

高速自動微分法の定式化と計算複雑度の解析†

久保田 光一^{††} 伊 理 正 夫^{††}

高速自動微分法は、数値計算と数式処理の中間に位置し、多変数の関数の正確な勾配を変数の個数によらずに高速に計算し、かつ、関数値に含まれる丸め誤差の評価も行える手法である。しかも、この手法はヘッセ行列とベクトルの積および（ベクトル値関数の）ヤコビ行列とベクトルの積をも変数の個数によらない手間で計算できるので、非線形最適化などの分野で基礎的な技術となるものである。この論文の目的は次の2点にある。第1点は、計算グラフに対してその部分計算グラフという概念を導入し、さらに部分計算グラフのBU拡大とTD拡大という操作を定義することによって、関数の中間変数に関する偏導関数の定義と高速自動微分法の算法の記述とを従来よりも厳密にすることである。第2点は、四則演算といくつかの初等関数を用いて表される関数に対して、関数自身の計算複雑度を基準として高速自動微分法の算法の計算複雑度を詳しく解析し、勾配計算は関数計算の4倍以下の総演算回数で可能であり、またヘッセ行列とベクトルの積の計算は14倍以下の総演算回数で可能であることを示すことである。

1. ま え が き

高速自動微分法 (Fast Automatic Differentiation, 以降 FAD と略す) は “勾配を高速に計算する算法”^{1), 2), 8), 13)} と “偏導関数を自動的に計算する手法”¹²⁾ とを組み合わせたものである。FAD は次の長を持つ。

- i) 高速性: 一多変数関数のすべての変数に関する偏導関数 (勾配) を数値的に計算するのに必要な手間が、関数の値だけを計算するのに必要な手間の高々定数 (変数の数には依存しない) 倍である。
- ii) 自動性: 一関数を計算する “手続き” を与えることにより、勾配の値を計算する “手続き” が機械的に生成できる。
- iii) 高精度: 一数値微分により得られる勾配の値よりも正確な値を計算することができる。その正確さは、数式微分を行って導いた勾配の式を計算して得られる値の正確さと同等である。
- iv) 誤差評価能力: 一関数の計算値に含まれる丸め誤差の良い評価が副産物として得られる。

FAD は次の二つの算法を基本とする。一つは、関数のある1個の変数 (これは任意に指定できる) に関する偏導関数を計算する算法 (§4.3.1 の BU 算法) である。これはよく知られた算法であり、これを利用して関数を表す式を入力文字列として与えると偏導関

数を自動的に計算する手法 (自動微分) も提案されている¹²⁾。しかし、この方法で勾配を計算するのに要する手間は、数値微分法などと同様に、関数計算の手間の “変数の数” 倍くらいとなる。もう一つは、Baur, Iri, Kim らが論じている新しい算法 (§4.3.1 の TD 算法) であり、これによれば、関数計算の手間の “定数倍” の手間で勾配計算ができる。特に、関数が有理式で表現される場合に乗除算の回数で手間を計れば、“定数” は3であることが示されている^{1), 2), 8), 10), 13), 15)}。

FAD は、この二つの算法を、独立にあるいは適宜組み合わせて用いて、関数の勾配やヘッセ行列とベクトルとの積などを高速に計算する、いくつかの算法の総称である。また、FAD は、共役勾配法等への応用可能性も高く、数値計算の基礎的な手法として確立されるべきものである。

本論文の目的は二つある。一つは、Kantorovich に由来すると言われている計算グラフの概念^{2), 7)} を FAD の算法を記述するために利用しながら、新たに “部分計算グラフ” の概念を導入して偏導関数の定義と算法の記述を厳密化することである。もう一つは、有理式で表される関数 (これらについては文献 1), 2) で考察されている) より広い範囲の関数に対して、FAD の算法に必要な手間が、関数だけを計算するのに必要な手間に比べて何倍になるのかを詳しく解析することである。

§2 で用語・記号の定義を行った後、§3 で計算グラフ、部分計算グラフとその拡大の概念を導入する。§4 では FAD の諸算法を記述し、§5 で手間の解析を行う。なお、FAD の〈自動性〉に関してはあらためて別の論文で論じることとし、本論文では触れない⁹⁾。

† Formulation and Analysis of Computational Complexity of Fast Automatic Differentiation by KOICHI KUBOTA and MASAO IRI (Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo).

†† 東京大学工学部計数工学科

2. 基本演算と分解可能関数

関数を計算する過程で使用するこのできる演算を基本演算と呼ぶ。基本演算の集合を $\mathcal{P}$ と記す。 $\mathcal{P}$ はなるべく多くの演算を含んでいる方が取り扱うこのできる関数の範囲が広がるが、計算複雑度の解析の無用の複雑化を避けるため、ここでは $\mathcal{P}$ の要素をいくつかの単項、2項演算および無引数の擬似演算「定数生成（所望の定数の値を与える演算）」、「外部入力（入力変数の値を与える演算）」に限定する。）

簡単のため、本論文では、基本演算の合成の形に表すことができる、いわゆる“分解可能関数”¹²⁾ だけについて論じる。しかし、FORTRAN, PASCAL 等のプログラム言語で記述できる関数であれば、たとえ IF 文等の条件分岐がプログラム中に含まれているものでも、以下の議論はそのまま成立する。（もちろん、導関数を“擬導関数 (quasi-derivative, subdifferential)”で置き換える必要が生じることもある。）なぜならば、そのような関数についても、計算を進行させながら後述の計算グラフを作成できるからである。

単項の基本演算は u_1 を仮引数として $\phi(u_1)$ 、2項の基本演算は u_1 と u_2 を仮引数として $\phi(u_1, u_2)$ のように記す。このとき、 $\phi_1 = \partial\phi/\partial u_1$ 、 $\phi_2 = \partial\phi/\partial u_2$ を要素的偏導関数と呼ぶ。同じ変数 v が2項の基本演算の第1、第2実引数となる場合、つまり $\phi(v, v)$ となる場合にも、形式的に $\partial\phi/\partial u_1$ 、 $\partial\phi/\partial u_2$ は定義されることとする。（例えば $x = v * v$ のとき、 $\partial\phi/\partial u_1$ も $\partial\phi/\partial u_2$ も v であって、 $2v$ ではない。）

入力変数と中間変数という2種類の変数を考える。入力変数とは、それらに外部から値を与えると関数の値が計算されるような変数のことである。中間変数とは、関数の値を計算する過程で、基本演算を実行するごとに生成される値（関数計算の中間結果）を保持するための変数のことである。したがって、中間変数の総数は関数の値を計算するときに実行される基本演算の総実行回数に等しい。スカラ値関数の場合、通常、関数を計算するときに最後に実行される基本演算の値を保持する中間変数がその関数の値を与える。

