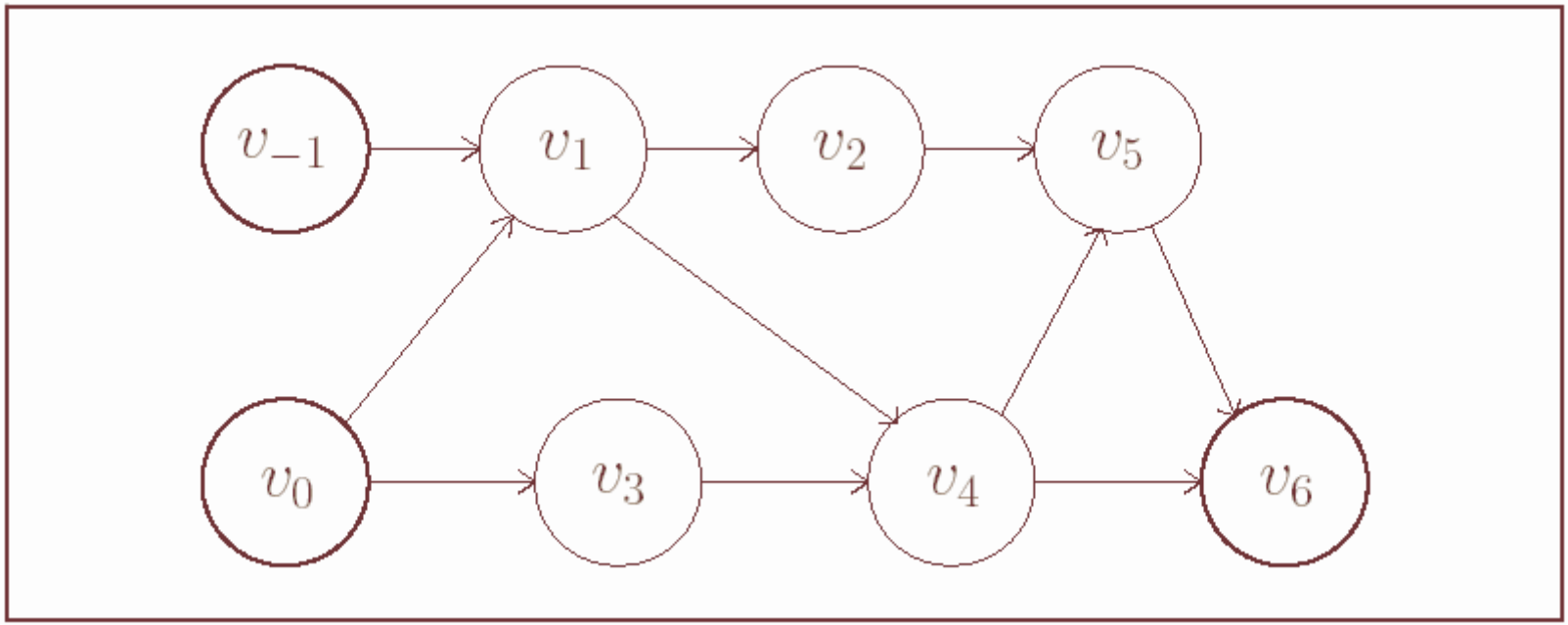
$$j \prec p_1 \ \wedge \ p_k \prec p_{k+1}\ \forall\, k \in \{1, \ldots, r-1\} \ \wedge \ p_r \prec i$$

stellt eine partielle Ordnung auf I dar. $i \prec i$ entspricht dabei einer Iteration, welche vorerst nicht zulässig ist.

