高速自動微分法に基づく総和・総乗演算式の偏導関数計算プログラム作成手法の開発

野中久典† 小林康弘† 田村正義††

グラフ理論的な考え方により偏導関数を計算する FAD (Fast Automatic Differentiation) に基づいて総和・総乗関数の偏導関数を効率的に計算する Fortran プログラムの生成手法を開発した．非線形計画法の分野では，多変数関数の導関数を効率的に計算するプログラムが必要とされている．従来の FAD により総和・総乗関数の偏導関数を計算するためには，まず，これらの関数を，繰返し構造を持たない関数に展開し，それからその関数に対して FAD の算法を実行する必要があった．このため，生成された偏導関数計算プログラムは，ベクトル化困難である．今回，総和・総乗演算子を，FAD の処理の中で陽に取り扱うための処理手続きを開発したことにより，もとの関数の繰返し構造を反映した偏導関数の計算プログラムを生成することが可能となった．開発手法により生成されたプログラムのサイズは小さく，並列処理に適している．また，開発手法が総和・総乗演算式が多重に組み合わさった演算式に対しても適用可能であることを示した．数値実験によると，開発手法で生成されたプログラムは，ベクトル化された場合に，従来法により生成されたプログラムのほぼ 100 倍の速さで，すべての入力変数に対する総和・総乗関数の偏導関数の値を計算できることを確認した．

A Generation Method for a Program Computing Derivatives of Summation/Production Functions Based on Fast Automatic Differentiation

Hisanori Nonaka,† Yasuhiro Kobayashi† and Masayoshi Tamura††

An automated generation method for a Fortran program which efficiently computes derivatives of summation/production functions has been developed on the basis of FAD (Fast Automatic Differentiation), a graph theoretical technique for automatic differentiation. It is indispensable for multi-variate nonlinear programming problems to generate efficient programs which calculate partial derivatives of functions. A conventional FAD approach requires two steps to differentiate summation/production functions; transformation of the functions into expressions with no repetitive structure, and application of FAD to the expressions. It is hard to vectorize a program with an unstructured derivative expression obtained in two steps. In the proposed method for program generation, the summation/production operators are explicitly treated in the FAD processing, and the repetitive structures of the function are maintained in their derivative expression. Therefore, this method assures a compact and parallel-processing-oriented program. The method can also be applied to functions with a hierarchical summation/production structure. From results of numerical experiments with vectorized programs, it was found that programs generated by the proposed method can calculate partial derivatives of large-size summation/production functions about 100 times as fast as those generated by the conventional approach.

1. はじめに

高速自動微分法 (Fast Automatic Differentiation: 以下，FAD と略記する) は，数値微分法や数式微分法にかわる新しい高速・高精度偏導関数計算手法として，Wengert[1], Iri[2),3)], Rall[4], Griewank[5] らによって研究されてきた．その基本的な考え方は，①関数の計算プロセスを，計算グラフと呼ばれる非巡回グラフで表現し，②このグラフをたどる過程において偏導関数の式や値を求める，というものである．FAD の算法は，まず計算グラフのたどり方に基づいて，BU 算法 (Bottom-Up 算法) および TD 算法 (Top-Down 算法) に大別される．これらのうち，FAD の基本算法とされる TD 算法によると，すべての入力変数に関する偏導関数の正確な値を，関数自身を計算する高々定数倍の手間で計算することができる．

† (株)日立製作所エネルギー研究所
Energy Research Laboratory, Hitachi Ltd.
†† (株)日立製作所ソフトウェア開発本部
Software Development Center, Hitachi Ltd.

TD 算法はさらにそのインプリメント形式に基づいて，動的実現法と静的実現法とに分けられる[6)~9)]．動的実現法では，計算グラフは関数値の計算と並行して作成される．これに対し，静的実現法では，計算に先立って関数を構文解析することによって作成される．

今回，TD 算法を静的実現法でインプリメントした FAD を用いる偏導関数作成機能を開発した．本機能は，記号処理言語 Lisp で記述されており，非線形最適化プログラム用の入力データ作成支援の一機能として，目的関数や制約関数の偏導関数を計算する Fortran プログラムを自動作成することを目的としている．

静的実現法は，動的実現法と比較して計算時間が短くて済むという利点がある一方で，計算グラフを陽に記憶する必要があることから，メモリを多く必要とするという問題点を持つ．特に，非線形最適化問題に多く登場する．総和演算式や総乗演算式などの繰返し構造を含む関数に対する計算グラフの規模は，繰返しの数に比例して大きくなるため，繰返しの数が大きい場合，静的実現法を用いた FAD の実行に要するメモリ量や計算時間は膨大なものになってしまう．

本報では，このような問題点を解決する，FAD に基づく総和・総乗演算式の偏導関数作成手法を提案する．

2. 高速自動微分法の概要

以下では，まず計算グラフを定義し，次にこれを用いて FAD の原理を説明する．

2.1 計算グラフ

計算グラフとは，関数の計算プロセスを，四則演算や初等関数などの単項または二項の演算からなる基本的な計算ステップの組合せによって表現した非巡回グラフである．例えば，関数 F の I 番目の計算ステップは，

$$V_I=\phi_I(U_{I1}) \text{ または}$$
$$V_I=\phi_I(U_{I1}, U_{I2}). \qquad (2.1)$$

V_I：中間変数

ϕ_I：単項または二項演算

U_{IJ}：中間変数，入力変数，定数のいずれか

と表すことができる．ここで中間変数 V_I を頂点とし，これを頂点 U_{IJ} と枝で結ぶことによって，関数の計算過程を一つのグラフで表すことができる．このグラフを計算グラフ（または Kantorovich グラフ）と呼ぶ．

