# 偏導関数自動導出システム†

吉田 利信††

偏導関数値を計算する方法として，数値微分法や数式処理による方法のほかに自動微分法が提案されており，関数を記述するだけで正確な偏導関数値が自動的に，かつ，高速に求められることが示されている．したがって，この方法は数値計算の種々の分野に大きな影響を与えうるものであり，手軽に自動微分法を利用するためのシステムの開発が急がれている．この論文の目的は，自動微分法のためのプログラミング言語，および，その言語で記述されたプログラムを処理する偏導関数自動導出システムを示すことにある．ここに示す言語は，ベクトルや行列に対する演算を記述でき，さらに，それらに関する微分演算も記述できる言語である．偏導関数自動導出システムは，プログラムを解釈し式を計算過程に分解する構文解析部，計算過程を計算グラフに表現する計算グラフ生成部，自動微分法に基づき偏導関数の計算過程を導出する微分部，計算グラフから計算過程を生成する出力部から構成されており，出力される計算過程には計算グラフの操作が含まれない．いくつかの問題に対して，これらの問題が簡潔なプログラムに表現されること，および，計算結果の再利用や簡約化を行うため自動微分法自体が生成する計算過程の計算量よりも少ない計算量の計算過程が生成されることが示された．

## 1. はじめに

数値計算の種々の分野において偏導関数値が必要とされ，多くの場合，数値微分法が用いられ，あるいは，手計算や数式処理システムを用いて求めた偏導関数を表す式が用いられてきた．また，偏導関数値を直接必要としない算法も開発されてきた．

これらの方法に対して Wengert[1] は，関数の計算過程を記述するだけで正確な全微分，偏導関数値あるいは高階の偏導関数値が自動的に計算される自動微分法を提案した．この方法は関数の計算過程に沿って，つまり，変数側から関数側へ向かう順で微分を求める方法（BU 法）であるが，関数側から変数側へ向かう順で微分を求める方法（TD 法）が Baur ら[5]，Kim ら[7),8)]，Iri[9]，Sawyer[10] によって独立に示された．Iri は，関数の計算過程を計算グラフに表現することによって非常に見通しよくこの方法を示した．この方法は，スカラ関数の偏導関数値が変数の数によらずに関数計算の手間の定数倍で得られることから，高速自動微分法とも呼ばれている．自動微分法については，文献 1)，9)，18)，19) に詳しく示されているのでここでは説明を省略する．自動微分法を利用する数値計算の算法はすでにいくつか開発されており[14),17)] 手軽に自動微分法を利用するためのシステムの開発が急がれている．

BU 法の自動微分法は，基本演算（加減乗除算，巾乗算および初等関数などの演算）とともにその基本演算に対する全微分，偏微分，高階の微分などを同時に計算するサブルーチンライブラリを用意し，関数を基本演算からなる計算過程に分解し，その計算過程を各基本演算に対するサブルーチンで記述することによって実現される．Wengert[1]，Kalaba ら[6),13)] などは，関数を基本演算からなる計算過程に分解する作業を手で行う方法を示し，既存のプログラミング言語の処理系のみを用いて BU 法を実現する方法を示した．Kalaba ら[16] はベクトルや行列を扱うサブルーチン列によって関数の計算過程を記述する方法も示した．また，Pugh[2]，Kedem[3]，Rall[4] は BU 法を実現するための専用の言語とその処理系を作成した．

TD 法の自動微分法では計算過程を関数側から変数側へ逆にたどることから，その実現方法には，関数の計算時にその計算過程における中間結果を記憶し偏導関数値の計算に利用する動的な方法と，構文解析された関数の計算過程から偏導関数の計算過程を導出する静的な方法とがある．動的な方法では，変数の値によって計算過程が変化する関数に対してもその偏導関数値を求めることができるという特徴がある．静的な方法では，計算結果の再利用，簡約化，高階の微分が可能であり，さらに，計算の実行時に計算過程に関する操作が必要ないという特徴がある．岩田[11]，久保田ら[15] は，関数値を計算する FORTRAN 副プログラムを入力し，関数値および偏導関数値などを計算する FORTRAN 副プログラムを出力する動的な方法に基

† Automatic Derivative Derivation System by TOSHINOBU YOSHIDA (Department of Electrical and Electronic Engineering, Faculty of Engineering, Chiba University).

†† 千葉大学工学部電気電子工学科

づく前処理システムを作成した.

本研究では静的な方法に基づき，自動微分法 (BU 法および TD 法) を実現する偏導関数自動導出システム (Automatic Derivative Derivation System, 以下 ADDS と略す) を開発した．このシステムは，ベクトルや行列に対する演算，および，それらに関する微分演算が記述可能な独自の言語を持ち，偏導関数の効率的な計算過程を導出する.