一つの基本演算を実行してその結果を中間変数に代入する操作を計算ステップと呼ぶ。計算ステップは中間変数 v と1対1に対応し、仮引数を u_1, u_2 とすれば、基本演算が単項演算のときには“ $v \leftarrow \phi(u_1)$ ”，2項演算のときには“ $v \leftarrow \phi(u_1, u_2)$ ”と表現できる。ただし、 u_1, u_2 に対応する実引数は、入力変数、定数、ま

$$\left. \begin{array}{l} v_1 \leftarrow \phi_1(u_{11}, u_{12}) \text{ または } \phi_1(u_{11}) \\ \vdots \\ v_j \leftarrow \phi_j(u_{j1}, u_{j2}) \text{ または } \phi_j(u_{j1}) \\ \vdots \\ v_r \leftarrow \phi_r(u_{r1}, u_{r2}) \text{ または } \phi_r(u_{r1}) \end{array} \right\} \text{ 計算過程}$$

$v_1, \dots, v_r$: 中間変数
 $\phi_1, \dots, \phi_r$: 基本演算
 u_{11}, u_{12} : 入力変数あるいは定数
 u_{j1}, u_{j2} : 入力変数、定数、 $v_1, \dots, v_{j-1}$
 のいずれか ($j=2, \dots, r$)

$$f(x_1, x_2) = \frac{1}{\sqrt{2\pi x_1 x_2}}$$

$$v_1 = x_1 * x_2$$

$$v_2 = 2\pi * v_1$$

$$v_3 = \sqrt{v_2}$$

$$v_4 = 1/v_3$$

図1 計算ステップ、計算過程とその例

Fig. 1 Computational steps and computational process.

たは v を計算する計算ステップより前に実行した計算ステップの中間変数のいずれかである。関数を計算する過程は、この計算ステップの列（計算過程と呼ぶ）で表現される（図1）。

中間変数、入力変数あるいは定数 u, v に対して「 v の値を計算するためには u の値が定まっていることが必要」という関係があるときに

$$u < v \quad (2.1)$$

と記すことにすると、“ $<$ ”は変数、定数の集合の上の一つの半順序となる。計算過程は、この半順序に矛盾しない全順序で並べられた中間変数の列（すなわち計算ステップの列）であるといえる。

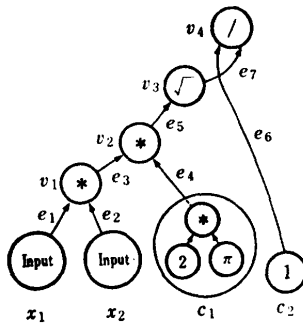
3. 計算グラフ

計算グラフは、関数の計算過程を、変数、定数の間の半順序“ $<$ ”をそのままの形で表現する無閉路有向グラフ (acyclic graph) である。

3.1 計算グラフの定義

計算グラフ $G = (V, E, \delta^+, \delta^-, \omega, n, d)$ とは、頂点集合 V 、枝集合 E 、接続関係 δ^+ 、 δ^- により定義される有向グラフと、その上に定義された三つの関数 ω, n, d からなる複合概念である。ここで、 $V = V_x \cup V_v \cup V_k$ ($V_x \cap V_v = V_v \cap V_k = V_k \cap V_x = \emptyset$) であり、 V_x は入力変数の集合、 V_v は中間変数の集合、 V_k は定数の集合である（定数だけから生成される中間変数、例えば $2 * \pi$ など、はあらかじめその計算をしておいて、一つの定数頂点に変換しておく；なお、 $V_x \cup V_k = \{v \in V | \delta^- v = \emptyset\}$ である）。 $E = E_v \cup E_k$ であり、 E_v は始点が入力変数または中間変数の枝の集合 $\{e \in E | \delta^+ e \in V_x \cup V_v\}$ 、 E_k は始点が定数の枝の集合 $\{e \in E |$

$$f(x_1, x_2) = \frac{1}{\sqrt{2\pi x_1 x_2}}$$



$V_X = \{x_1, x_2\}$;
 $V_U = \{v_1, \dots, v_4\}$;
 $V_K = \{c_1, c_2\}$;
 $E_U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$;
 $E_K = \{e_4, e_6\}$;

$\omega(x_1) = \omega(x_2) = \text{'外部入力'}$, $\omega(c_1) = \omega(c_2) = \text{'定数生成'}$,
 $\omega(v_1) = \text{'*'}$, $\omega(v_2) = \text{'*'}$, $\omega(v_3) = \text{'1/'}$, $\omega(v_4) = \text{'/'}$;
 $n(e_1) = 1$, $n(e_2) = 2$, $n(e_3) = 1$, $n(e_4) = 2$, $n(e_5) = 1$,
 $n(e_6) = 1$, $n(e_7) = 2$;
 $d(e_1) = x_1$, $d(e_2) = x_1$, $d(e_3) = c_1$, $d(e_4) = v_1$,
 $d(e_5) = 1/2v_1$, $d(e_6) = 1/v_1$, $d(e_7) = -v_1/v_3$.

図 2 計算グラフ (簡単のため、変数 (定数) と頂点の名前を同じにした.)

Fig. 2 Computational graph.

$\partial^* e \in V_K$ である. $\omega: V \rightarrow \Psi$ は、頂点 v に基本演算を対応させる関数である. $v \in V_X$ あるいは $v \in V_K$ のとき $\omega(v)$ は入力変数あるいは定数の値を与える無引数の演算であり、 $v \in V_U$ のとき $\omega(v)$ は単項あるいは 2 項の基本演算である. $n: E \rightarrow \mathbf{Z}_+$ ($\mathbf{Z}_+$: 正の整数の集合) は、枝 e が $\omega(\partial^* e)$ の $n(e)$ 番目の仮引数に対応することを表す関数である. $d: E \rightarrow \mathbf{R}$ は $\omega(\partial^* e)$ ($= \phi$) の $n(e)$ 番目の仮引数に関する要素的偏導関数 $\phi_{n(e)}$ の値を枝 e に対応づける関数である (図 2). (頂点 v について基本演算 $\omega(v)$ の仮引数は $\partial^* v = \{e \mid \partial^* e = v\}$, 実引数は $I^* v = \{\partial^* e \mid e \in \partial^* v\}$ であるとみなす.)

(2.1) の半順序 “ $<$ ” は、計算グラフの言葉を使えば、「頂点 u から v へのパスがあるときに $u < v$ 」という頂点の間の半順序として定義される.

計算過程が一つ与えられれば対応する計算グラフが定められるが、一般には、一つの計算グラフには複数の計算過程が対応する. (半順序に矛盾しない全順序が一般には多数存在することに相当する.)

3.2 部分計算グラフ

計算グラフ $G = (V, E, \partial^*, \partial^-, \omega, n, d)$ の部分計算グラフ $H = (W, F, \hat{\partial}^*, \hat{\partial}^-, \hat{\omega}, \hat{n}, \hat{d})$ とは次の諸条件を満たすもののことである.

条件 0: グラフとして H は G の部分グラフである.

条件 1: $V_K \subseteq W \subseteq V$.

条件 2: $F \subseteq E$ かつ、すべての $v \in W$ について

「 $\partial^* v \cap F = \emptyset$ 」または「 $\partial^* v \subseteq F$ 」.

条件 3: $\hat{\partial}^*$, $\hat{\partial}^-$, $\hat{\omega}$, $\hat{n}$, $\hat{d}$ は、それぞれ、 ∂^* , ∂^- , ω , n , d の定義域を F , F , W , F , F に制限したものである.