2.2 FAD の原理

上記の計算グラフにおいて，頂点 V_I と頂点 U_{IJ} とを結ぶ枝に，要素的偏導関数 $\partial V_I/\partial U_{IJ}$ の式または値を対応付けて記憶する．関数 F の入力変数 X_m に関する偏導関数は，計算グラフ上で頂点 X_m から頂点 F へ至るパス上にある要素的偏導関数の積を，頂点 X_m から頂点 F へのあらゆるパスについて加え合わせたものである．

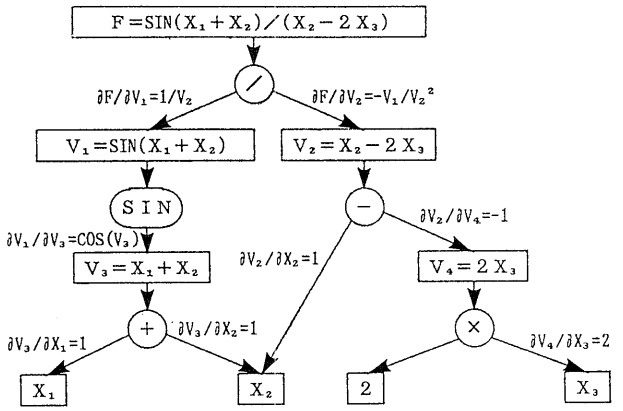
図 1 に FAD による偏導関数作成の具体的な例として，

$$F=\mathrm{SIN}(X_1+X_2)/(X_2-2X_3), \qquad (2.2)$$
$$X_1, X_2, X_3: \text{入力変数}.$$

なる関数に対応する計算グラフ，およびこの計算グラフを用いて求めた偏導関数を示す．

前章でも触れたように，FAD は，要素的偏導関数の積を作成する際の計算グラフのたどり方によって，BU 算法と TD 算法とに分類できる．

BU 算法では，**図 2**（a）に示すように，ある一つの入力変数（X_n）に着目して，計算グラフを入力変数側から関数側に向かってたどり，その過程で要素的偏導



$F=\mathrm{SIN}(X_1+X_2)/(X_2-2X_3)$

$\partial F/\partial X_1=1\cdot\cos(V_3)\cdot 1/V_2=\cos(X_1+X_2)/(X_2-2X_3)$

$\partial F/\partial X_2=1\cdot\cos(V_3)\cdot 1/V_2+1\cdot(-V_1/V_2^2)=\cos(X_1+X_2)/(X_2-2X_3)-\sin(X_1+X_2)/(X_2-2X_3)^2$

$\partial F/\partial X_3=2\cdot(-1)\cdot(-V_1/V_2^2)=2\cdot\sin(X_1+X_2)/(X_2-2X_3)^2$

図 1 FAD による偏導関数の作成
Fig. 1 Generation of derivatives based on FAD.

関数の積を計算して，最終的に $\partial F/\partial X_n$ を求める．すなわち，実際の関数計算と並行して，着目した入力変数に関する偏導関数を計算できる．このため，計算グラフを陽に記憶する必要がなく，メモリ量の点で有利である．一方，すべての入力変数に関する偏導関数の値を計算するためには，以上の処理をすべての入力変数について実行する必要があり，計算の手間は，入力変数の数に比例して増える．

TD 算法は，BU 算法とは逆に，図2(b)のように要素的偏導関数を対応付けた計算グラフをあらかじめ作成しておき，これを関数側から入力変数側に向かってたどる過程で要素的偏導関数の積を求める．この算法によると，すべての入力変数に関する偏導関数を，計算グラフの枝の数に比例した手間，すなわち，関数自身を計算する高々定数倍の手間で計算できる．ただし，計算グラフを陽に記憶しておかなくてはならないため，BU 算法よりも多くのメモリ量を要する．TD 算法はその偏導関数計算の高速性から FAD の基本算法とされる．

TD 算法はさらに，そのインプリメント形式によって，動的実現法と静的実現法とに分類できる．

動的実現法では，①与えられた関数の計算の実行と並行して，要素的偏導関数の値を頂点を結ぶ枝に対応付けた計算グラフを作成する，②作成した計算グラフ上で FAD の算法を実行して偏導関数の値を計算する，という処理を行う．

静的実現法は，①与えられた関数を構文解析することによって，要素的偏導関数の式を頂点を結ぶ枝に対応付けた計算グラフを作成する，②作成された計算グラフ上で FAD の算法を実行して偏導関数の式（あるいは偏導関数の値を計算するプログラム）を生成する，という処理から成る．

動的実現法は関数の表現に対して柔軟であり，条件分岐を含むような関数に対しても，偏導関数の値を計算することができるが，計算のたびに計算グラフを作り直す必要がある．これに対して静的実現法は計算過程が固定した関数にしか適用できないが，FAD の算法の実行に先立って計算グラフを作成しておくため，動的実現法を用いる場合よりも高速に偏導関数の値を計算できるという特徴を持つ．

本研究による偏導関数作成機能の適用対象である非線形最適化計算においては，目的関数や制約関数の偏導関数値の計算は求解の過程において繰返し行われる．このため，偏導関数値の計算の高速化に対する要求は強い．このような理由から，本研究では FAD の実現法として静的実現法を採用した．

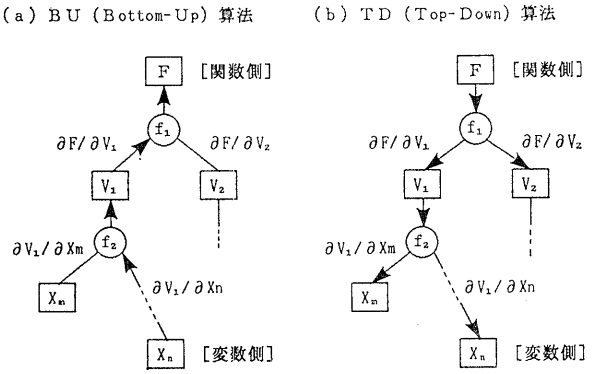


図 2 BU 算法と TD 算法における計算グラフ
Fig. 2 Computational graph for BU-method and TD-method.

