数理解析研究所講究録
第648巻 1988年 227-244

偏導関数の計算過程を導出するための前処理システムの改良

千葉大　工　吉田　利信（Toshinobu Yoshida）

1．はじめに

Wengert[1)]は、関数を計算過程に分解することによって自動的に全微分あるいは偏導関数値を計算する自動微分法を提案した。また Iri[9)]は、関数の計算過程を計算グラフに表現し、関数側から変数側に計算グラフを逆順にたどる方法を提案し、この方法によって効率的な偏導関数の計算過程が得られることを示した。

著者はこれらの自動微分法に基づく前処理システムをLispを用いて開発し、改良を重ねてきた[14),16),20)]。この前処理システムは、

①計算過程を計算グラフに表現する計算グラフの生成部、
②計算グラフ上を正順あるいは逆順にたどりながら全微分あるいは偏導関数の計算過程を計算グラフに追加する微分部、
③計算グラフ上の計算過程を数式として出力する出力部、
④上記の各部を操作するためのプログラムを解釈する構文解析部
から構成されている。

第2節に Wengert や Iri によって提案された自動微分法を示し、第3節に前処理システムの構造を示し、その改良点を示す。

2．自動微分法

2.1　計算過程

数式で与えられる関数は基本演算の列である計算過程に分解される。関数の値は計算過程の基本演算を順次評価することにより求められる。計算過程を次のように定義する。

①入力変数を $x_1, x_2, \cdots, x_n$ とする。
②中間変数を $v_1, v_2, \cdots, v_k$ とし、

$$v_j = \phi_j(u_1, u_2, \cdots, u_{lj}), \qquad (1)$$

と定義する。ここで、ϕ_j は四則演算、初等関数および関数副プログラムで定義される演算子、あるいは、定数演算子、入力変数演算子のいずれかを表す。l_j は被演算子の数を表し、u_i は $v_1, v_2, \cdots, v_{j-1}$ のいずれかの中間変数を表す。定数演算子および入力変数演算子には被演算子はなく、それぞれ定数または入力変数を値に持つ。

③出力変数を $f_1, f_2, \cdots, f_m$ とする。それぞれは中間変数v_{f1}, $v_{f2}, \cdots, v_{fm}$ に対応する。

2.2 全微分と偏微分

計算過程が与えられているとき、その全微分は次のように定義される。

①入力変数 $x_1, x_2, \cdots, x_n$ に対して、その全微分をそれぞれ $dx_1, dx_2, \cdots, dx_n$ とする。

②式(1)で定義される中間変数 $v_1, v_2, \cdots, v_k$ に対して、それらの全微分を次のように定義する。

$$dv_j = \sum_{i=1}^{l_j} \partial\phi_j/\partial u_i \cdot du_i . \qquad (2)$$

ここで、$\partial\phi_j/\partial u_i$ を要素的偏導関数と呼び、$\partial v_j/\partial u_i$ と書くこともある。du_i は u_i に対応する中間変数 v の全微分 dv とする。また、ϕ_j が定数演算子のときは $dv_j=0$、ϕ_j が x_i に対応する入力変数演算子のときは $dv_j=dx_i$ と定義する。

計算過程の各演算ごとに式(2)の計算を行うことによって、次の式で表される出力変数 f_j の全微分df_j が、f_j に対応する中間変数 v_{fj} の演算とともに dv_{fj} として得られる。

$$df_j = \sum_{i=1}^{n} \partial f_j/\partial x_i \cdot dx_i , \quad 1 \leqq j \leqq m. \qquad (3)$$

出力変数 f_j の入力変数 x_i に関する偏導関数値 $\partial f_j/\partial x_i$, $1 \leqq j \leqq m$ は、

$$dx_{i'}=0, \ i' \neq i, \quad dx_i=1, \qquad (4)$$

と置き、全微分を求めることにより df_j に得られる。したがって、すべての偏導関数値はこの全微分の操作を 1 からn までの i について繰り返すことにより求められる。