(以下では単に頂点集合、枝集合だけを用い、 $H = (W, F)$ と記し、 $\hat{\partial}^*$, $\hat{\partial}^-$, $\hat{\omega}$, $\hat{n}$, $\hat{d}$ は陽には書かない. また、 $\delta^\pm$, $I^\pm$ 等は H に関する接続関係を表すことと約束する.)

計算グラフ $G = (V, E)$ の部分計算グラフ $H = (W, F)$ は、以下のようにして一つの計算グラフと解釈できる. すなわち、 H の極小頂点の集合を $M = \{v \in W \mid \partial^* v = \emptyset\}$ とすれば、 H は、 $W_X (= M - V_K)$ を入力変数頂点の集合、 $W_U (= W - M)$ を中間変数頂点の集合、 $W_K (= V_K)$ を定数頂点の集合として、極大頂点 (一般には複数存在する) $u \in \{v \in W \mid \partial^+ v = \emptyset\}$ を W_X から計算する関数の計算グラフである.

部分計算グラフを以上のように厳密に定義する必要があるのは、次節の算法の記述の便のためもあるが、一つには“中間変数の間の偏導関数”の厳密な定義のためでもある. すなわち、 $\partial v / \partial u$ と記されるものが、 v を u のどのような関数とみたときの偏導関数であるかを明示するためである. 部分計算グラフ H (v を極大頂点、 u を極小頂点に持つ) を計算グラフとしてもつ関数については、 v の u に関する偏導関数

$$\left. \frac{\partial v}{\partial u} \right|_H$$

の定義には曖昧性はないので、これにより、もとの計算グラフ G の頂点 u と v の間の偏導関数を定義することができる. 部分計算グラフを用いないと、たとえ

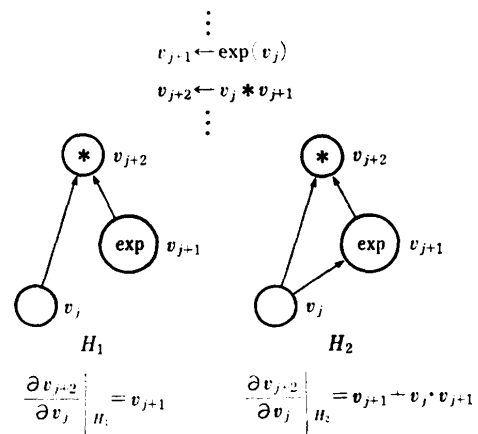


図 3 部分計算グラフと偏導関数

Fig. 3 Different computational subgraphs and partial derivatives.

ば図3での $\partial v_{j+2}/\partial v_j$ は v_{j+1} と $v_{j+1}+v_j \cdot v_{j+1}$ のどちらであるかが曖昧になる。

計算グラフ G の中で、部分計算グラフ $H=(W, F)$ を拡大して新しい部分計算グラフ $\tilde{H}=(\tilde{W}, \tilde{F})$ を作る方法として特別に次の2種類の手続きを考える。(このような拡大を考える理由は、以降の算法の記述を厳密にするためである。)

一つは、 H の頂点を実引数にもつ G の頂点 (計算ステップ) を追加する BU 拡大 (Bottom-Up extension) である。すなわち、 $\tilde{H}=(\tilde{W}, \tilde{F})$ が $H=(W, F)$ の $v \in V-W$ に関する BU 拡大であるとは、 v が条件「 $\Gamma^-v \subseteq W$ かつ $\Gamma^-v \neq \emptyset$ 」を満たして、 $\tilde{W} = W \cup \{v\}$, $\tilde{F} = F \cup \delta^-v$ であることを言う。

他の一つは、 H の入力変数の集合 W_x の中で G における半順序「 $<$ 」に関して極大な中間変数の一つ v に関する TD 拡大 (Top-Down extension) である。すなわち、 $\tilde{H}=(\tilde{W}, \tilde{F})$ が $H=(W, F)$ の $v \in W$ に関する TD 拡大であるとは、 v が条件「 G における半順序に関して $W_x \cap V_v$ の中で極大」を満たして、 $\tilde{W} = W \cup \Gamma^-v$, $\tilde{F} = F \cup \delta^-v$ であることを言う。

これらの拡大によって部分計算グラフの枝数は増加する。拡大のための頂点 v の選び方は一意ではない。

4. 算 法

4.1 関数の偏微分

関数 f_i の入力変数 x_j に関する偏導関数は、通常、合成関数の偏微分則 (いわゆる chain rule) によって、要素的偏導関数の積の和の形で

$$\frac{\partial f_i}{\partial x_j} = \sum \frac{\partial f_i}{\partial v_{k_1}} \cdot \frac{\partial v_{k_1}}{\partial v_{k_2}} \cdots \frac{\partial v_{k_{l-1}}}{\partial v_{k_l}} \cdot \frac{\partial v_{k_l}}{\partial x_j}$$

に書かれている。これは、計算グラフ G の記法を用いると、

$$\frac{\partial f_i}{\partial x_j} = \sum_{p \in P(x_j, f_i)} \left[\prod_{e \in E(p)} d(e) \right]$$

と書くことができる。ここで、 $P(x_j, f_i)$ は G の上の頂点 x_j から頂点 f_i へのパスの集合、 $E(p)$ はパス p を構成する枝の集合を表す。部分計算グラフ H についても、その極大頂点 u の極小頂点 v (入力変数) に関する偏導関数を

$$\left. \frac{\partial u}{\partial v} \right|_H = \sum_{p \in P(v, u)} \left[\prod_{e \in E(p)} d(e) \right]$$

と書くことができる ($\hat{P}$ は H におけるパス)。

4.2 算法の記述

算法中に現れる ∂^+ , ∂^- , ω , n , d は $G=(V, E)$ に

関するものとして、次のことを仮定する。

- i) 関数の値を計算することができる (零割り等が生じない) 点 x での偏導関数などの値を求める。
- ii) 点 x での関数の値の計算が終了して、関数の計算グラフ $G=(V, E)$ の構造が既知である (すなわち $d(e)$ の値の計算が終了していること; ただし、§4.3.2 においてはこの仮定をはずす)。

以降の算法中に現れる記号の意味は次のとおり:

- f_i : m 次元ベクトル値関数 f の第 i 番目の成分の値を持つ中間変数に対応する頂点;
- f : スカラ値関数 f の値を持つ中間変数に対応する頂点;
- v : 計算グラフの頂点;
- x_j : n 次元ベクトル x (入力変数) の第 j 番目の成分に対応する頂点;
- y_i : 外部から与える任意の定数ベクトル y (m 次元または n 次元) の第 i 番目の成分の値。

各種の変数の値を格納するために以下の作業用の場所を頂点 $v \in V_x \cup V_v$ に対して用意する:

- $V(v)$: v に対応する入力変数または中間変数;
- $S(v)$: 勾配計算のための作業変数;
- $B(v), T(v)$: ヘッセ行列とベクトルとの積を計算するための作業変数。