2.3 FAD のアルゴリズム

図 3 に本機能で用いる FAD の基本的な処理アルゴリズムを示す．本アルゴリズムへの入力は，Lisp のリストを用いて表現した関数 F の計算グラフである．このアルゴリズムによると，Lisp 関数 "FAD" は，自分自身を再帰的に呼び出して実行することによって，与えられた計算グラフを**図 4** のように関数側から入力変数側に向かって深さ優先でたどり，その過程において要素的偏導関数の式の積 (form) を作成して行く．計算グラフ上での探索が入力変数 X_m に達したならば，form を，入力変数 X_m に対応する記憶領域 $\partial F/\partial X_m$ に記憶されている式に加える．計算グラフのすべての枝をたどり終えた時点で処理を終了する．このとき $\partial F/\partial X_m$ には，F の X_m に関する偏導関数の式が記憶されている．

3. 総和・総乗演算式の偏導関数作成手法

3.1 総和・総乗演算式の偏導関数作成

前章で，FAD の TD 算法は，計算グラフを陽に記憶するためにメモリを多く必要とすることを述べた．特に，本研究で採用した TD 算法の静的実現法では，要素的偏導関数を式の形で記憶するため，要素的偏導関数を数値で記憶する TD 算法の動的実現法の場合よりもさらに多くのメモリを必要とする．

この静的実現法の問題点は，非線形最適化問題にしばしば登場する総和演算式や総乗演算式の偏導関数を作成する際に顕著となる．

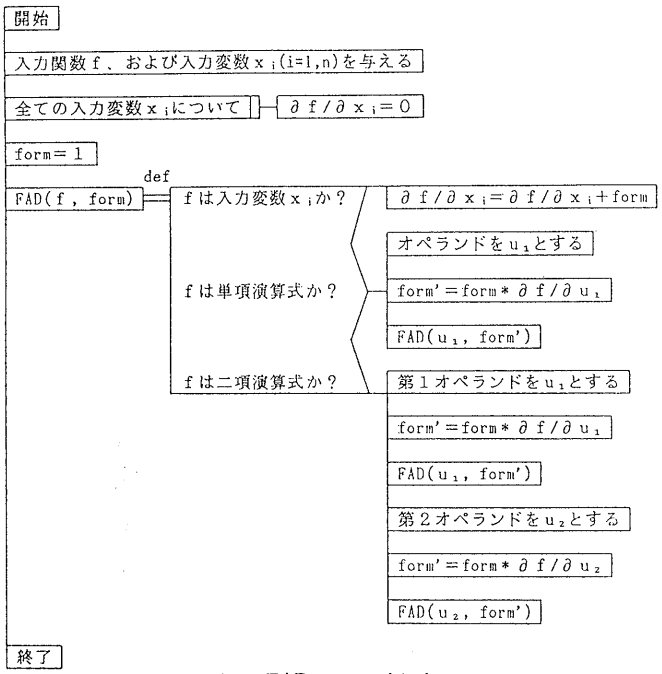


図 3 FAD のアルゴリズム
Fig. 3 Algorithm of FAD.

ここまでに説明した計算グラフでは，総和・総乗演算式を直接表現できない．総和・総乗演算式の偏導関数作成に静的実現法を用いる FAD を適用するためには，計算グラフの作成に先立ち，演算式を和または積からなる単純な算術式に展開する必要がある．例えば，

$$\sum_{I=1}^{5} X_I. \tag{3.1}$$

という総和演算式は，まずこれを，

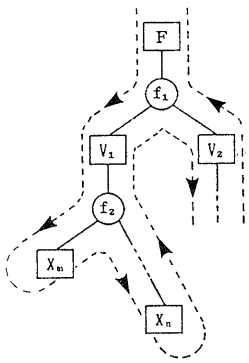


図 4 計算グラフ上での偏導関数の作成
Fig. 4 Generation of derivatives on computational graph.

$$X_1+X_2+X_3+X_4+X_5. \tag{3.2}$$

という和からなる算術式に展開する．次に，この算術式に対して静的実現法を用いる FAD を適用し，偏導関数を作成する．ここでは，このような総和・総乗演算式の偏導関数作成手法を，展開微分法と呼称する．

展開微分法によると，総和・総乗演算式およびこれらの偏導関数に対応する計算グラフの規模は，繰返しの数に比例して大きくなり，繰返しの数が大きい場合にメモリ量，偏導関数の式の記述量の上で問題となる．また，総和・総乗演算式を (3.2) 式のように展開すると，繰返し構造に関する情報は失われてしまう．このため，展開微分法に基づいて作成される偏導関数計算プログラムも繰返し構造を持たず，ベクトル化困難である．繰返しの数が大きい場合には，計算時間は膨大となる可能性がある．