- **Datenabhängigkeiten können durch einen *Berechnungsgraphen (computational graph)* dargestellt werden**
- **Dieser ist gerichtet, zusammenhängend und azyklisch**

2.2 und Datenabhängigkeitsrelation

- **Berechnungsgraph des Beispiels:**



- **Es gilt u. a. $1 \prec 4$, $3 \prec 4$, $0 \not\prec 4$, $0 \prec^* 4$**

2.2 und Datenabhängigkeitsrelation

- **Auf Grund von Definition 2.1 können Auswertungsprozeduren formalisiert werden:**

$$\begin{array}{rcll} v_{i-n} & = & x_i & \forall\, i \in \{1,\ldots,n\} \\ v_i & = & \varphi_i(v_j)_{j \prec i} & \forall\, i \in \{1,\ldots,l\} \\ y_i & = & v_{l-m+i} & \forall\, i \in \{1,\ldots,m\} \end{array}$$

- **Dies liefert F als Komposition von Elementarfunkionen:**

$$\begin{array}{rcl} & F(x_1,\ldots,x_n) & = & (y_1,\ldots,y_m) \\ \Longleftrightarrow & F(v_{1-n},\ldots,v_0) & = & (v_{l-m+1},\ldots,v_l) \\ & = (\varphi_i \circ \varphi_{i,1} \circ \ldots \circ \varphi_{i,n_i}(v_{j_i}))_{i=l-m+1,\ldots,l} & & \end{array}$$

2.2 und Datenabhängigkeitsrelation

- Die obigen Darstellung ist nicht eindeutig: Verschiedene Auswertungsprozeduren können die selbe Funktion beschreiben
- Schon einfache symbolische Umformungen führen zu unterschiedlicher Stabilität und Effizienz
- Dies ist jedoch kein spezifisches Problem der AD
- Regel 2: Ein abgeleitetes Auswertungsprogramm ist genau so gut oder schlecht wie das ursprüngliche Auswertungsprogramm.

2. Konzepte rund um die Funktionsauswertung

2.3 Erweiterungen

2.3 Erweiterungen

- Zwischenvariablen v_i lassen sich als Funktionen $v_i(v_j)$, $1 \leq j < i \leq l\text{-}m$, auffassen
- Sie dürfen auch *vektorwertig* sein. Wir definieren:

$$m_i := \dim(v_i) \geq 1 \quad \forall\, i \in \{1, \ldots, l-m\}$$
$$m_i := 1 \quad \forall\, i \in \{1-n, \ldots, 0\} \cup \{l-m+1, \ldots, l\}$$

- Elementarfunktionen aus der linearen Algebra
- Auswertungsprozedur als Super-Elementarfunktion

2.3 Erweiterungen

- Bei *mehrstelligen* Funktionen fassen wir die Argumente zu *einem* Argumentvektor zusammen:

$$u_i := (v_j)_{j \prec i} \in \mathbb{R}^{n_i} \quad \forall\, i \in \{1, \ldots, l\}$$

$$\text{mit} \quad n_i := \sum_{j \prec i} m_j \quad \forall\, i \in \{1, \ldots, l\}$$

- Dies dient vorrangig zur Vereinfachung der Notation
- In der Praxis führt dies zu Effizienzeinbußen

2.3 Erweiterungen

- Wir können eine Auswertungsprozedur auch als nichtlineares Gleichungssystem formulieren

$$0 = E(x; v) := (\varphi_i(u_i) - v_i)_{i=1-n,\ldots,l}$$

- Weiter führen wir folgende Notation ein:

$$c_{ij} := c_{ij}(u_i) := \frac{\partial \varphi_i}{\partial v_j}(u_i) \quad \forall\, i, j \in \{1-n, \ldots, l\}$$

- Komponenten von y seien paarweise unabhängig

2.3 Erweiterungen

- **Dann hat die Jacobi-Matrix von E(x; v) die Form**

$$E'(x;v) = (c_{ij} - \delta_{ij})_{i,j=1-n,\ldots,l} = C - Id_{n+l}$$

$$= \left(\begin{array}{cccc|cccccccc} -1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ \hline * & \cdots & \cdots & * & -1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & & \vdots & * & \ddots & \ddots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots & \vdots & \ddots & \ddots & 0 & \vdots & & \ddots & \vdots \\ * & \cdots & \cdots & * & * & \cdots & * & -1 & 0 & \cdots & \cdots & 0 \\ * & \cdots & \cdots & * & * & \cdots & \cdots & * & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots & \vdots & \ddots & & \vdots & 0 & \ddots & \ddots & 0 \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots & \vdots & \ddots & \ddots & 0 \\ * & \cdots & \cdots & * & * & \cdots & \cdots & * & 0 & \cdots & 0 & -1 \end{array}\right) = \left(\begin{array}{ccc} -Id_n & 0 & 0 \\ B & L - Id_{l-m} & 0 \\ R & T & -Id_m \end{array}\right)$$