以下，第2章に ADDS の言語を示し，第3章に ADDS の構造を示す．第4章にプログラム例と計算量の評価を行う.

## 2. 言　　語

### 2.1 型

変数や式などは値を複数持つことができ，その構造を型と呼ぶ．一つの値からなる構造をスカラ型と呼び，[ ]で表す．$k$ 個の添字を持ち，添字の範囲が1からそれぞれ $n_1, n_2, \cdots, n_k$ である構造を型 $[n_1, n_2, \cdots, n_k]$ と表し，$k$ をその型の階数と呼ぶ.

### 2.2 定　　数

定数には整数，単精度実数および倍精度実数のスカラ型定数があり，それぞれに対して計算グラフの節点が生成される.

### 2.3 変数とその宣言

スカラ型変数は宣言せずに用いる．そのほかの変数は次のように宣言してから用いる.

array 〈変数名〉〈型〉, … ;

ただし，代入文の左辺に初めて現れる変数は，右辺の型に自動的に宣言されるので，array 文による宣言は省略できる．変数が宣言されるとその変数のすべての要素に対して計算グラフの節点が生成される．変数 $a$ の型が $[n_1, n_2, \cdots, n_k]$ であるとき，要素全体を $a$ と引用し，添字が $i_1, i_2, \cdots, i_k$ である要素を $a[i_1, i_2, \cdots, i_k]$ と引用する．また，$a[i_1, i_2, \cdots, i_j]$, $1 \leqq j < k$ と引用することによって，型が $[n_{j+1}, n_{j+2}, \cdots, n_k]$ である $k-j$ 階の型の変数として扱うこともできる．たとえば，変数 $a$ が array $a[2, 3]$; と宣言されているとき，$a[1]$ と引用すると $a[1, 1]$ から $a[1, 3]$ までを要素とする型 [3] の値が引用される.

### 2.4 演　算　子

①加算，減算の演算子をそれぞれ＋，－とする．同じ型の式の間で演算が定義され，その型の対応する各要素間の演算に対してそれぞれ計算グラフの節点が生成される.

②乗算の演算子を＊とする．スカラ倍については，スカラでない被演算子の各要素とスカラの被演算子との積に対して，それぞれ計算グラフの節点が生成される．第1被演算子 $a$ が型 $[n_1, n_2, \cdots, n_j, n]$，第2被演算子 $b$ が型 $[n, n_{j+1}, \cdots, n_k]$ のとき，その積は $a$ の最後の添字で表されるベクトルと $b$ の最初の添字で表されるベクトルとの内積を要素として持ち，その内積を求めるための積と和の演算に対して計算グラフの節点が生成される．演算結果の型は $[n_1, n_2, \cdots, n_j, n_{j+1}, \cdots, n_k]$ となる．たとえば，$a$ と $b$ の型がそれぞれ [2, 3, 4], [4, 3] であるとき，その積 $a*b$ の型は [2, 3, 3] である.

③除算の演算子を／とする．第2被演算子がスカラ型のときのみ定義され，第1被演算子の各要素を第2被演算子で割る演算に対して，それぞれ計算グラフの節点が生成される.

④巾乗算の演算子を＊＊とする．スカラのスカラ乗および正方行列の正整数乗についてのみ定義され，その計算過程が計算グラフに生成される.

⑤3次元ベクトルと3次元ベクトルとのベクトル積を求める演算子を vp とする．その計算過程が計算グラフに生成される.

⑥行列の転置を求める演算子を tp とする.

⑦階数をあげる演算子を vec とする．被演算子として $n$ 個の同一の型 $[n_1, n_2, \cdots, n_k]$ の式を持つとき，演算結果の型は $[n, n_1, n_2, \cdots, n_k]$ となる.

⑧スカラ演算子には，sqrt, cbrt, exp, log, log10, sin, cos, tan, cotan, asin, acos, atan, atan2, sinh, cosh, tanh がある．これらの演算に対して計算グラフの節点が生成される.

⑨微分演算子には dfuv, dfu, dfdv および dfd がある．これらの演算子は第1被演算子として微分される式を，第2被演算子として微分する変数を持つ．ここで，微分される式 $f$ の型を $[n_1, n_2, \cdots, n_j]$，微分する変数 $x$ の型を $[n_1', n_2', \cdots, n_k']$ とする．演算子 dfu および dfd はそれぞれ BU 法および TD 法によって偏導関数 $\partial f_{i_1, i_2, \cdots, i_j} / \partial x_{i_1', i_2', \cdots, i_k'}$ の計算過程を計算グラフに生成する．これらの演算結果の型は $[n_1, n_2, \cdots, n_j, n_1', n_2', \cdots, n_k']$ となる．演算子 dfuv はさらに変数 $x$ と同じ型 $[n_1', n_2', \cdots, n_k']$ の式 $y$ を第3被微分演算子として持ち，BU 法によって全微分

$$\sum_{i_1'=1}^{n_1'} \sum_{i_2'=1}^{n_2'} \cdots \sum_{i_k'=1}^{n_k'} \frac{\partial f_{i_1, i_2, \cdots, i_j}}{\partial x_{i_1', i_2', \cdots, i_k'}} \cdot y_{i_1', i_2', \cdots, i_k'}$$