以下では，このような問題点を解決した総和・総乗演算式の偏導関数作成手法について述べる．本手法は，総和・総乗演算式を和または積の形に展開することなく，すべての入力変数に関する偏導関数の値を計算する Fortran プログラムを直接作成する．本手法の基本的な考え方は，総和・総乗演算子を陽に表した計算グラフを想定し，このグラフ上で FAD の算法を実行するというものである．作成される偏導関数計算プログラムは，もとの式の繰返し構造を反映した構造を持っており，ベクトル化可能である．ここではこの手法を総和・総乗演算式の非展開微分法と呼称する．

3.2 総和演算式の非展開微分法

総和演算式に対する非展開微分法の処理を，次の演算式を用いて説明する．

$$F=F(G)=F\left(\sum_{I=I\,\mathrm{MIN}}^{I\,\mathrm{MAX}} g_I(X_m)\right). \tag{3.3}$$

m：入力変数の添字

(i) Fを，**図 5** に示す計算グラフで表現する．このグラフ上で，総和演算子 (Σ) は陽に表されている．また，総和演算子に対する要素的偏導関数を，

$$\frac{\partial G}{\partial g_I}=1. \tag{3.4}$$

とする.

(ii) Fを始点として計算グラフを深さ優先でたどり，要素的偏導関数の積を求めて行く，総和演算式Gに到達した時点で，**図6**(a)に示すように，ここまでに作成した$\partial F/\partial G$の式を変数$V01$に代入するFortranのステートメントを生成する.

(iii) 図6(b)に示すように，Gの繰返し条件に基づいてDOループを生成する．ここでの繰返し条件は，Iを繰返し変数として，$IMIN$から$IMAX$まで増分1で増加させるというものである.

(iv) g_Iを始点として計算グラフをたどり，要素的偏導関数の積を求めて行く．入力変数X_mに到達した時点で，図6(c)に示すように，ここまでに作成した$\partial g_I/\partial X_m$の式を変数$V02$に代入するFortranのステートメントと，$V01$と$V02$との積を$F$の$X_m$に関する偏導関数値の記憶場所DFDX($m$)に足し込むステートメントを生成し，$G$のDOループ内部に挿入する.

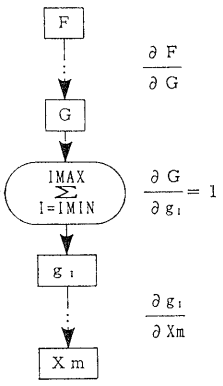


図5 総和演算式の計算グラフ
Fig. 5 Computational graph for summation function.

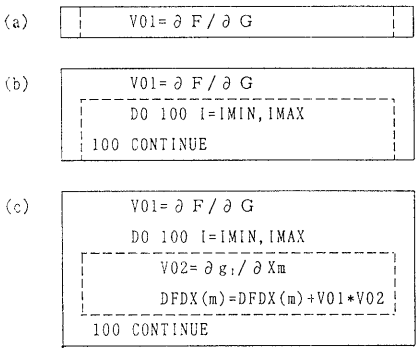


図6 総和演算式の偏導関数計算プログラムの作成過程
Fig. 6 Generation of differentiation program for summation function.

以上で説明した非展開微分法を，

$$F=\sum_{I=1}^{8} X_{I+1}\cdot \mathrm{SIN}(X_I). \tag{3.5}$$

に適用した結果作成された偏導関数計算プログラムの主要部分を**図7**に示す.

図7のプログラムから，すべての入力変数に関する偏導関数の値を計算するために要する手間が，総和演算式そのものを計算する手間の高々定数倍であることがわかる.

3.3 総乗演算式の非展開微分法

総乗演算式に対する非展開微分法の処理を，次の演算式を用いて説明する.

$$F=F(G)=F\left(\prod_{I=I\,\mathrm{MIN}}^{I\,\mathrm{MAX}} g_I(X_m)\right). \tag{3.6}$$

総乗演算子に対する要素的偏導関数を，

$$\frac{\partial G}{\partial g_I}=\prod_{J\neq I} g_J. \tag{3.7}$$

とする．単純に考えると，

$$\frac{\partial G}{\partial g_I}=\frac{G}{g_I}. \tag{3.8}$$

であるが，(3.8)式では$g_I=0$の場合にゼロ割りが発生してしまう．この問題点を解決するために，ここでは$\partial G/\partial g_I$は次のような手順で計算する.

(i) I($IMIN\leqq I\leqq IMAX$なる整数)を引き数とする$G1_I$, $G2_I$なる二つの配列について，以下の初期設定を行う.

$$G1_{I\,\mathrm{MIN}}=1.$$
$$G2_{I\,\mathrm{MAX}}=1. \tag{3.9}$$

```
      V01=1
      DO 100 I=1,8
         V02=SIN(X(I))
         DFDX(I+1)=DFDX(I+1)+V01*V02
         V03=X(I+1)*COS(X(I))
         DFDX(I)=DFDX(I)+V01*V03
  100 CONTINUE
```

図7 (3.5)式の関数Fの偏導関数計算プログラムの主要部分
Fig. 7 Main part of differentiation program for function F in (3.5).