($v \in V_x$ については、 $V(v)$ に v に対応する定数の値を格納しておく。)

4.3 算 法

4.3.1 基本算法

基本的な算法である BU 算法と TD 算法は、本論文で導入した概念を用いると、次のように記述できる。

<BU 算法>

任意に決めたある一つの成分 x_j について、 $\partial f_1/\partial x_j, \dots, \partial f_n/\partial x_j$ を $S(f_1), \dots, S(f_n)$ に作り出す。

1) 初期化

$S(v) \leftarrow 0$ for all $v \in V_x$;

$S(x_j) \leftarrow 1$;

$H \leftarrow (V_x \cup V_x, \emptyset)$ 。

2) 計算

H が BU 拡大できなくなるまで以下を繰り返す:

H を頂点 v に関して BU 拡大して部分計算グラフ $\tilde{H}$ を得ることができるとする;

v について

$$S(v) \leftarrow \sum_{e \in \delta^-v \cap E_v} S(\partial^+e) \cdot d(e); \quad (4.3.1)$$

$$\left[S(v) = \frac{\partial v}{\partial x_j} \right]_{\tilde{H}} \text{ である. }]$$

$$H \leftarrow \tilde{H}.$$

<TD 算法>^{2), 8), 13)}

任意に決めたある一つの成分 f_i について, $\partial f_i / \partial x_1, \dots, \partial f_i / \partial x_n$ を $S(x_1), \dots, S(x_n)$ に作り出す.

1) 初期化

$$S(v) \leftarrow 0 \text{ for all } v \in V_v \cup V_x;$$

$$S(f_i) \leftarrow 1;$$

$$H \leftarrow (\{f_i\} \cup V_k, \phi).$$

2) 計算

H が TD 拡大できなくなるまで以下を繰り返す:

H を頂点 v に関して TD 拡大して部分計算グラフ $\tilde{H}$ を得ることができるとする;

すべての $e \in \delta^+ v \cap E_v$ について

$$S(\partial^+ e) \leftarrow S(\partial^+ e) + S(v) \cdot d(e); \quad (4.3.2)$$

$$\left[w = \partial^+ e \text{ なら } S(w) = \frac{\partial f_i}{\partial w} \right]_{\tilde{H}} \text{ である. }]$$

$$H \leftarrow \tilde{H}.$$

4.3.2 基本算法の変種

前節では, 要素的偏導関数の値 $d(e)$ がすべての $e \in E_v$ について計算済みであると仮定した. しかし, $d(e)$ をあらかじめ計算しておかなくとも, 必要になったときにはじめて $S(\partial^+ e) \cdot d(e)$ や $S(\partial^- e) \cdot d(e)$ という因子の計算をしてもよい. このような改良算法を BU* 算法, TD* 算法と呼ぶ.

<BU* 算法>

[上述のように (4.3.1) 式を計算するほかは BU 算

<BU 算法>

$$v = x/y$$

$$d_1 = 1/y$$

$$d_2 = -v/y$$

$$S(v) = S(x) \cdot d_1 + S(y) \cdot d_2$$

/: 3 回, *: 2 回,

+: 1 回

<TD 算法>

$$v = x/y$$

$$d_1 = 1/y$$

$$d_2 = -v/y$$

$$S(x) = S(x) + S(v) \cdot d_1$$

$$S(y) = S(y) + S(v) \cdot d_2$$

/: 3 回, *: 2 回,

+: 2 回

(a)

<BU* 算法>

$$v = x/y$$

$$S(v) = (S(x) - S(y) \cdot v) / y$$

/: 2 回, *: 1 回,

-: 1 回

<TD* 算法>

$$v = x/y$$

$$tmp = S(v) / y$$

$$S(x) = S(x) + tmp$$

$$S(y) = S(y) - tmp \cdot v$$

/: 2 回, *: 1 回,

+: 1 回, -: 1 回

(b)

図 4 基本算法とその変種 (除算の場合)

(tmp はこの場面だけで参照する一時的な変数)

Fig. 4 Basic algorithms and their variations.

(tmp is a temporary variable referred to in this context alone.)

法に同じ.]

<TD* 算法>

[上述のように (4.3.2) 式を計算するほかは TD 算法に同じ.]

これらが“改良”になることを, 除算の場合を例にして説明する. $d(e)$ を陽に計算した場合には, 図 4 (a) <BU> (<TD>) のように除算 3 回, 乗算 2 回, 加算 1 回 (2 回) を必要とする. 一方, $S(\partial^+ e) \cdot d(e)$ ($S(v) \cdot d(e)$) を直接計算することにすれば, 図 4 (b) <BU*> (<TD*>) のように除算 2 回, 乗算 1 回, 減算 1 回 (および加算 1 回) で済む.

4.3.3 ヤコビ行列の求め方

基本算法から明らかなように, $f(x) \in \mathbb{R}^m$ ($x \in \mathbb{R}^n$) について, BU 算法によれば $\partial f_i / \partial x_j$ ($i=1, \dots, m$) を, TD 算法によれば $\partial f_i / \partial x_j$ ($j=1, \dots, n$) を計算できる. したがって, j を 1 から n まで変えながら合計 n 回 BU 算法を繰り返す, あるいは, i を 1 から m まで変えながら合計 m 回 TD 算法を繰り返すことにより, ヤコビ行列 J の全要素を計算することができる. 通常は, $m \gg n$ ($m \ll n$) のときは BU 算法 (TD 算法) を使用して J を計算すればよい. (厳密には, 計算グラフの形を詳しく調べなければどちらの算法がより有効かは決定できない.)

4.4 ヤコビ行列とベクトルとの積

関数 $f(x) \in \mathbb{R}^m$ ($x \in \mathbb{R}^n$) の点 x_0 におけるヤコビ行列 J と縦ベクトル $y \in \mathbb{R}^m$ との積 Jy を計算するには, パラメタ t を導入して, $g(t) = f(x_0 + ty)$ を t に関して微分すればよい. 変数は t だけであるから, BU 算法を 1 回適用すればベクトル Jy の全成分を得ることができる. そのためには, BU 算法または BU* 算法を次のように変更するだけでよい.

<JBU 算法>, <JBU* 算法>

1) 初期化

$$S(x_j) \leftarrow y_j \quad (j=1, \dots, n);$$

$$H \leftarrow (V_x \cup V_k, \phi).$$

2) 計算

[BU 算法または BU* 算法と同じ]

一方, J と横ベクトル y^T ($y \in \mathbb{R}^m$) との積 $y^T J$ は, f と y との内積 $h(x) = \sum_{i=1}^m y_i \cdot f_i(x)$ の勾配として計算できる. すなわち, $h(x)$ に対して TD 算法または TD* 算法を 1 回適用すればベクトル $y^T J$ の全成分が得られる. そのためには, TD 算法または TD* 算法を次のように変更するだけでよい.