の演算過程を計算グラフに生成し，結果の型は関数 $f$

と同じ型 $[n_1, n_2, \cdots, n_j]$ となる．また，演算子 dfdv は関数 $f$ と同じ型 $[n_1, n_2, \cdots, n_j]$ の式 $y$ を第3被微分演算子として持ち，TD 法によって

$$\sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\cdots\sum_{i_j=1}^{n_j} y_{i_1,i_2,\cdots,i_j}\cdot\frac{\partial f_{i_1,i_2,\cdots,i_j}}{\partial x_{i_1',i_2',\cdots,i_k'}}$$

の演算過程を計算グラフに生成し，結果の型は変数 $x$ と同じ型 $[n_1', n_2', \cdots, n_k']$ となる．

たとえば，array $f$[2], $x$[3], $y$[3], $z$[2]; と宣言されているとき，微分演算 dfu($f, x$) および dfd($f, x$) の結果の型は [2, 3]，dfuv($f, x, y$) の結果の型は [2]，dfdv($f, x, z$) の結果の型は [3] になる．

⑩丸め誤差を推定する演算子を err とする．伊理ら[12]によると，関数 $f$ の計算過程 $v_1, v_2, \cdots, v_k$ で生じる丸め誤差は次のように評価できる．

$$\left((1/3)\cdot\sum_{i=1}^{k}(\partial f/\partial v_i\cdot v_i)^2\right)^{1/2}\varepsilon.$$

ここで，$\varepsilon$ はマシンイプシロンと呼ばれ，浮動小数点演算において $(1+\varepsilon)\neq 1$ となるような最小の数である．演算子 err はこの式に基づいて丸め誤差を評価する計算過程を計算グラフに生成する．演算子 err は，第1被演算子として誤差評価の対象となる式を，第2被演算子としてその式における独立変数を，第3被演算子として $\varepsilon$ を表すスカラ変数を持つ．演算結果の型は第1被演算子の型と同一である．

⑪上記以外の演算は

operator 〈演算子名〉〈型〉(〈被演算子数〉), … ;

と宣言することによって演算子として式中に記述できる．ここで〈型〉は演算結果の型を表し，スカラ型のときは省略できる．また〈被演算子数〉はその演算子の被演算子の要素の数を表す．たとえば，

operator f[2, 5](4) ; array $x$[3];

と演算子 f と変数 $x$ が定義されているとき，スカラ変数 $t$ と変数 $x$ との合計の要素数が4であることから f($t, x$) と引用することができる．変数の場合と同様に f($t, x$) と引用するとその型は [2, 5] となり，f[1]($t, x$) と引用するとその型は [5] となる．

演算結果の型が $[n_1, n_2, \cdots, n_k]$，被演算子の要素数が $n$ の演算子 f が宣言されているとき，dfu または dfd で微分すると，被演算子の要素数が f と同じ $n$，演算結果の型が $[n_1, n_2, \cdots, n_k, n]$ の演算子で，f の演算子名の先頭に d をつけた演算子名を持つ演算子が自動的に宣言される．また，dfuv で微分すると被演算子の要素数が $2n$，演算結果の型が f と同じ $[n_1, n_2, \cdots, n_k]$ の演算子で，f の演算子名の先頭に u をつけた演算子名を持つ演算子が自動的に宣言される．同様に，dfdv で微分すると被演算子の要素数が $n+m$，演算結果の型が $[n]$ の演算子で，f の演算子名の先頭に v をつけた演算子名を持つ演算子が自動的に宣言される．ここで，$m$ は f の演算結果の全要素数 $n_1 n_2 \cdots n_k$ である．たとえば，演算子 h が operator h[2](3) ; と定義されているとき，各微分演算によって自動的に宣言される演算子とそれらの値の型および引数の個数は次のようになる．

| 微分演算 | 演算子名 | 値の型 | 引数の個数 |
|---|---|---|---|
| dfu, dfd | dh | [2, 3] | 3 |
| dfuv | uh | [2] | 6 |
| dfdv | vh | [3] | 5 |

### 2.5 文

プログラムは文を ; で区切って記述する．行中の % から行末までは注釈を表す．

① comment 〈 ; を含まない任意の文字列〉 ;

文字列は注釈を表す．

② clear ;