(ii) I を $IMIN+1$ から $IMAX$ まで1ずつ増加させながら以下の処理を行う．

$$J=IMAX+IMIN-I,$$
$$G1_I=G1_{I-1}\times g_{I-1}, \qquad (3.10)$$
$$G2_J=G2_{J+1}\times g_{J+1}.$$

(iii) I を $IMIN$ から $IMAX$ まで1ずつ増加させながら以下の処理を行う．

$$\frac{\partial G}{\partial g_I}=G1_I\times G2_I. \qquad (3.11)$$

以上の処理によると，G の値を計算するおよそ3倍の手間ですべての I に対して $\partial G/\partial g_I$ の値を計算することができる．

以下に，総乗演算式の非展開微分法の処理手順を示す．ここで，第3ステップは，$\partial G/\partial g_I$ を計算するための処理である．

(i) F を，**図8** に示す計算グラフで表現する．

(ii) F を始点として計算グラフをたどり，要素的偏導関数の積を求めて行く．総乗演算式 G に到達した時点で，**図9**（a）に示すように，ここまでに作成した $\partial F/\partial G$ の式を変数 $V01$ に代入する Fortran のステートメントを生成する．

(iii) 図9（b）に示すように，配列 $G1(I)$ および $G2(I)$ を計算するプログラムを作成する．

(iv) 図9（c）に示すように，G の繰返し条件に基づいて DO ループを生成する．ここでの繰返し条件は，I を繰返し変数として，$IMIN$ から $IMAX$ まで増分1で増加させるというものである．

(v) g_I を始点として計算グラフをたどり，要素的偏導関数の積を求めて行く．入力変数 X_m に到達した時点で，図9（d）に示すように，ここまでに作成した $\partial g_I/\partial X_m$ の式を変数 $V02$ に代入す

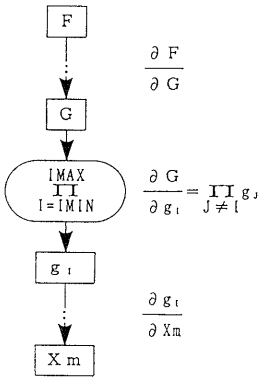


図8 総乗演算式の計算グラフ
Fig. 8 Computational graph for production function.

(a)

```
      V01=∂F/∂G
```

(b)

```
      V01=∂F/∂G
      G1(IMIN)=1
      G2(IMAX)=1
      DO 100 I=IMIN+1,IMAX
         J=IMAX+IMIN-I
         G1(I)=G1(I-1)*g(I-1)
         G2(J)=G2(J+1)*g(J+1)
  100 CONTINUE
```

(c)

```
      V01=∂F/∂G
      G1(IMIN)=1
      G2(IMAX)=1
      DO 100 I=IMIN+1,IMAX
         J=IMAX+IMIN-I
         G1(I)=G1(I-1)*g(I-1)
         G2(J)=G2(J+1)*g(J+1)
  100 CONTINUE
      DO 200 I=IMIN,IMAX
  200 CONTINUE
```

(d)

```
      V01=∂F/∂G
      G1(IMIN)=1
      G2(IMAX)=1
      DO 100 I=IMIN+1,IMAX
         J=IMAX+IMIN-I
         G1(I)=G1(I-1)*g(I-1)
         G2(J)=G2(J+1)*g(J+1)
  100 CONTINUE
      DO 200 I=IMIN,IMAX
         V02=∂g(I)/∂Xm
         DFDX(m)=DFDX(m)+V01*G1(I)*G2(I)*V02
  200 CONTINUE
```

[ ]：各ステップで新たに生成した部分

図9 総乗演算式の偏導関数計算プログラムの作成過程
Fig. 9 Generation of differentiation program for production function.

る Fortran のステートメントと，$V01$, $G1(I)$, $G2(I)$, および $V02$ との積を F の X_m に関する偏導関数値の記憶場所 DFDX(m) に足し込むステートメントを生成し，G の DO ループ内部に挿入する．

以上で説明した非展開微分法を，

$$F=\prod_{I=1}^{8} X_{I+1}\cdot \mathrm{SIN}(X_I). \qquad (3.12)$$

に適用した結果生成された偏導関数計算プログラムの主要部分を**図 10** に示す．

総乗演算式においても，すべての入力変数に関する偏導関数の値を計算するために要する手間は，総乗演算式そのものを計算する手間の高々定数倍である．

3.4 多重の総和・総乗演算式に対する非展開微分法

前節の総和演算式の関数 g_I の中に総乗演算式が存在するような多重の総和・総乗演算式の偏導関数計算プログラムは，基本的には非展開微分法の処理を繰り返し実行することにより作成できる．

ここでは次のような演算式 F を考える．

$$F=F(G),$$
$$G=\sum_{I=I\,\mathrm{MIN}}^{I\,\mathrm{MAX}} g_I(H), \qquad (3.13)$$
$$H=\prod_{J=J\,\mathrm{MIN}}^{J\,\mathrm{MAX}} h_J(X_m).$$

F の偏導関数計算プログラムの作成プロセスは，以下のとおりである．

```
      V01=1
      G1(1)=1
      G2(8)=1
      DO 100 I=2,8
         J=9-I
         G1(I)=G1(I-1)*X(I)*SIN(X(I-1))
         G2(J)=G2(J+1)*X(J+2)*SIN(X(J+1))
  100 CONTINUE
      DO 200 I=1,8
         V02=SIN(X(I))
         DFDX(I+1)=DFDX(I+1)+V01*G1(I)*G2(I)*V02
         V03=X(I+1)*COS(X(I))
         DFDX(I)=DFDX(I)+V01*G1(I)*G2(I)*V03
  200 CONTINUE
```

図 10　(3.12)式の関数 F の偏導関数計算プログラムの主要部分
Fig. 10　Main part of differentiation program for function F in (3.12).