2.3 Erweiterungen

- Sie ist eine normierte, untere Dreiecksmatrix und daher regulär
- Aus dem Satz über implizite Funktionen folgt die Existenz einer Funktion

$$g : \mathbb{R}^n \longrightarrow \mathbb{R}^l$$

$$E(x; v) = 0 \iff v_i = g_i(x) \quad \forall\, i \in \{1, \ldots, l\}$$

- Damit ist der Ergebnisvektor y implizit durch eine Funktion in x bestimmt

2. Konzepte rund um die Funktionsauswertung

2.4 Elementarfunktionen und ihre Differenzierbarkeit

2.4 Elementarfunktionen

- Welche Funktionen sollten in der Menge der Elementarfunktionen Φ enthalten sein?
- Mit dem sog. *polynomialen Kern (polynomial core)*

$$\Phi_{min} = \{+, \cdot, -, c\}$$

können alle Polynome ausgewertet werden

- Außerdem nötig: exp(a), log(a), sin(a), cos(a), 1/a

2.4 Elementarfunktionen

	glatt	Lipschitz-stetig	sonstige
essentiell	$a+b,\ a\cdot b,\ -a,\ c,$ $1/a,\ \exp(a),\ \log(a),$ $\sin(a),\ \cos(a),\ \lvert a\rvert^c\ (c>1)$	$\lvert a\rvert,\ \lvert(a,b)\rvert=\sqrt{a^2+b^2}$	$\text{heaviside}(a)$
optional	$a-b,\ a/b,\ c\cdot a,\ c\pm a,$ $a^k,\ a^c,\ \tan(a),\ \arcsin(a)$	$\max(a,b),\ \min(a,b)$	$\text{sign}(a),$ if-then-else
vektorwertig	$\sum_k a_k\cdot b_k,\ \sum_k c_k\cdot a_k$	$\max\{\lvert a_k\rvert\}_k,\ \sum_k \lvert a_k\rvert,\ \left(\sum_k a_k^2\right)^{1/2}$	–

$k\in\mathbb{N},\quad a,b,a_k,b_k\in\mathbb{R}$ (Variablen), $\quad c,c_k\in\mathbb{R}$ (Konstanten)

- **Da wir die Kettenregel anwenden wollen, beschränken wir uns auf glatte Funktionen**

2.4 und ihre Differenzierbarkeit

- **Annahme ED (Elementare Differenzierbarkeit):**

  Alle Elementarfunktionen $\varphi_i \in \Phi$ sind in ihrem offenen Definitionsbereich $D_i \in \mathbb{R}^n$ d-mal stetig-differenzierbar, $0 \leq d \leq \infty$.

- **Satz 2.2**

  Falls die Annahme ED gilt, ist die Menge

  $$D := \{x \in \mathbb{R}^n \mid y = F(x) \textit{ ist durch Auswertungsprozedur wohldefiniert}\}$$

  eine offene *Teilmenge des* $\mathbb{R}^n$ *und* $F \in \mathcal{C}^d(D)$, $0 \leq d \leq \infty$.