2.3 計算グラフ

計算過程に対して計算グラフを次のように定義する。
①中間変数 $v_1, v_2, \cdots, v_k$ に対して同名の節点 $v_1, v_2, \cdots, v_k$ を対応させる。
②各演算 $v_j=\phi_j(u_1, u_2, \cdots, u_{lj})$ に対して、節点 $u_i(1 \leqq i \leqq l_j)$から節点 v_j への有向辺を対応させ、それぞれの有向辺に要素的偏導関数 $\partial \phi_j / \partial u_i$ を置く。

2.4 計算グラフ上の微分算法

2.4.1 正順全微分算法（算法A）

2.2 節に定義した全微分を求める方法は計算グラフ上では次の操作に対応する。

算法A

計算グラフの節点 $v_1, v_2, \cdots, v_k$ についてこの順に次の操作を行う。節点 v_j に入る有向辺の始点の節点を$u_1, u_2, \cdots, u_{lj}$ とするとき、各節点 u_i に置かれたdu_iと、節点u_i から節点 v_j への有向辺上の要素的偏導関数 $\partial v_j / \partial u_i$ との積をそれぞれの有向辺について加え合わせ dv_jとし、つまり、式(2)を計算し、これを節点v_j に置く。ただし、節点 v_j が入力変数 x_i に対応する計算グラフの節点であるときは、その節点に dx_iを置く。

計算グラフの節点の数を k_v、辺の数を k_e、加減算の数を k_a、入力変数の数を n とすると、算法Aに必要な乗算の数は高々 k_e-2k_a、加算の数は高々 $k_e-(k_v-n)$ である。すべての演算が単項または二項演算のときは $k_e \leqq 2(k_v-n)$ より、算法Aに必要なすべての計算が高々 $3(k_v-n)$ で求められる。実際多くの場合、計算過程中の演算はほとんどが四則演算であり、要素的偏導関数の計算に必要な計算量はごくわずかである。したがって、関数自身を計算する計算量の高々４倍の計算量で関数値と全微分が求められる。

2.4.2 正順偏微分算法（算法B）

すべての偏導関数値は計算グラフ上の次の操作を用いて求められる。

算法B

１からｎまでの各ｉについて、式(4)により $dx_1, dx_2, \cdots, dx_n$ を定め算法Aを行う。

この算法では算法Aをn回繰り返すことから、関数自身を計算する計算量の高々 1+3n 倍の計算量で関数値とすべての偏導関数値が得られる。

2.4.3 逆順微分算法（算法C）

算法Aとは逆に出力変数側から入力変数側へ計算する算法Cを次のように定義する。

算法C

計算グラフの節点 $v_k, v_{k-1}, \cdots, v_1$ についてこの順に次の操作を行う。節点 v_i から出る有向辺の終点の節点を $u_1, u_2, \cdots, u_{1j}$ とするとき、各節点 u_i に置かれた du_i と、節点 v_j から節点 u_i への有向辺上の要素的偏導関数 $\partial u_i / \partial v_j$ との積をそれぞれの有向辺について加え合わせ dv_j とし、つまり、

$$dv_j = \sum_{i=1}^{1j} \partial u_i / \partial v_j \cdot du_i \qquad (5)$$

を計算し、これを節点 v_j に置く。ただし、節点 v_j が出力変数 f_i に対応する計算グラフの節点であるときは、この dv_j に y_i を加えたものを節点 v_j に置く。

この算法Cによって、入力変数 $x_1, x_2, \cdots, x_n$ に対応する計算グラフの節点 $v_{x1}, v_{x2}, \cdots, v_{xn}$ に次の値が得られる。

$$dv_{xi} = \sum_{j=1}^{m} \partial f_j / \partial x_i \cdot y_j, \quad 1 \leqq i \leqq n. \qquad (6)$$

計算グラフの節点の数を k_v、辺の数を k_e、加減算の数を k_a、出力変数の数を m とすると、算法Cに必要な乗算の数は高々 $k_e - 2k_a$、加算の数は高々 $k_e - (k_v - m)$ である。したがって、すべての演算が単項または二項演算で、要素的偏導関数の計算に計算量がほとんど必要でない場合は、関数自身を計算する計算量の高々４倍の計算量で関数値と式(6)の各 dv_{xi} が求められる。

2.4.4 逆順偏微分算法（算法D）

すべての偏導関数値は次の操作を用いても求められる。

算法D

1からmまでの各iについて、

$y_{i'}=0,\ i'\neq i,\quad y_i=1,$ (7)