(i) F を，**図 11** に示す計算グラフで表現する．

(ii) F を始点として計算グラフをたどり，要素的偏導関数の積を求めて行く，総和演算式 G に到達した時点で，**図 12** (a)に示すように，これまでに作成した $\partial F/\partial G$ の式を変数 $V01$ に代入する Fortran のステートメントを生成する．

(iii) 図 12 (b)に示すように，G の繰返し条件に基づいて DO ループを生成する．ここでの繰返し条件は，I を繰返し変数として，IMIN から IMAX まで増分 1 で増加させるというものである．

(iv) $g_I(H)$ を始点として計算グラフをたどり，要素的偏導関数の積を求めて行く．総乗演算式 H に到達した時点で，図 12 (c)に示すように，ここまでに作成した $\partial g_I/\partial H$ の式を $V02$ に代入する Fortran のステートメント，および配列 $H1(J)$ および $H2(J)$ を計算するプログラムを作成する．

(v) 図 12 (d)に示すように，H の繰返し条件に基づいて DO ループを生成する．ここでの繰返し条件は，J を繰返し変数として，JMIN から JMAX まで増分 1 で増加させるというものである．また，$h_J(X_m)$ を始点として計算グラフを

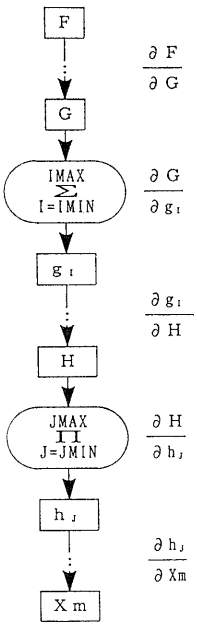


図 11　多重の総和・総乗演算式の計算グラフ
Fig. 11　Computational graph for multiple summation/production function.

たどり，要素的偏導関数の積を求めて行く．入力変数 X_m に到達した時点で，ここまでに作成した $\partial h_J/\partial X_m$ の式を変数 $V03$ に代入する Fortran のステートメントを作成する．さらに $V01$, $V02$, $V03$, $H1(J)$, および $H2(J)$ との積を F の X_m に関する偏導関数値の記憶場所 DFDX(m) に足し込むステートメントを生成し，H の DO ループ内部に挿入する．

図 13 に，

$$F=\sum_{I=1}^{5}\left[X_I\cdot\prod_{J=1}^{5}(X_I-X_J{**}2)\right]. \quad (3.14)$$

なる関数に非展開微分法を適用した結果生成された偏導関数計算プログラムの主要部分を示す．

4. 非展開微分法の評価

以下では，展開微分法，および非展開微分法を用いて作成した総和演算式の偏導関数計算プログラムにおいて，特にすべての入力変数に関する偏導関数値を計算するのに要する時間を評価する．評価に用いる演算式は以下の総和演算式である．

$$F1=\sum_{I=1}^{IMAX}(X_{I+1}\cdot\mathrm{SIN}(X_I)). \quad (4.1)$$

$$F2=\sum_{I=1}^{IMAX}(100\cdot(X_{I+1}-X_I{**}2){**}2+(1-X_I){**}2). \quad (4.2)$$

$F1$ は (3.5) 式の総和演算式において，繰返し回数の上限を $IMAX$ とおいたものであり，$F2$ は，代表的な非線形関数として知られる Rosenbrock 関数に対する総和演算式である．

非展開微分法を用いて求めた $F1$ の偏導関数計算プログラムの主要部分を**図 14** に示す．展開微分法による偏導関数計算プログラムはここでは示さないが，FAD を用いて求めた DFDX(1) から DFDX($IMAX$+1) までの $IMAX$+1 個の偏導関数の式を単純に列挙した形の計算プログラムとなる．このため，$IMAX$ に比例してプログラムの記述量は大きくなる．

図 15 に $IMAX$ を 1 から 10000 まで変化させた場合の $F1$ のすべての入力変数に関する偏導関数を計算するのに要した計算時間のグラフを示す．使用した計算機は HITAC-S 820 である．

展開微分法による偏導関数計算プログラムは繰返し構造を含んでいないため，一般にはベクトル化できない．これに対して，非展開微分法による偏導関数計算プログラムは，単純な繰返し構造を有しており，ベクトル化可能である．図 15 において，非展開微分法はスカラ計算とベクトル計算とのそれぞれの場合について計算時間を示している．以下では記述の簡単のため，それぞれの偏導関数計算プログラムに以下のような名前を付ける．

(a)
```
      V01=∂F/∂G
```

(b)
```
      V01=∂F/∂G
      DO 100 I=IMIN,IMAX
100 CONTINUE
```

(c)
```
      V01=∂F/∂G
      DO 100 I=IMIN,IMAX
         V02=∂g_I/∂H
         H1(JMIN)=1
         H2(JMAX)=1
         DO 200 J=JMIN+1,JMAX
            K=JMAX+JMIN-J
            H1(J)=H1(J-1)*h_{J-1}
            H2(K)=H2(K+1)*h_{K+1}
200      CONTINUE
100 CONTINUE
```

(d)
```
      V01=∂F/∂G
      DO 100 I=IMIN,IMAX
         V02=∂g_I/∂H
         H1(JMIN)=1
         H2(JMAX)=1
         DO 200 J=JMIN+1,JMAX
            K=JMAX+JMIN-J
            H1(J)=H1(J-1)*h_{J-1}
            H2(K)=H2(K+1)*h_{K+1}
200      CONTINUE
         DO 300 J=JMIN,JMAX
            V03=∂h_J/∂Xm
            DFDX(m)=DFDX(m)+V01*V02*H1(J)*H2(J)*V03
300      CONTINUE
100 CONTINUE
```