  Beweis: Ist F in x wohldefiniert, so muss $u_i = u_i(x) \in D_i$ für alle $i \in \{1, \ldots, l\}$ gelten. Da alle D_i nach Voraussetzung offen sind, folgt aus Satz 1.1, dass alle v_i und insbesondere auch alle $y_i = F_i(x)$ in einer vollen Umgebung von x (mindestens) d-mal stetig-differenzierbar sind. □

2.4 und ihre Differenzierbarkeit

- **Korollar 2.3**

  Werden von der Vektorfunktion $F : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ Ableitungen bis zur Ordnung $d > 0$ benötigt, müssen die Elementarfunktionen $\varphi_i \in \Phi$ so gewählt werden, dass sie auf ihren offenen Definitionsbereichen d-mal stetig-differenzierbar sind und die Auswertungsprozedur für alle $x \in D$ wohldefiniert ist.

- **Der maximale Definitionsbereich D kann leer sein**
- **Die Zahl d kann durch verschiedene Umstände beschränkt sein**

2.4 und ihre Differenzierbarkeit

- Wird eine Funktion durch Splines interpoliert, so gilt typischerweise d = 2
- Wenn Potenzen des Betrags auftauchen, ist d endlich:

$$F(x) = \sqrt{x^6} = |x|^3 \implies F'(x) = 3x\,|x|$$

- Auswertung durch

$$v_0 := x,\ v_1 := v_0^6,\ y = v_2 := \sqrt{v_1} \quad \text{oder}$$

$$\widetilde{v}_0 := x,\ \widetilde{v}_1 := |\widetilde{v}_0|\ ,\ y = \widetilde{v}_2 := \widetilde{v}_1^3$$

- Aber Ableitungen im Nullpunkt nicht endlich

2.4 und ihre Differenzierbarkeit

- Dies kann man umgehen, wenn man

$$\varphi(x) = |x|^c, \quad \varphi'(x) = cx\,|x|^{c-2} \in \mathcal{C}^{\lfloor c-1 \rfloor}(\mathbb{R})$$

  als „Mikrocode“ mitliefert

- Wir wählen die Menge der Elementarfunktionen so, dass ihre Elemente Annahme ED mit d ≥ 1 erfüllen:

$$\Phi := \{c,\ \pm,\ \cdot,\ \exp,\ \log,\ \sin,\ \cos,\ ^{-1},\ \ldots\}$$

- Menge der Funktionen, die als Auswertungsprozedur mit Elementen aus Φ definiert werden können:

$$\mathcal{F} := \operatorname{span}(\Phi) \subseteq \mathcal{C}^d(\mathbb{R}^n)$$

2. Konzepte rund um die Funktionsauswertung

2.5 Speicherverwaltung

2.5 Speicherverwaltung

- **Bisher wurden Zwischenvariablen nur aus mathematischer Sicht betrachtet**
- **Zwischenvariablen sollen sich nun Speicherplätze teilen dürfen**
- **Definition 2.4 (Adressierungsfunktion)**

Eine Adressierungsfunktion (adressing scheme) *ist eine Abbildung*

$$\&: \{1,\ldots,l\} \rightarrow \mathcal{P}(\mathcal{R}) \setminus \emptyset, \quad \mathcal{R} := \{1,\ldots,r\},\ r \in \mathbb{N}$$

die jedem v_i, $i > 0$, *eine nichtleere Teilmenge* $\&v_i = \&i$ *der Speicherplätze* $\mathcal{R}$ *zuordnet.*

2.5 Speicherverwaltung

- **Speicherplätze dürfen nicht wiederverwendet werden, bevor ihr Inhalt das letzte Mal gebraucht wurde**
- **Man nennt die Adressfunktion *vorwärtskompatibel (forward compatibel)* wenn**

$$j \prec i,\ j < k \leq i \implies \&v_k \neq \&v_j \ (\iff \&v_k \cap \&v_j = \emptyset)$$

- **Analog kann man *Rückwärtskompatibilität (reverse compatibility)* fordern**

$$j \prec i,\ j \leq k < i \implies \&v_i \neq \&v_k$$

2.5 Speicherverwaltung

- **Wenn man einer Zwischenvariable erlaubt, ihr eigenes Argument zu überschreiben**

$$j \prec i, \quad \&v_i = \&v_j$$

spricht man von *schwacher* Kompatibilität

- **Wir wollen unser Beispiel**

$$y = F(x_1, x_2) =$$

$$(\sin(x_1/x_2) + x_1/x_2 - \exp(x_2))\,(x_1/x_2 - \exp(x_2))$$

entsprechend umformulieren:

2.5 Speicherverwaltung

$v_1 = \frac{v_{-1}}{v_0}$	$v_1 = \frac{v_{-1}}{v_0}$
$v_2 = \sin(v_1)$	$v_2 = \exp(v_0)$
$v_3 = \exp(v_0)$	$v_3 = v_1 - v_2$
$v_4 = v_1 - v_3$	$v_1 = \sin(v_1)$
$v_1 = v_2 + v_4$	$v_2 = v_1 + v_3$
$v_2 = v_1 \cdot v_4$	$v_3 = v_3 \cdot v_2$
$y = v_2$	$y = v_3$

2.5 Speicherverwaltung

- Anzahl der benötigten Speicherplätze hängt von verwendeter Adressierungsfunktion ab
- Noch eine Bezeichnung:

$$RAM(F) := r := \max\{k \in \mathbb{N} \mid k \in \&v_i,\ 1 \leq i \leq l\}$$

- Sie deutet an, dass der Zugriff auf die Speicherplätze in der Regel zufällig erfolgt
- Auch wenn keine Vor-/Rückwärtskompatibilität gilt, wird eine (andere) Funktion ausgewertet

2.5 Speicherverwaltung

- Gilt z. B. &v_0 = &v_1 in den Zeilen

$$
\begin{aligned}
v_1 &= v_{-1}/v_0 \\
\dot{v}_1 &= \frac{\dot{v}_{-1}v_0 - v_{-1}\dot{v}_0}{v_0^2} = \frac{\dot{v}_{-1} - v_1\dot{v}_0}{v_0}
\end{aligned}
$$

  so wird die Ableitung falsch berechnet

- Als weitere Bedingung wollen wir unsere Auswertungsprozedur *alias-sicher (alias-safe)* machen
- Beim Vorwärtsmodus hilft Vertauschen von Funktions- und Ableitungsberechnung

2. Konzepte rund um die Funktionsauswertung

2.6 Ein zeitbezogenes Komplexitätsmodell

2.6 Komplexitätsmodell

- Automatische Differentiation erlaubt eine genaue Vorhersage des Rechenaufwands der Ableitung
- In der Regel nicht mehr als das Fünffache einer zugrundeliegenden Funktionsauswertung
- Wir entwickeln nun ein einfaches, *zeitbezogenes Komplexitätsmodell* für die sequentielle Ausführung
- Dazu zählen wir die Anzahl der arithmetischen Operationen und der Speicherzugriffe
- Dabei wird ein *uniformer Speicher* vorausgesetzt

2.6 Komplexitätsmodell

- Wir bezeichnen mit

$$eval(F) \qquad task(F)$$

die Auswertung einer Funktion F bzw. eine andere additive Aufgabe

- Und mit

$$eval'(\varphi) \qquad task'(\varphi)$$

die Auswertung einer Elementarfunktion bzw. einen Teil der Aufgabe, der eine solche betrifft

2.6 Komplexitätsmodell

- **Annahme TA (Zeitliche Additivität):**

$$WORK\{task(F)\} \leq \sum_{i=1}^{l} WORK\{task'(\varphi_i)\}$$

$$WORK\{eval(F)\} = \sum_{i=1}^{l} WORK\{eval(\varphi_i)\}$$

- **Dabei wählen wir als *WORK{task(F)}* einen vierdimensionalen, reellen Vektor:**

$$WORK\{task(F)\} := \begin{pmatrix} MOVES(F) \\ ADDS(F) \\ MULTS(F) \\ NLOPS(F) \end{pmatrix}$$