その時点での計算グラフを消去し，初期状態にする．

③ free ;

その時点の計算グラフにおいて，ほかの節点から参照されない節点を削除する．

④ in "〈ファイル名〉" ;

ADDS の起動直後，入力は標準入力から1文ずつなされる．この文の実行後，入力は指定されたファイルからなされる．ファイル中に in 文を含んでもよい．ファイルからの入力を終了後，入力はこの文の次の文から再開される．

⑤ out "〈ファイル名〉" ;

ADDS の起動直後，出力は標準出力に対してなされる．この文の実行後，出力は指定されたファイルに対してなされる．out 文で切り替えながらいくつかのファイルに出力することができる．

⑥ shut "〈ファイル名〉" ;

指定されたファイルへの出力を終了する．この文の実行後，出力は標準出力に対してなされる．

⑦ on echo ;

入力を標準エラー出力にエコーする．

off echo ;

入力を標準エラー出力にエコーしない．

on fort ;

FORTRAN 形式で出力する．

off fort;

入力と同じ形式で出力する.

⑧ write "〈文字列〉";

文字列を出力する.

⑨ outcode($x_1, x_2, \cdots, x_m$)$f_1, f_2, \cdots, f_n$;

入力変数を $x_1, x_2, \cdots, x_m$ としたときの出力変数 $f_1, f_2, \cdots, f_n$ の計算過程を出力する.

⑩ count($x_1, x_2, \cdots, x_m$)$f_1, f_2, \cdots, f_n$;

入力変数を $x_1, x_2, \cdots, x_m$ としたときの出力変数 $f_1, f_2, \cdots, f_n$ の計算過程に含まれる演算子数を注釈として出力する.

⑪ array 文, operator 文については それぞれ 2.3 節, 2.4 節⑪に示した.

⑫ $a := e$;

変数 $a$ の型が宣言されているとき, その型は式 $e$ の型と一致しなければならない. 変数 $a$ の型が宣言されていないとき, 変数 $a$ の型は式 $e$ の型として自動的に宣言される. 式 $e$ の各要素の値は変数 $a$ の対応する要素に代入される.

⑬ $a[i_1, i_2, \cdots, i_j] := e$;

ここで $a$ は型 $[n_1, n_2, \cdots, n_k]$ を持つ変数名, $e$ は型 $[n_{j+1}, n_{j+2}, \cdots, n_k]$ を持つ式, $j$ は $1 \leqq j \leqq k$ を満たすものとする. 式 $e$ の各要素の値は変数 $a$ の対応する要素に代入される.

## 3. システムの構成

ADDS は Common Lisp で記述され, **図 1** に示すように構文解析部, 計算グラフ生成部, 微分部および出力部から構成されている.

### 3.1 構文解析部

構文解析部は, プログラムを解釈し, 式を計算過程に分解し, 計算グラフ生成部, 微分部および出力部を操作する.

たとえば次の ADDS の実行例 (例 1) では, 構文解析部は, 1 文字ずつプログラムを読み込み, 定数, 変数, 演算子を認識しながら f の計算グラフ (**図 2**) を表す内部表現 (**表 1**) を計算グラフ生成部を用いて生成する. また, outcode 文を認識し, 出力部を用いて f の計算過程を出力する. (ADDS では, 英小文字は英大文字に変換される.)

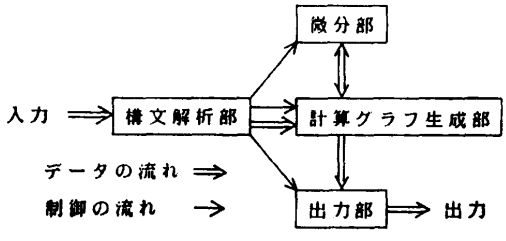


**図 1** ADDS の構造
Fig. 1 The structure of ADDS.

例 1

```
adds〉 f := (a+1)*exp(x)+(a-1)/(exp(x)+1);
adds〉 outcode(x)f;
V 0003 := A+1;
V 0008 := A-1;
V 0005 := EXP(X);
```