[ ]: 各ステップで新たに生成した部分

図 12 多重の総和・総乗演算式の偏導関数計算プログラムの作成過程
Fig. 12 Generation of differentiation program for multiple summation/production function.

```
      V01=1
      DO 100 I=1,5
         V02=1
         DO 200 J=1,5
            V02=V02*(X(I)-X(J)**2)
200      CONTINUE
         DFDX(I)=DFDX(I)+V01*V02
         V03=X(I)
         H1(1)=1
         H2(5)=1
         DO 300 J=2,5
            K=6-J
            H1(J)=H1(J-1)*(X(I)-X(J-1)**2)
            H2(K)=H2(K+1)*(X(I)-X(K+1)**2)
300      CONTINUE
         DO 400 J=1,5
            V04=1
            DFDX(I)=DFDX(I)+V01*V03*H1(J)*H2(J)*V04
            V05=-2*X(J)
            DFDX(J)=DFDX(J)+V01*V03*H1(J)*H2(J)*V05
400      CONTINUE
100   CONTINUE
```

図 13 (3.14)式の関数 F の偏導関数計算プログラムの主要部分
Fig. 13 Main part of differentiation program for function F in (3.14).

EX：展開微分法による偏導関数計算プログラム
NS：非展開微分法による偏導関数計算プログラム
NV：NS プログラムをベクトル化した偏導関数計算プログラム

本図によると，NS プログラムは EX プログラムよりやや遅い．NS プログラムと EX プログラムとは本質的には同じ計算をしているはずであるが，NS プログラムでは，DO ループを制御するためのオーバーヘッドがかかるためと考えられる．

```
      V01=1
      DO 100 I=1,IMAX
         V02=SIN(X(I))
         DFDX(I+1)=DFDX(I+1)+V01*V02
         V03=X(I+1)*COS(X(I))
         DFDX(I)=DFDX(I)+V01*V03
  100 CONTINUE
```

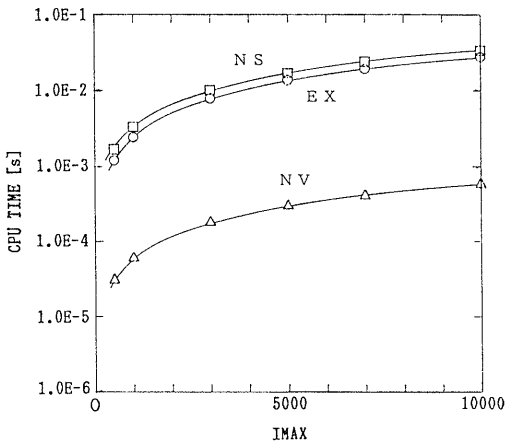
図 14 (4.1)式の関数 $F1$ の偏導関数計算プログラムの主要部分
Fig. 14 Main part of differentiation program for function $F1$ in (4.1).

NV プログラムは，図 15 に示した三つの偏導関数計算プログラムの中で最も速い．例えば，$IMAX$ が 5000 の場合で EX プログラムより 2 桁近く速く，また，この差は $IMAX$ によらず，ほぼ一定である．

いずれのプログラムにおいても，計算時間は $IMAX$ に対し対数尺度でほぼ同じ傾きで，すなわち $IMAX$ に比例して増加している．

図 16 に非展開微分法を用いて作成した，$F2$ の偏導関数計算プログラムの主要部分を示す．また，**図 17** に $F2$ の偏導関数計算に要した計算時間のグラフを示す．$F2$ においても $F1$ と定性的には同様の結果が得られている．ただし，EX プログラムと NS プログラムの計算時間の比は約 2.6 と $F1$ の場合よりも大きくなっている．これは図 16 からわかるように，DO ループの中身が十分に簡約化されていないことによる．数式の簡約化を実施することにより，NS プログラムの計算時間は，EX プログラムと同程度になると考える．

なお，これらのプログラムによって求められる偏導関数値は，理論的には同一のものであるが，実際には計算機に依存する計算誤差の累積により差異が現れることが予想される．ただし，ここで用い



EX：展開微分法による偏導関数計算プログラム
NS：非展開微分法による偏導関数計算プログラム
NV：NS プログラムをベクトル化したプログラム

図 15 $F1$ の偏導関数の計算時間
Fig. 15 Computational time to evaluate derivatives of $F1$.

た例においては，*IMAX* が 10000 の場合にも，偏導関数値は 7 桁以上の精度で一致していた．

以上から得られた知見を，以下にまとめる．

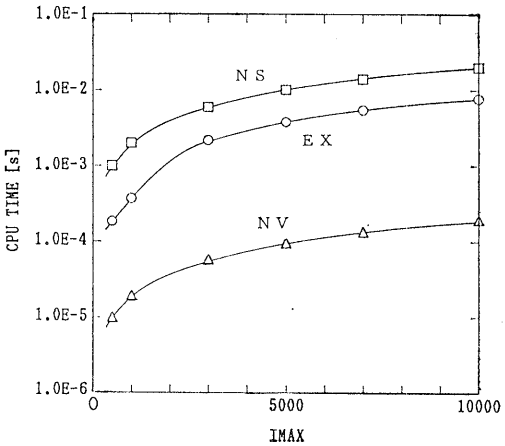
(a) 偏導関数計算プログラムの記述量は，EX プログラムが *IMAX* に比例して大きくなるのに対し，NS プログラム（NV プログラム）は一定である．