2.6 Komplexitätsmodell

- Jeder Elementaroperation lässt sich ein solcher Vektor zuordnen
- Z. B. benötigt die *Auswertung* einer Addition
  3 Speicherzugriffe und eine Addition (als Kostenmaß)

	c	$\pm$	$\cdot$	ψ
$MOVES$	1	3	3	2
$ADDS$	0	1	0	0
$MULTS$	0	0	1	0
$NLOPS$	0	0	0	1

2.6 Komplexitätsmodell

- Wir bezeichnen diese Tabelle als Kostenmatrix W_{eval}
- Enthalten die Komponenten des Vektors |F| die Anzahl der Initialisierungen, Additionen, Multiplikationen und der nichtlinearen Operationen, so ergibt sich

$$WORK\{eval(F)\} = W_{eval}\,|F| = \begin{pmatrix} 1 & 3 & 3 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{pmatrix}$$

2.6 Komplexitätsmodell

- **Weiter definieren wir das *Laufzeit-Funktional***

$$TIME\{task(F)\} := w^T WORK\{task(F)\}, \quad w \in \mathbb{R}^{dim(WORK)}$$

- **Die Komponenten von w sind positiv und enthalten z. B. die Anzahl der CPU-Zyklen pro Operation**
- **Damit lässt sich unser Modell an verschiedene Rechnersysteme anpassen**
- **Wir wählen**

$$w^T = (\mu, 1, \pi, \nu)^T \quad \text{mit} \quad \mu \geq \max(1, \pi/2),\ \pi \geq 1,\ \nu \geq 2\pi$$

2.6 Komplexitätsmodell

- **Unter der Annahme TA leiten wir nun eine wichtige Schranke her:**

$$\frac{TIME\{task(F)\}}{TIME\{eval(F)\}} = \frac{w^T WORK\{task(F)\}}{w^T WORK\{eval(F)\}}$$

$$\leq \frac{w^T \sum_{i=1}^{l} WORK\{task'(\varphi_i)\}}{w^T \sum_{i=1}^{l} WORK\{eval(\varphi_i)\}} = \frac{\sum_{i=1}^{l} TIME\{task'(\varphi_i)\}}{\sum_{i=1}^{l} TIME\{eval(\varphi_i)\}}$$

$$\leq \max_{1\leq i\leq l} \frac{TIME\{task'(\varphi_i)\}}{TIME\{eval(\varphi_i)\}} \leq \sup_{\varphi\in\Phi} \frac{TIME\{task'(\varphi)\}}{TIME\{eval(\varphi)\}}$$

2.6 Komplexitätsmodell

- **Wenn Φ nur endlich viele Elemente enthält, haben wir damit eine einheitliche Schranke für das Laufzeitverhältnis von *task*(*F*) zu *eval*(*F*) bewiesen**

- **Definition 2.5 (Linear beschränkte Komplexität)**

  Wir sagen eine Aufgabe hat linear beschränkte Komplexität (linearly bounded complexity) über $\mathcal{F}$, *falls eine quadratische Matrix* C_{task} *existiert, so dass*

  $$WORK\{task(F)\} \leq C_{task} WORK\{eval(F)\} \quad \forall F \in \mathcal{F}$$

  Wir nennen eine solche Matrix Komplexitätsschranke.

- **Neben linear beschränkter Komplexität gibt es auch *polynomial* beschränkte Komplexität**

2.6 Komplexitätsmodell

- **Man kann zeigen:**

$$WORK\{eval(F')\} = \mathcal{O}\left(WORK\{eval(F)\}^2\right)$$

- **Doch bereits polynomialer Kern ist unendlich, daher**
- **Annahme EB (Lineare Beschränktheit der elementaren Aufgaben)**