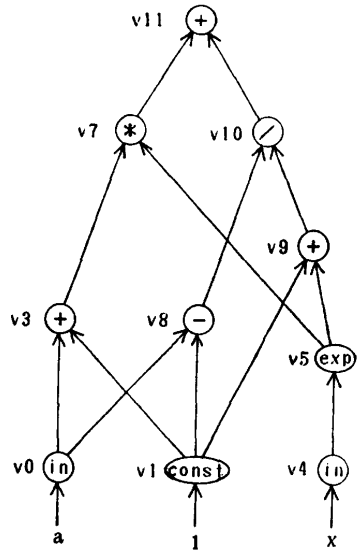


**図 2** 計算グラフ
Fig. 2 Computational graph.

**表 1** 計算グラフの内部表現
Table 1 Internal representation of computational graph.

| 節点 | op | args | to |
|---|---|---|---|
| V0000 | IN | A | (V0008 V0003 A) |
| V0001 | CONST | 1 | (V0009 V0008 V0003) |
| V0002 | CONST | 0 | () |
| V0003 | + | (V0000 V0001) | (V0007) |
| V0004 | IN | X | (V0005 X) |
| V0005 | EXP | V0004 | (V0009 V0007) |
| V0006 | CONST | −1 | () |
| V0007 | * | (V0003 V0005) | (V0011) |
| V0008 | − | (V0000 V0001) | (V0010) |
| V0009 | + | (V0001 V0005) | (V0010) |
| V0010 | / | (V0008 V0009) | (V0011) |
| V0011 | + | (V0007 V0010) | (F) |

F : = V 0003*V 0005+V 0008/(1+V 0005);

プログラム中の定数に対して，その値を表す節点を計算グラフ生成部を用いて探索あるいは生成する．節点が生成されたときは Lisp の大域変数 *numlist* に定数と節点との対を蓄える．例1では *numlist* に

((−1 . V 0006)(0 . V 0002)(1 . V 0001))

が蓄えられる．

変数はその名前の Lisp のアトムで表し，そのアトムの性質 type に変数の型を，性質 value に変数の値を表す計算グラフの節点あるいは節点の組を蓄える．ただし，スカラ型[ ]の変数の性質 type には T を蓄える．例1では，変数 $a, x$ および $f$ は性質 type に T を，性質 value にそれぞれ V 0000, V 0004 および V 0011 を蓄える．

式中の演算はベクトルや行列などに対する演算も含めて，四則演算や初等関数などのスカラの基本演算に分解する．また，式中の微分演算は微分部によって偏導関数を計算する基本演算に分解する．その際，

$x+0\rightarrow x,\ x*1\rightarrow x,\ x+(-y)\rightarrow x-y,\ x/x\rightarrow 1$

などの簡約化を行う．計算グラフ生成部によってこれらの基本演算を表す節点を探索あるいは生成する．

### 3.2 計算グラフ生成部

計算グラフの節点は，Lisp のアトム V 0000, V 0001, …で表す．これらの節点は，性質 op, args, to, diff にそれぞれ演算子，被演算子，その節点を被演算子とする節点のリスト，および，微分部の作業過程における中間結果を蓄える（表1）．

計算グラフ生成部は，次のように節点を探索あるいは生成する．

①定数に対しては大域変数 *numlist* を検索してその定数に対応する節点を求める．存在しないときは節点を生成し，性質 op に const，性質 args にその定数を蓄え，定数と節点の対を *numlist* に蓄える．

②変数に対してはその性質 value からその値を表す計算グラフ上の節点を求める．性質 value が定義されていないとき，その変数をスカラの入力変数と解釈し，節点を生成し，その節点の性質 op に in，性質 args にその変数を蓄える．また，その変数の性質 value にその節点を蓄える．

③基本演算に対しては以下に示す算法1によって，与えられた演算を表す節点がすでに存在しているか探索する．存在していないときは節点を生成し，性質 op にその演算子を蓄える．単項演算子に対してはその被演算子として与えられた節点を，多項演算子に対しては与えられた節点のリストを性質 args に蓄える．ただし，可換な加算と乗算については，被演算子の節点を生成の順に並べ性質 args に蓄える．また，生成した節点を被演算子の節点の性質 to に追加する．

算法1（節点の探索）

演算子を $\phi$，被演算子の節点を $u_1, u_2, \cdots, u_l$ とする．

①節点 $u_1$ から出る辺の終点の節点のリスト $v_1, v_2, \cdots, v_k$ を性質 to から得る．

②各節点 $v_i$ について③を行う．もし，すべての節点について③を行ったときは，この演算を行う節点は存在しないことが示され，終了する．

③節点 $v_i$ の性質 op からその節点の演算子を得る．それが $\phi$ と一致すれば④を行う．一致しなければ②へ戻る．

④節点 $v_i$ の性質 args が被演算子 $u_1, u_2, \cdots, u_l$ と一致すれば，この演算がすでに節点 $v_i$ で計算されていることが発見され，終了する．一致しなければ②へ戻る．

たとえば，表1において V 0008 まで生成してあるとき V 0001+V 0005 に対して，演算子が +，被演算子が V 0001 と V 0005（節点の生成順）である節点を探索する．この時点での V 0001 の性質 to から V 0008 と V 0003 を得る．V 0008 の性質 op は−で一致しない．V 0003 の性質 op は+で一致するが，性質 args は (V 0000 V 0001) で (V 0001 V 0005) とは異なり V 0001+V 0005 が存在しないことがわかる．