<JTD 算法>, <JTD* 算法>

1) 初期化

$$S(v) = 0 \text{ for all } v \in V_X \cup V_U;$$

$$S(f_i) = y_i \quad (i=1, \dots, m);$$

$$H \leftarrow (\{f_1, \dots, f_m\} \cup V_K, \phi).$$

2) 計算

[TD 算法または TD* 算法と同じ]

4.5 ヘッセ行列とベクトルとの積

スカラー値関数 $f(x)$ ($x \in \mathbf{R}^n$) に TD 算法 (§4.3) を適用すれば $f(x)$ の勾配 $g(x) (= \nabla f(x))$ が計算される。 $g(x)$ は、 x のベクトル値関数であり、その計算過程は $f(x)$ の計算過程および TD 算法により与えられる。したがって、 $f(x)$ の計算過程 (グラフ) から $g(x)$ の計算過程を作り、それにヤコビ行列とベクトルとの積を計算する算法 (§4.4) を適用すれば、 f のヘッセ行列 H とベクトル y との積 Hy のすべての成分

$$(Hy)_i = \left[\sum_{k=1}^n y_k \cdot \frac{\partial^2 f}{\partial x_i \partial x_k} \right] \quad (j=1, \dots, n)$$

を計算することができる。 $g(x)$ の計算過程は、 f の計算過程の後に TD 算法を使用して f の勾配を計算する計算過程を付け加えたものである。しかし、 $g(x)$ の計算過程を作るのに必要十分な情報は $f(x)$ の計算過程に含まれているから、以下に示すように $f(x)$ の計算過程だけから直接 Hy を計算することもできる。(文献 2), 8) 等にもこのような着想は述べられているが、部分計算グラフの拡大という概念を用いるとより一般的な記述が可能となる。)

基本演算 $\phi(u_1), \phi(u_1, u_2)$ の要素的 2 階偏導関数を $\phi_{11}(u_1), \phi_{11}(u_1, u_2), \phi_{12}(u_1, u_2), \phi_{21}(u_1, u_2), \phi_{22}(u_1, u_2)$ と記し、関数 $d2: E \times E \rightarrow \mathbf{R}$ を導入して計算グラフの枝 e, e' に $d2(e, e') = \phi_{ij}$ ($i=n(e), j=n(e')$) を対応させる。(ただし、 $d2$ の定義域は、 $\{(e, e') | \partial^- e = \partial^- e' \in V_U\}$ である。)

<H 算法>

$\partial f / \partial x_1, \dots, \partial f / \partial x_n$ を $S(x_1), \dots, S(x_n)$ に作り出し、 Hy の第 j 成分を $T(x_j)$ に作り出す ($j=1, \dots, n$)。

1) 初期化

$$S(v) \leftarrow 0 \text{ for all } v \in V_X \cup V_U;$$

$$S(f) \leftarrow 1;$$

$$B(x_j) \leftarrow y_j \quad (j=1, \dots, n);$$

$$T(v) \leftarrow 0 \text{ for all } v \in V_X \cup V_U;$$

$$H_1 \leftarrow (\{f\} \cup V_K, \phi);$$

$$H_2 \leftarrow (V_X \cup V_K, \phi);$$

$$H_3 \leftarrow (\{f\} \cup V_K, \phi).$$

2) 計算

H_1 が TD 拡大できなくなるまで以下を繰り返す:

H_1 を頂点 v に関して TD 拡大することによって部分計算グラフ $\tilde{H}_1$ が得られるとする;

すべての $e \in \partial^- v \cap E_U$ について

$$S(\partial^+ e) \leftarrow S(\partial^+ e) + S(v) \cdot d(e); \quad (4.5.1)$$

$$H_1 \leftarrow \tilde{H}_1.$$

H_2 が BU 拡大できなくなるまで以下を繰り返す:

H_2 を頂点 v に関して BU 拡大することによって部分計算グラフ $\tilde{H}_2$ が得られるとする;

$$B(v) \leftarrow \sum_{e \in \partial^- v \cap E_U} B(\partial^+ e) \cdot d(e); \quad (4.5.2)$$

$$H_2 \leftarrow \tilde{H}_2.$$

H_3 が TD 拡大できなくなるまで以下を繰り返す:

H_3 を頂点 v に関して TD 拡大することによって部分計算グラフ $\tilde{H}_3$ が得られるとする;

すべての $e \in \partial^- v \cap E_U$ について

$$\begin{aligned} T(\partial^+ e) &\leftarrow T(\partial^+ e) + T(v) \cdot d(e) \\ &\quad + S(v) \cdot \left\{ \sum_{e' \in \partial^- v \cap E_U} d2(e', e) \cdot B(\partial^+ e') \right\}; \end{aligned} \quad (4.5.3)$$

$$H_3 \leftarrow \tilde{H}_3.$$

H 算法は、BU 算法を 1 回、TD 算法を 2 回含んでいる。これらを、BU*, TD* 算法と同様に、要素的 (2 階) 偏導関数を含む式の計算をまとめて計算の手間を減らした算法を H* 算法と呼ぶ。

4.6 丸め誤差の評価^{2), 3), 11), 14)}

関数 f を計算した値に含まれる丸め誤差の近似値 Δf は、中間変数 v と、 v を計算した時の発生誤差 δv とを用いて

$$\Delta f = \sum_{v \in V_U} \frac{\partial f}{\partial v} \cdot \delta v \quad (4.6.1)$$

と表すことができる。一方、 $\partial f / \partial v$ の値は TD 算法により $S(v)$ に作られている。したがって、 $|\delta v| < |v| \cdot \varepsilon$ (ε : マシンエプシロン) などとすれば (4.6.1) の絶対値の上界などを計算することができる。

5. 算法の計算複雑度

§5.1~§5.3 では、上述の各算法の時間計算複雑度を、その算法を遂行するために必要な基本演算の回数として評価する。領域計算複雑度、すなわち計算に必要な作業領域の広さについては §5.4 で論じる。

5.1 要素的偏導関数の計算

ここでは、5 種類の単項演算と 3 種類の 2 項演算を基本演算とし、これらの要素的偏導関数および要素的

2階偏導関数を計算するために必要な演算回数を以下の①～⑧に示す。なお、符号反転操作の手間は以下では無視する。 e_1, e_2 はそれぞれ $\psi=\omega(v)$ の第1仮引数、第2仮引数を表す枝とし、§4と同様に $V(v)$ により、頂点 $v (= \partial^- e_1 = \partial^- e_2)$ に対応する変数や定数を表す。

①加算/減算 (+, -) :— 符号反転の手間を無視するので、加算と減算は同じ演算とみなされる。 $d(e_1)=1$, $d(e_2)=\pm 1$ で、要素的2階偏導関数はすべて零であるから、これらの計算のための手間は不要である。

②定数倍 ($c*$) :— 見かけ上は2項演算であるが実質的には単項演算である。 $d(e_1)=c$ であり、要素的2階偏導関数は零であるから、これらの計算のための手間は不要である。

③乗算 ($*$) :— $d(e_1)=V(\partial^+ e_2)$, $d(e_2)=V(\partial^+ e_1)$, $d2(e_1, e_1)=d2(e_2, e_2)=0$, $d2(e_1, e_2)=d2(e_2, e_1)=1$ であり、これらの計算のための手間は不要である。