(b) EX プログラムはベクトル化できないが，NS プログラムはベクトル化可能である．

(c) EX プログラムと NS プログラムの計算時間は同程度であるが，NS プログラムをベクトル化した NV プログラムは典型的な場合にこれらより 2 桁程度速い．

```
      V01=1
      DO 100 I=1,IMAX
         V02=100*2*(X(I+1)-X(I)**2)
         DFDX(I+1)=DFDX(I+1)+V01*V02
         V03=100*2*(X(I+1)-X(I)**2)*(-1)*2*X(I)
         DFDX(I)=DFDX(I)+V01*V03
         V04=2*(1-X(I))*(-1)
         DFDX(I)=DFDX(I)+V01*V04
100   CONTINUE
```

図 16 (4.2)式の関数 *F2* の偏導関数計算プログラムの主要部分
Fig. 16 Main part of differentiation program for function *F2* in (4.2).



EX：展開微分法による偏導関数計算プログラム
NS：非展開微分法による偏導関数計算プログラム
NV：NS プログラムをベクトル化したプログラム

図 17 *F2* の偏導関数の計算時間
Fig. 17 Computational time to evaluate derivatives of *F2*.

ベクトル化の効果は，コンパイラおよびハードウエアの構成や性能に依存する面があるが，一般的なベクトル化機能を備えた計算機システムを使用する限り，NV プログラムは NS プログラムより十分に高速であると言って良い．

以上より，非展開微分法が，展開微分法より偏導関数作成手法として優れていることを確認した．

5. おわりに

総和・総乗演算式の偏導関数の値を効率良く計算するプログラムの自動作成手法を開発した．

本手法は，FAD を用いて，総和・総乗演算式の繰返し構造を反映した偏導関数計算プログラムを作成する．作成されたプログラムはベクトル化可能であり，すべての入力変数に関する偏導関数の値を高速に計算することが期待できる．ここで本手法を非展開微分法と呼称した．非展開微分法が総和・総乗演算式が多重に組み合わさった演算式に対しても適用可能であることを示した．

また，作成した偏導関数計算プログラムを用いて，簡単な総和演算式のすべての入力変数に関する偏導関数の値を計算する時間を評価し，非展開微分法の有効性を確認した．

ここで述べた偏導関数作成手法は，非線形最適化プログラムへの適用を開発目的としているが，設計や制御などの工学分野においても非線形関数の偏導関数を必要とする問題は多い．本手法はこのような問題に対しても適用することができる．

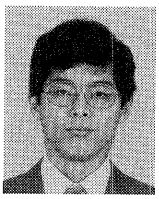
今後は，本手法を高階の偏導関数計算手法へ拡張することを検討する．例えば，正確な二階の偏導関数（すなわちヘッセ行列）が求められると，これを非線形最適化計算に応用することにより，求解の効率を高めることなどが期待される．

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) Iri, M.: Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors—Complexity and Practicality—, *Japan J. Appl. Math.*, Vol. 1, pp. 223-252 (1984).
3) Iri, M., Tsuchiya, T. and Hoshi, M.: Automatic Computation of Partial Derivatives and Rounding Error Estimates with Applications

to Largescale Systems of Nonlinear Equations, *J. Comput. Appl. Math.*, Vol. 24, pp. 365-392 (1988).
4) Rall, L. B.: *Automatic Differentiation—Techniques and Applications, Lecture Notes in Computer Science*, Vol. 120, Springer-Verlag, Berlin (1981).
5) Griewank, A.: *On Automatic Differentiation, Mathematical Programming*, pp. 83-107, KTK Science Publishers, Tokyo (1989).
6) Kagiwada, H., Kalaba, R., Rosakhoo, N. and Spingarn, K.: *Numerical Derivatives and Nonlinear Analysis*, Plenum Press, New York (1986).
7) 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).
8) 吉田利信：偏導関数自動導出システム, 情報処理学会論文誌, Vol. 30, No. 7, pp. 799-806 (1989).
9) 伊理正夫, 久保田光一：高速微分法とその応用, 第7回数理計画シンポジウム論文集, pp. 159-184 (1986).

(平成5年4月 9 日受付)
(平成6年6月20日採録)

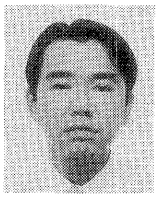


野中 久典（正会員）

昭和37年生. 昭和59年大阪大学工学部原子力工学科卒業. 昭和61年同大学大学院修士課程修了. 同年(株)日立製作所エネルギー研究所入社. 知識工学的手法を用いるプラント建設工程計画支援システム, 非線形最適化・組合せ最適化システムなどの研究開発に従事. 人工知能学会, 日本原子力学会会員.

小林 康弘（正会員）

昭和22年10月21日生. 昭和45年3月東京大学原子力工学科卒業. 昭和 50 年3月同大学院原子力工学専攻博士課程修了. 同年4月(株)日立製作所入社. 昭和 53 年4月より同社エネルギー研究所勤務, 現在に至る. プラント設計の自動化システムおよびその基盤ソフトウェア技術の開発に従事. 工学博士. 人工知能学会, 電気学会, 日本原子力学会, IEEE, AAAI などの会員.



田村 正義（正会員）

昭和37年生. 昭和59年筑波大学第3学群情報学類卒業. 昭和 61 年同大学大学院工学研究科修士課程修了. 同年(株)日立製作所入社. 現在ソフトウェア開発本部にて, 数理計画, 数値計算プログラムの開発に従事. 日本 OR 学会, 米国 OR 学会, 日本応用数理学会会員.