### 3.3 微 分 部

微分演算子の第1被演算子によって指定される被微分式の値を表す節点を $f_1, f_2, \cdots, f_n$ とし，第2被演算子によって指定される微分変数を表す節点を $x_1, x_2, \cdots, x_m$ とする．微分演算子 dfuv および dfdv の第3被演算子によって指定される節点をそれぞれ $y_1, y_2, \cdots, y_m$ および $y_1, y_2, \cdots, y_n$ とする．被演算子の型や演算結果の型については，構文解析部が管理する．

微分演算子 dfuv に対しては，次の算法2を用いて $f_1, f_2, \cdots, f_n$ に接続する部分グラフを抽出し，BU 法を用いて全微分を行う．

微分演算子 dfu に対しては，1から $m$ までの各 $i$ について，$y_{i'}=0,\ i'\neq i,\ y_i=1$ となるように $y_1, y_2, \cdots, y_m$ を定め dfuv を $m$ 回繰り返す．

微分演算子 dfdv に対しては，次の算法3を用いて

$x_1, x_2, \cdots, x_m$ に接続する部分グラフを抽出し，TD 法を用いて $\sum_j y_j \partial f_j/\partial x_j$ を求める．

微分演算子 dfd に対しては，1から $n$ までの各 $j$ について，$y_{j'}=0$, $j' \neq j$, $y_j=1$ となるように $y_1, y_2, \cdots, y_n$ を定め dfdv を $n$ 回繰り返す．

算法2 (mark down)

①節点 $f_1, f_2, \cdots, f_n$ のそれぞれの性質 diff にマーク0を蓄える．

②節点の生成とは逆の順序ですべての節点について以下を行う．性質 to のリスト中にマーク0の付けられた節点があるとき，その節点の性質 diff にマーク0を蓄える．

算法3 (mark up)

①節点 $x_1, x_2, \cdots, x_m$ のそれぞれの性質 diff にマークTを蓄える．

②節点の生成の順序ですべての節点について以下を行う．性質 args のリスト中にマークTの付けられた節点が[illegible]の節点の性質 diff にマークTが蓄える．

### 3.4 出力部

偏導関数値のみ必要で関数値は不要である場合など，計算グラフに表されている計算過程のすべてについて数値計算をする必要がない場合がある．また，計算過程が入力変数とパラメータを含み，かつ，繰り返し演算が行われる場合は，パラメータ間の演算を分離しておくと効率がよい．そこで，outcode 文によって指定される入力変数に対応する節点 $x_1, x_2, \cdots, x_m$ と出力変数に対応する節点 $f_1, f_2, \cdots, f_n$ に対して，次のように計算過程を導出する．

①算法2 (mark down) を用いて，出力変数に関係する部分計算グラフを抽出する．

②算法3 (mark up) を用いて，この部分計算グラフをさらに入力変数に関係する部分計算グラフとパラメータのみに関係する部分計算グラフとに分離する．

③パラメータ間の演算に関する計算過程をさきに出力し，次に独立変数に関係する演算の計算過程を出力する．このとき，それぞれの部分計算グラフにおいて1回のみ用いられる演算は代入文として生成せず，その演算結果を用いる演算に組み込んで出力する．

例1において，V0003 と V0008 は1回のみ用いられているが，独立変数 $x$ に依存しないためこれらの計算過程がさきに出力されている．また，V0005 は2回用いられるので代入文として生成されているが，他の節点は1回のみ用いられているので，これらの演算は代入文 F:= … に組み込まれて出力されている．

## 4. プログラム例と計算量の評価

次の例2に示すプログラムは，4変数のスカラ関数 $f$ に対して，dfd($f, x$) で $f$ の勾配ベクトル $g$ を求め，dfuv($g, x, p$) で $f$ のヘッセ行列とベクトル $p$ との積のベクトル $Hp$ を求め，さらに，ベクトル $p$ とベクトル $Hp$ との内積 $p^THp$ を求めるプログラムである．その出力を付録に示す．ADDS によって $f$ の計算過程として，加算7個，乗算が 12 個からなる計算過程が得られ，$f$ と $g$ と $p^THp$ の計算過程として，$f$ の約3倍の加算 22 個，乗算 34 個からなる計算過程が得られた．ADDS は計算結果の再利用や簡約化によって計算量を削減するが，この例の dfd においては再利用や簡約化をしない場合に比べ 56% に，dfuv においては 95% に計算量が削減された．しかし，さらに簡約化の余地があり，この例においては，加算 18 個，乗算 30 個にすることができる．