④除算 ($/$) :— $d(e_1)=1/V(\partial^+ e_2)$, $d(e_2)=-V(\partial^+ e_1)/V(\partial^+ e_2)^2$, $d2(e_1, e_2)=d2(e_2, e_1)=-1/V(\partial^+ e_2)^2$, $d2(e_1, e_1)=0$, $d2(e_2, e_2)=2 \cdot V(\partial^+ e_1)/V(\partial^+ e_2)^3$ であるが、これらの計算は

$$\begin{aligned} d(e_1) &= 1/V(\partial^+ e_2), \quad d(e_2) = -d(e_1) \cdot V(v), \\ d2(e_1, e_1) &= 0, \quad d2(e_1, e_2) = -d(e_1) \cdot d(e_1), \\ d2(e_2, e_1) &= d2(e_1, e_2), \\ d2(e_2, e_2) &= -2 \cdot d(e_1, e_2) \cdot V(v) \end{aligned}$$

と実行すればよいから、要素的偏導関数を計算する

のに除算1回と乗算1回、要素的2階偏導関数を計算するには、さらに、乗算2回、定数倍1回が必要である。

⑤指数関数 (exp) :— $d(e_1)=V(v)$, $d2(e_1, e_1)=V(v)$ であり、これらの計算のための手間は不要である。

⑥対数関数 (log) :— $d(e_1)=1/V(\partial^+ e_1)$, $d2(e_1, e_1)=-d(e_1) \cdot d(e_1)$ である。したがって、要素的偏導関数のための手間は除算1回、さらに要素的2階偏導関数の計算のために必要な手間は乗算1回である。

⑦正弦・余弦関数 (sin, cos) :— $V(v)=\sin(V(\partial^+ e_1))$ の場合は、 $d(e_1)=\cos(V(\partial^+ e_1))$, $d2(e_1, e_1)=-V(v)$ である。したがって、要素的偏導関数の計算には余弦関数を1回必要とし、要素的2階偏導関数の計算のための手間は不要である。 $(V(v)=\cos(V(\partial^+ e_1)))$ の場合も同様。

⑧開平 (sqrt) :— $d(e_1)=1/(2 \cdot V(v))$, $d2(e_1, e_1)=-0.5 \cdot d(e_1)/V(\partial^+ e_1)$ である。したがって、要素的偏導関数の計算のために定数倍を1回、除算を1回、さらに要素的2階偏導関数の計算のために定数倍を1回、除算を1回必要とする。

5.2 各算法の時間複雑度

関数の計算過程に現れる基本演算の出現回数を、

§5.1 ①, ..., ⑧の順番に、 $n_A, n_S, n_M, n_D, n_E, n_L, n_T, n_R$ とする ($n_A = |\{v | \omega(v) = '+'\} \cup \{v | \omega(v) = '-'\}|$ など)。 $|V_U| = n_A + n_S + n_M + n_D + n_E + n_L + n_T + n_R$ であ

表1 算法中の式を計算するための時間

(実引数が変数である場合だけについて示す。)

Table 1 The time for computing the expressions in the algorithms for each basic operation.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
$\pm$	0	0	$A+2M$	$2A+2M$	$6A+8M$	$2A$	A	$2A$	A	$2A$
定数倍	0	0	M	$A+M$	$2A+3M$	A	S	$S+A$	S	A
$*$	0	0	$A+2M$	$2A+2M$	$6A+8M$	$4A+4M$	$A+2M$	$2A+2M$	$A+2M$	$4A+4M$
$/$	$M+D$	$S+2M$	$A+2M$	$2A+2M$	$6A+8M$	$5A+7M$	$A+M+D$	$2A+M+D$	$A+M+D$	$4A+3M+D$
exp	0	0	M	$A+M$	$2A+3M$	$2A+2M$	M	$A+M$	M	$2A+2M$
log	D	M	M	$A+M$	$2A+3M$	$2A+3M$	D	$A+D$	D	$2A+M+D$
sin, cos	T	0	M	$A+M$	$2A+3M$	$2A+3M$	$M+T$	$A+M+T$	M	$2A+3M$
sqrt	$S+D$	$S+D$	M	$A+M$	$2A+3M$	$2A+3M$	$S+D$	$A+S+D$	D	$2A+S+M+D$

[1] 要素的偏導関数の計算のための時間

[2] 要素的2階偏導関数の計算のための時間

[3] 式(4.3.1), (4.5.2)の計算のための時間

[4] 式(4.3.2), (4.5.1)の計算のための時間

[5] 式(4.5.3)の計算のための時間

[6] 要素的2階偏導関数が構造的に0または1であることを利用して式(4.5.3)を計算するための時間

[7] 式(4.3.1)を BU* 算法で計算するための時間

[8] 式(4.3.2), (4.5.1)を TD* 算法で計算するための時間

[9] 式(4.5.2)を BU* 算法で計算するための時間

[10] 式(4.5.3)を TD* 算法で計算するための時間

る. 基本演算を1回実行するのに必要な時間を, 上と同じ順番に, A, S, M, D, E, L, T, R とする. 基本演算の種類ごとに, 前述の算法中の式を計算するのに必要な時間を表1に示す. また, 以下では, 基本演算の出現回数 (n_A など) を重みとした表1の列 $[i]$ の成分の加重和を " $[i]$ " と書くことにする ($i=1, \dots, 10$).

計算グラフ G の定義から, 関数の値を計算するのに必要な時間 $\nu(G)$ は

$$\nu(G) = n_A A + n_S S + n_M M + n_D D + n_E E \\ + n_L L + n_T T + n_R R$$

である.

関数の計算と要素的偏導関数の計算も含めた BU 算法の実行時間 $\nu_{BU}(G)$ は,

$$\nu_{BU}(G) \leq \nu(G) + ([1]) + ([3]) \\ \leq (2A + 2M)n_A + (S + M)n_S + (A + 3M)n_M \\ + (A + 3M + 2D)n_D + (M + E)n_E \\ + (M + D + L)n_L + (M + 2T)n_T \\ + (S + M + D + R)n_R$$

であり, 一方 BU* 算法の実行時間 $\nu_{BU*}(G)$ は,

$$\nu_{BU*}(G) \leq \nu(G) + ([7]) \\ \leq 2An_A + 2Sn_S + (A + 3M)n_M \\ + (A + M + 2D)n_D + (M + E)n_E \\ + (D + L)n_L + (M + 2T)n_T \\ + (S + D + R)n_R$$

となる.