Es gibt eine Matrix $C_{task}(\Phi)$, *so dass*

$$WORK\{task'(\varphi)\} \leq C_{task}(\Phi) \cdot WORK\{eval(\varphi)\}$$

für alle $\varphi \in \Phi$ *gilt.*

- **Sie gilt insbesondere, falls die Menge**

$$\{(WORK\{task'(\varphi)\}, WORK\{eval(\varphi)\}) \mid \varphi \in \Phi\}$$

endlich ist

2.6 Komplexitätsmodell

- **Damit können wir folgenden Satz beweisen:**

- **Satz 2.6 (Lineare Beschränktheit von additiven Aufgaben)**

Die Annahmen TA und EB implizieren linear beschränkte Komplexität über $\mathcal{F}$ im Sinne der Definition 2.5 mit $C_{task} = C_{task}(\Phi)$. Außerdem gilt für jedes Laufzeit-Funktional die Abschätzung

$$TIME\{task(F)\} \leq \omega_{task} TIME\{eval(F)\} \quad \forall F \in \mathcal{F}$$

$$mit \quad \omega_{task} := \sup_{\varphi \in \Phi} \frac{w^T WORK\{task'(\varphi)\}}{w^T WORK\{eval(\varphi)\}}$$

$$\leq \left\| DC_{task} D^{-1} \right\|_1 < \infty, \quad D := diag(w)$$

2.6 Komplexitätsmodell

Beweis: Aus den beiden Annahmen folgt direkt

$$WORK\{task(F)\} \leq \sum_{i=1}^{l} WORK\{task'(\varphi_i)\}$$

$$\leq \sum_{i=1}^{l} C_{task}(\Phi) \cdot WORK\{eval(\varphi_i)\}$$

$$= C_{task}(\Phi) \cdot WORK\{eval(F)\}$$

Damit ist der erste Teil der Aussage bewiesen. Wir folgern weiter

2.6 Komplexitätsmodell

$$\frac{TIME\{task(F)\}}{TIME\{eval(F)\}} \leq \max_{1\leq i\leq l} \frac{TIME\{task'(\varphi_i)\}}{TIME\{eval(\varphi_i)\}}$$

$$\leq \sup_{\varphi\in\Phi} \frac{w^T WORK\{task'(\varphi)\}}{w^T WORK\{eval(\varphi)\}}$$

$$\leq \sup_{\varphi\in\Phi} \frac{w^T C_{task} WORK\{eval(\varphi)\}}{w^T WORK\{eval(\varphi)\}}$$

$$\leq \sup_{0\neq z\in\mathbb{R}^{\dim(w)}} \frac{w^T C_{task} z}{w^T z}$$

2.6 Komplexitätsmodell

$$= \sup_{0 \neq z \in \mathbb{R}^{\dim(w)}} \frac{\left\| DC_{task} z \right\|_1}{\left\| Dz \right\|_1}$$

$$= \left\| DC_{task} D^{-1} \right\|_1 < \infty \qquad \square$$

- **Wir formulieren diesen Satz als**
- **Regel 3: Additive Aufgaben, die über der Menge der Elementarfunktionen linear beschränkte Komplexität besitzen, haben auch über der Menge der aus diesen Elementarfunktionen zusammengesetzten Funktionen linear beschränkte Komplexität.**

2.6 Komplexitätsmodell

- Unter den Annahmen TA und EB ist also das Verhältnis von *TIME*{*task*(*F*)} zu *TIME*{*eval*(*F*)} beschränkt
- Für unseren vier-elementigen Komplexitätsvektor ergibt sich

$$\omega_{task} := \max_{1 \leq i \leq 4} \frac{w^T W_{task} e_i}{w^T W_{eval} e_i}$$

3. Literatur

3. Literatur

[1] Griewank, A.: *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia (2000).

[2] Griewank, A.: *http://www.math.tu-dresden.de/wir/staff/griewank*.

[3] Heun, V.: *Grundlegende Algorithmen*. Vieweg, Wiesbaden (2003).

[4] Königsberger, K.: *Analysis 2*. Springer, Heidelberg (2000).

[5] Nocedal, J.; Wright, S. J.: *Numerical Optimization*. Springer, New York (1999).

Das Handout und diese Präsentationsfolien sind auch unter http://www.mathi.uni-heidelberg.de/~ferreau/ad/ verfügbar

Fragen ?