例2

```
on fort;
on echo;
out "powell. out";
x := vec(x 1, x 2, x 3, x 4);
p := vec(p 1, p 2, p 3, p 4);
f := (x 1+10*x 2)**2+5*(x 3-x 4)**2
  +(x 2-2*x 3)**4+10*(x 1-x 4)**4;
g := dfd(f, x);
php := p*dfuv(g, x, p);
outcode(p)f, g, php;
shut "powell. out";
```

**表2**に，関数の計算量と関数および偏導関数の計算量をいくつかの微分演算例について示す．ここで $b$ 欄には，計算結果の再利用や簡約化を行わずに自動微分法の算法どおりに計算する場合の計算量を示した．これに比べ，$c$ 欄に示したように ADDS は計算量の少ない計算過程を出力している．$c/b$ 欄に示すように関数によって計算量の削減の程度は大きく異なるが，これらの例では平均して約 77% に計算量が削減された．

## 5. おわりに

自動微分法を手軽に利用するための言語とその処理系 ADDS を Common Lisp を用いて作成した．この言語は行列などの演算やそれらの微分演算を式中に含むことができ，記述性，操作性がよい．この言語で

表 2 微分演算例における計算量
Table 2 Number of computational steps in some examples.

| op | args | $a$ | $b$ | $c$ | $b/a$ | $c/a$ | $c/b$ |
|---|---|---|---|---|---|---|---|
| dfu | [ ][2] | 8 | 45 | 24 | 5.63 | 3.00 | 0.533 |
| | [ ][2] | 12 | 32 | 29 | 2.67 | 2.42 | 0.906 |
| | [ ][4] | 19 | 70 | 37 | 3.68 | 1.95 | 0.529 |
| | [4][4] | 28 | 90 | 59 | 3.21 | 2.11 | 0.656 |
| | [2][5] | 209 | 1,272 | 982 | 6.09 | 4.70 | 0.772 |
| | [5][5] | 17 | 106 | 67 | 6.26 | 3.94 | 0.632 |
| dfuv | [ ][2][2] | 21 | 65 | 58 | 3.10 | 2.76 | 0.892 |
| | [ ][4][4] | 17 | 38 | 36 | 2.24 | 2.12 | 0.947 |
| | [ ][10][10] | 112 | 232 | 187 | 2.07 | 1.67 | 0.806 |
| | [4][4][4] | 28 | 69 | 60 | 2.46 | 2.14 | 0.870 |
| | [5][5][5] | 20 | 61 | 60 | 3.05 | 3.00 | 0.984 |
| | [5][5][5] | 252 | 738 | 436 | 2.93 | 1.73 | 0.591 |
| dfd | [ ][2] | 8 | 38 | 28 | 4.75 | 3.50 | 0.737 |
| | [ ][4] | 29 | 74 | 66 | 2.55 | 2.28 | 0.892 |
| | [ ][10] | 66 | 142 | 121 | 2.15 | 1.83 | 0.852 |
| | [2][2] | 28 | 138 | 89 | 4.93 | 3.18 | 0.645 |
| | [2][5] | 209 | 655 | 596 | 3.13 | 2.85 | 0.910 |
| | [4][4] | 20 | 68 | 38 | 3.40 | 1.90 | 0.559 |
| dfdv | [4][3][4] | 12 | 40 | 38 | 3.33 | 3.17 | 0.950 |
| | [4][4][4] | 20 | 50 | 40 | 2.50 | 2.00 | 0.800 |

op: 微分演算子, args: 被微分演算子の型, $a$: 関数の計算量, $b$: 関数および偏導関数の計算量 (自動微分法), $c$: 関数および偏導関数の計算量 (ADDS), $c/b$ の平均値は 0.773.

書かれたプログラムを ADDS は構文解析し，計算グラフを作成し，計算結果の再利用を行い簡約化を行う．そのため，多くの実験例において，計算結果の再利用や簡約化を行わない場合と比べ計算量の少ない計算過程が得られた．また，これらの計算過程は計算グラフの操作を含んでいないため，数値計算を高速に行うことができた．

このシステムは Common Lisp で書かれているため，移植性はよい．このプログラミング言語を拡張し，IF 文，FOR 文，WHILE 文などが記述できるようにすることが今後の課題である．