関数の計算と要素的偏導関数の計算も含めた TD 算法の実行時間 $\nu_{TD}(G)$ は, $\nu(G) + ([1]) + ([5])$ である. しかし, 算法の中で変数を零に初期化せずに, 最初に零を加える加算 (これは極大でない各 $v \in V_X \cup V_U$ に対して1回ずつある) を消去して図5のように代入文に置き換えれば

$$\nu_{TD}(G) \leq \nu(G) + ([1]) + ([4]) \\ - (|V_U \cup V_X| - 1)A \\ \leq (2A + 2M)n_A + (S + M)n_S + (A + 3M)n_M$$

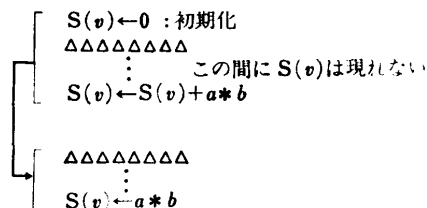


図5 TD 算法, TD* 算法における加算の消去
Fig. 5 Eliminating additions in TD-algorithm and TD*-algorithm.

$$+ (A + 3M + 2D)n_D + (M + E)n_E \\ + (M + D + L)n_L + (M + 2T)n_T \\ + (S + M + D + R)n_R$$

と手間を減らすことができる²⁾.

一方 TD* 算法の実行時間 $\nu_{TD*}(G)$ は, 同様にして, 零への加算を消去すれば,

$$\nu_{TD*}(G) \leq \nu(G) + ([8]) - (|V_U \cup V_X| - 1)A \\ \leq 2An_A + 2Sn_S + (A + 3M)n_M \\ + (A + M + 2D)n_D + (M + E)n_E \\ + (D + L)n_L + (M + 2T)n_T \\ + (S + D + R)n_R$$

となる.

JBU 算法, JBU* 算法は, BU 算法, BU* 算法と同様で,

$$\nu_{JBU}(G) = \nu_{BU}(G), \quad \nu_{JBU*}(G) = \nu_{BU*}(G).$$

JTD 算法, JTD* 算法は, TD 算法, TD* 算法と同様で,

$$\nu_{JTD}(G) = \nu_{TD}(G), \quad \nu_{JTD*}(G) = \nu_{TD*}(G).$$

要素的2階偏導関数を陽に利用するH算法の場合は,

$$\nu_H(G) \leq \nu(G) + ([1]) + ([2]) + ([3]) + ([4]) + ([6]) \\ - 2 \cdot (|V_X \cup V_U| - 1)A \\ \leq (4A + 4M)n_A + (S + 2M)n_S + (5A + 9M)n_M \\ + (6A + S + 14M + 2D)n_D + (A + 4M + E)n_E \\ + (A + 6M + D + L)n_L + (A + 5M + 2T)n_T \\ + (A + 2S + 5M + 2D + R)n_R.$$

前と同様に, 第2辺の最後の項は零を加える加算を消去することによる手間の減少である.

H* 算法については,

$$\nu_{H*}(G) \leq \nu(G) + ([8]) + ([9]) + ([10]) \\ - 2 \cdot (|V_X \cup V_U| - 1)A \\ \leq 4An_A + 3Sn_S + (5A + 9M)n_M \\ + (5A + 5M + 4D)n_D + (A + 4M + E)n_E \\ + (A + M + 3D + L)n_L + (A + 5M + 2T)n_T \\ + (A + 2S + M + 3D + R)n_R.$$

5.3 関数計算の時間複雑度との比較

関数計算のために必要な手間と, FAD により勾配などをも計算するのに必要な手間とを2種の仮定のもとで比較する. ここでは, 関数 $g_1, g_2, \dots$ の計算に必要な時間を $L(g_1, g_2, \dots)$ で表す. 以下では, f はスカラー値関数, g はベクトル値関数を表す.

[仮定 1: $A=S=M=D=E=N=T=R$] すべての基本演算の手間が等価である, すなわち, $A=S=M=D=E=N=T=R=1$ とおく (表2).

表 2 仮定 1 の下での算法の時間複雑度

Table 2 Time complexity under the assumption 1.

仮定 1: $[A=S=M=D=E=N=T=R]$:— すべての基本演算の時間が等価, すなわち, $A=S=M=D=E=N=T=R=1$ とおく.

$$\begin{aligned} \nu(G) &= n_A + n_S + n_M + n_D + n_E + n_L + n_T + n_R \\ \nu_{BU}(G) &\leq 4n_A + 2n_S + 4n_M + 6n_D + 2n_E + 3n_L + 3n_T + 4n_R \\ \nu_{BU^*}(G) &\leq 2n_A + 2n_S + 4n_M + 4n_D + 2n_E + 2n_L + 3n_T + 3n_R \\ \nu_{TD}(G) &\leq 4n_A + 2n_S + 4n_M + 6n_D + 2n_E + 3n_L + 3n_T + 4n_R \\ \nu_{TD^*}(G) &\leq 2n_A + 2n_S + 4n_M + 4n_D + 2n_E + 2n_L + 3n_T + 3n_R \\ \nu_H(G) &\leq 8n_A + 3n_S + 14n_M + 23n_D + 6n_E + 9n_L + 8n_T + 11n_R \\ \nu_{H^*}(G) &\leq 4n_A + 3n_S + 14n_M + 14n_D + 6n_E + 6n_L + 8n_T + 8n_R \end{aligned}$$

表 3 仮定 2 の下での算法の時間複雑度

Table 3 Time complexity under the assumption 2.

仮定 2: $[A=S=0, M=D=E=N=T=R]$:— 加減算と定数倍の手間を無視し, $A=S=0, M=D=E=N=T=R=1$ とおく.

$$\begin{aligned} \nu(G) &= n_M + n_D + n_E + n_L + n_T + n_R \\ \nu_{BU}(G) &\leq 2n_A + n_S + 3n_M + 5n_D + 2n_E + 3n_L + 3n_T + 3n_R \\ \nu_{BU^*}(G) &\leq 3n_M + 3n_D + 2n_E + 2n_L + 3n_T + 2n_R \\ \nu_{TD}(G) &\leq 2n_A + n_S + 3n_M + 5n_D + 2n_E + 3n_L + 3n_T + 3n_R \\ \nu_{TD^*}(G) &\leq 3n_M + 3n_D + 2n_E + 2n_L + 3n_T + 2n_R \\ \nu_H(G) &\leq 4n_A + 2n_S + 9n_M + 16n_D + 5n_E + 8n_L + 7n_T + 8n_R \\ \nu_{H^*}(G) &\leq 9n_M + 9n_D + 5n_E + 5n_L + 7n_T + 5n_R \end{aligned}$$

$$\text{BU 算法 [BU* 算法]: } L(f, \nabla f) \leq 6nL(f) \\ [4nL(f)]$$

$$\text{TD 算法 [TD* 算法]: } L(f, \nabla f) \leq 6L(f) \\ [4L(f)]$$

$$\text{H 算法 [H* 算法]: } L(f, (\nabla^2 f)y) \leq 23L(f) \\ [14L(f)]$$

$$\text{JBU 算法 [JBU* 算法]: } L(g, (\nabla g)y) \leq 6L(g) \\ [4L(g)]$$

$$\text{JTD 算法 [JTD* 算法]: } L(g, y^T(\nabla g)) \leq 6L(g) \\ [4L(g)]$$

[仮定 2: $A=S=0, M=D=E=N=T=R$] 加減算と定数倍の手間を無視し, $A=S=0, M=D=E=N=T=R=1$ とおく (表 3).

$$\text{BU* 算法: } L(f, \nabla f) \leq 3nL(f)$$

$$\text{TD* 算法: } L(f, \nabla f) \leq 3L(f)$$

$$\text{H* 算法: } L(f, (\nabla^2 f)y) \leq 9L(f)$$

$$\text{JBU* 算法: } L(g, (\nabla g)y) \leq 3L(g)$$

$$\text{JTD* 算法: } L(g, y^T(\nabla g)) \leq 3L(g)$$

5.4 領域複雑度

BU 算法と TD 算法とは, $|V_v|$ 個の中間変数 $V(v)$, $|V_x \cup V_v|$ 個の作業変数 $S(v)$, $|E_v|$ 個の要素的偏導関数 $d(e)$ を格納する領域を必要とする. BU* 算法と TD* 算法とは, $|V_v|$ 個の中間変数 $V(v)$, $|V_x \cup V_v|$ 個の作業変数 $S(v)$ を格納する領域を必要

とする. ただし, BU 算法と BU* 算法とにおいて, 値を保持する必要がなくなった作業変数などの再利用により, もっと小さい領域で算法が実行可能な場合もある.

H 算法は $|V_v|$ 個の中間変数 $V(v)$ と, $|V_x \cup V_v|$ 個ずつの作業変数 $S(v)$, $B(v)$, $T(v)$ と, 要素的 (2 階) 偏導関数 $d(e)$, $d_2(e, e')$ を格納する最大 $5 \cdot |E_v|$ 個の領域を必要とする. H* 算法は $|V_v|$ 個の中間変数 $V(v)$ と, $|V_x \cup V_v|$ 個ずつの作業変数 $S(v)$, $B(v)$, $T(v)$ と, $|V_v|$ 個の (同じ因子, 項の再計算を避けるために必要な) 作業領域を必要とする. また, H 算法, H* 算法においては $S(v)$ と $T(v)$ とを共用し, $|V_x \cup V_v|$ 個の領域を減らすこともできる.

6. む す び

高速自動微分法は, 従来の勾配計算, ヤコビ行列計算に代わりうる手法であるばかりでなく, ヘッセ行列計算や丸め誤差評価にも応用できる. したがって, 既存の各種の数値計算の分野に多くの影響を与えることが期待できる. 本論文では部分計算グラフの拡大という新しい概念を導入して, それにより, 従来より一般的で厳密な算法の記述を試み, その基礎の上に立って計算複雑度を詳しく解析・比較した.

高速自動微分法を実用化するには, 手軽に自動微分するためのシステムの開発が必要である. それについては岩田, 伊理^{5), 6)} により一つの方向が示唆されているが, 我々はかなり本格的なプリプロセッサ形式の利用システムを試作し⁴⁾, それによる実験を重ねているので, その結果についても別途報告したい⁹⁾.

謝辞 本研究をまとめるにあたり有益な助言をくださった東京大学工学部計数工学科室田一雄助教授に感謝いたします. なお, 本研究の一部は文部省科学研究費補助金 (一般研究 (B) 60460130: 偏導関数計算と丸め誤差評価のためのプログラム言語と処理系の研究) の援助による.

参 考 文 献

- 1) Baur, W., and Strassen, V.: The Complexity of Partial Derivatives, *Theor. Comput. Sci.*, Vol. 22, pp. 317-330 (1983).
- 2) Iri, M.: Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors—Complexity and Practicality, *Japan Journal of Applied Mathematics*, Vol. 1, No. 2, pp. 223-252 (1984).
- 3) 伊理正夫, 土谷 隆, 星 守: 偏導関数計算

- と丸め誤差推定の自動化の大規模非線形方程式系への応用, 情報処理, Vol. 26, No. 11, pp. 1411-1420 (1985).
- 4) 伊理正夫, 久保田光一: 高速微分法とその応用, 第7回数値計画シンポジウム論文集, pp. 159-184 (1986).
 - 5) 岩田憲和: 偏導関数計算の自動化, 東京大学大学院工学系研究科情報工学専門課程修士論文 (1984. 3).
 - 6) 岩田憲和, 伊理正夫: 多変数関数の勾配の計算方法, 情報処理学会数値解析研究会資料, 7-1 (1983. 12).
 - 7) Kantorovich, L.V.: Ob Odnoi Matematicheskoi Simvolike, Udobnoi pri Provedenii Vychislenii na Mashinakh, *Doklady Akademii Nauk SSSR*, Tom 113, No. 4, pp. 738-741 (1957).
 - 8) Kim, K. V., Nesterov, Yu. E., Skokov, V. A., and Cherkasskii, B. V.: Èffektivnyi Algoritm Vychisleniya Proizvodnykh i Èkstremal'nye Zadachi, *Èkonomika i Matematicheskie Metody*, Tom 20, Vyp. 2, pp. 309-318 (1984).
 - 9) 久保田光一, 伊理正夫: 高速自動微分法のためのプリプロセッサ (投稿予定).
 - 10) Lunin, V. Yu., and Urzhumtsev, A. G.: Program Construction for Macromolecule Atomic Model Refinement Based on the Fast Fourier Transform and Fast Differentiation Algorithms, *Acta Crystallographica*, Vol. A 41, pp. 327-333 (1985).
 - 11) Miller, W., and Wrathall, C.: *Software for Roundoff Analysis of Matrix Algorithms*, Academic Press, New York (1980).
 - 12) Rall, L. B.: *Automatic Differentiation—Techniques and Applications*, Lecture Notes in Computer Science, Vol. 120, Springer-Verlag, Berlin (1981).
 - 13) Sawyer, J. W. Jr.: First Partial Differentiation by Computer with an Application to Categorical Data Analysis, *The American Statistician*, Vol. 38, No. 4, pp. 300-308 (1984).
 - 14) 土谷 隆: 高速微分法および丸め誤差推定法とその応用, 東京大学大学院工学系研究科計数工学専門課程修士論文 (1986. 3).
 - 15) Volin, Yu. M., and Ostrovskii, G. M.: Automatic Computation of Derivatives with the Use of the Multilevel Differentiating Technique—1, Algorithmic Basis, *Computers and Mathematics with Applications*, Vol. 11, No. 11, pp. 1099-1114 (1985).

(昭和 63 年 1 月 29 日受付)

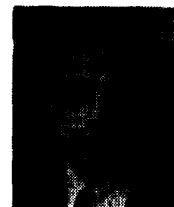
(昭和 63 年 4 月 14 日採録)

久保田光一 (正会員)

1960 年生. 1983 年東京大学工学部計数工学科卒業. 1985 年同大学院修士課程修了. 同年東京大学工学部計数工学科助手. 数値計算の研究に従事. ACM, 日本 OR 学会各会員.



伊理 正夫 (正会員)



昭和 8 年生. 昭和 30 年東京大学工学部応用物理学科 (数理工学専修) 卒業. 昭和 35 年同大学院博士課程修了. 工学博士. 九州大学工学部助手, 助教授 (通信工学科), 東京大学助教授 (工学部計数工学科) を経て, 現在同大教授. 回路, グラフ, 数値計算, 言語などの研究, 教育に従事. 昭和 40 年松永賞受賞. 著書「Network Flow, Transportation and Scheduling」など.