**謝辞** 本研究をまとめるにあたり，有益なご助言をいただいた千葉大学工学部星守助教授と小野令美講師に深謝いたします．また，日頃，ご指導いただく千葉大学工学部電気電子基礎講座および情報処理工学講座の皆様に感謝いたします．

## 参 考 文 献

1) Wengert, R. E.: A Simple Automatic Derivative Evaluation Program, *Comm. ACM*, Vol. 7, No. 8, pp. 463-464 (1964).
2) Pugh, R. E.: A Language for Nonlinear Programming Problems, *Math. Program.*, Vol. 2, No. 2, pp. 176-206 (1972).
3) Kedem, G.: Automatic Differentiation of Computer Programs, *ACM Trans. Math. Softw.*, Vol. 6, No. 2, pp. 150-165 (1980).
4) Rall, L. B.: *Automatic Differentiation: Techniques and Applications, Lecture Notes in Computer Science*, Vol. 120, Springer-Verlag, Berlin (1981).
5) Baur, W. and Strassen, V.: The Complexity of Partial Derivatives, *Theor. Comput. Sci.*, Vol. 22, pp. 317-330 (1983).
6) Kalaba, R., Rasakhoo, N. and Tishler, A.: Nonlinear Least Squares via Automatic Derivative Evaluation, *Appl. Math. Comput.*, Vol. 12, pp. 119-137 (1983).
7) Kim, K. V., Nesterov, Yu. E., Skokov, V. A. and Cherkasskii, B. V.: An Efficient Algorithm for Computing Derivatives and Extremal Problems, *Ekonomika i matematicheskie metody*, Vol. 20, No. 2, pp. 309-318 (1984).
8) Kim, K. V., Nesterov, Yu. E. and Cherkasskii, B. V.: An Estimate of the Effort in Computing the Gradient, *Soviet Math. Dokl.*, Vol. 29, No. 2, pp. 384-387 (1984).
9) Iri, M.: Simultaneous Computation of Functions Partial Derivatives and Estimates of Rounding Errors—Complexity and Practicality, *Jpn. J. Appl. Math.*, Vol. 1, No. 2, pp. 223-252 (1984).
10) Sawyer, J. W.: First Partial Differentiation by Computer with an Application to Categorical Data Analysis, *American Statistician*, Vol. 38, No. 4, pp. 300-308 (1984).
11) 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).
12) 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).
13) Kalaba, R. and Tesfatsion, L.: Automatic Differentiation of Functions of Derivatives, *Comput. Math. Appl.*, Vol. 12 A, No. 11, pp. 1091-1103 (1986).
14) Kagiwada, H., Kalaba, R., Rasakhoo, N. and Spingarn, K.: *Numerical Derivatives and Nonlinear Analysis*, Plenum Press, New York (1986).
15) 久保田光一，伊理正夫：高速自動微分法の定式化の試みと利用のためのシステム，統計数理研究所昭和 61 年度共同研究報告書，Vol. 61-共会-14, pp. 154-163 (1987).
16) Kalaba, R., Plum, T. and Tesfatsion, L.:

Automation of Nested Matrix and Derivative Operations, *Appl. Math. Comput.*, Vol. 23, pp. 243-268 (1987).
17) 小野令美，戸田英雄，伊理正夫：微分係数を用いた埋込み型 Runge-Kutta 系2段公式について，情報処理学会論文誌，Vol. 28, No. 8, pp. 807-814 (1987).
18) 吉田利信：グラフの変形を用いた偏導関数の計算過程の導出，情報処理学会論文誌，Vol. 28, No. 11, pp. 1112-1120 (1987).
19) 久保田光一，伊理正夫：高速自動微分法の定式化と計算複雑度の解析，情報処理学会論文誌，Vol. 29, No. 6, pp. 551-560 (1988).

## 付録　例2で出力された計算過程

```
V 0013=X 1+X 2*10
V 0017=X 3-X 4
V 0022=X 2-X 3*2
V 0061=V 0022*V 0022
V 0036=V 0022*V 0061
V 0026=X 1-X 4
V 0065=V 0026*V 0026
V 0031=V 0026*V 0065
V 0034=V 0031*40
V 0037=4*V 0036
V 0044=10*V 0017
V 0049=V 0013*2
V 0062=3*V 0061
V 0066=3*V 0065
F=V 0013*V 0013+5*V 0017*V 0017
&  +V 0022*V 0036+10*V 0026*V 0031
G(1)=V 0034+V 0049
G(2)=V 0037+10*V 0049
G(3)=V 0044-2*V 0037
G(4)=-(V 0034+V 0044)
V 0063=(P 2-P 3*2)*V 0062
V 0068=40*(P 1-P 4)*V 0066
V 0072=10*(P 3-P 4)
V 0077=2*(P 1+P 2*10)
PHP=P 1*(V 0068+V 0077)
&      +P 2*(4*V 0063+10*V 0077)
&      +P 3*(V 0072-V 0063*8)
&      -P 4*(V 0068+V 0072)
```

(昭和63年11月 8 日受付)
(平成元年 4 月 11 日採録)

**吉田　利信**（正会員）

1951年生．1978年東京大学大学院博士課程修了（計数工学）．工学博士．千葉大学工学部電気電子工学科助手．言語理解システム，神経細胞の応答特性，自動微分システムの研究に従事．電子情報通信学会，日本音響学会各会員．