となるように $y_1,y_2,\cdots,y_m$ を定め算法Cを行う。

この算法では算法Cをm回繰り返すことから、関数自身を計算する計算量の高々 1+3m 倍の計算量で関数値とすべての偏導関数値が得られる。出力変数が一つのときは関数自身を計算する計算量の高々4倍の計算量で、関数値とすべての偏導関数 $\partial f_1/\partial x_i$ が各 dv_{xi} に得られる。算法Bを用いた場合には高々 1+3n 倍の計算量が必要なことから、スカラー関数に対するこの算法Dは高速自動微分法と呼ばれる。

算法Aと算法C、あるいは、算法Bと算法Dは本質的に同じ算法である。すなわち、算法Aにおいて式(2)を数値計算をせずに dx_i との積和形式に展開し式(3)の $\partial f_j/\partial x_i$ を表現し、また、算法Cにおいて式(5)を数値計算をせずに y_j との積和形式に展開し式(6)の $\partial f_j/\partial x_i$ を表現すると、どちらも計算グラフの x_i に対応する節点v_{xi} からf_j に対応する節点 v_{fj} への経路上の要素的偏導関数の積のすべての経路についての和を表現している。

3．前処理システム

3.1 システムの概要

算法Aおよび算法Bあるいはその拡張である高階の偏導関数を求める算法の実現方法には、Wengert[1]，Kalaba et al.[6),13),18)] などに示されているように、関数の計算過程に沿って、基本演算と式(2)とを同時に計算するサブルーチンを呼び出す列を記述する方法がある。また、Wexler[19)] に示されているように、関数を表す式に対して、基本演算と式(2)とを同時に計算する関数副プログラムからなる式を記述する方法もある。一方、関数を表す式を構文解析し、関数の計算過程を自動的に導出し、算法Aなどを実現するシステムは Pugh[2)],Kedem[3)]，Rall[4)] などによって作成されている。

スカラー関数に対する高速自動微分法は、 Baur et al.[5)]，Kim et al.[7),8)]，Iri[9)]，Sawyer[10)] によって独立に示されているが、Iri[9)]の示した算法Dは、計算過程を計算グラフとして表現したことによって非常に見通しがよい。計算グラ

フを用いて算法A～Dを実現する方法として、計算の実行時に計算グラフを作成し偏導関数値を計算する方法と、前処理において計算グラフを作成し偏導関数の計算過程を出力する方法がある。実行時に計算グラフを作成する方法では、計算グラフの操作のために実行時間が多く必要になるが、実行時に計算過程が定まる場合でも偏導関数値を容易に求めることができる。この方法に基づいたシステムは岩田[11]、久保田ほか[17]により作成されている。

著者は次のような前処理システムを Lisp を用いて作成してきた[14),16),20)]。

①入力する関数の表現は、通常用いられている数式表現とする。

②前処理時に計算グラフを作成し、演算の再利用および簡約化を行う。

③算法A～Dによって得られる偏導関数の計算過程も計算グラフとして表現する。

④出力される関数の計算過程は、通常用いられている数式表現とする。

このシステムによって出力された計算過程は、実行時に計算グラフを操作しないこと、および、演算の再利用および簡約化がなされていることから、実行時に計算グラフを作成するシステムによって出力された計算過程に比べて高速に実行される。また、偏導関数の計算過程も計算グラフとして表現されることから、高階の偏導関数の計算過程も容易に導出できる。

しかし、この前処理システムの欠点として、

①入力された式を構文解析し計算グラフを生成する部分、微分を行う部分および出力部分を別個に操作しなければならない、

②スカラー関数の勾配ベクトルやヘッセ行列を扱うとき、その成分を記述しなければならない、

③利用者によって定義される演算子を記述できない

などがあった。そこで、前処理システム用のプログラミング言語を定義し、前処理システムを記述している言語 Lisp を離れて利用できるように改良した。この言語の特徴は次の通りである。

①ベクトルや行列などを用いて式を記述できる。

②式をベクトル変数や行列変数などで微分する微分演算子を式中に含むことができる。

③式中に関数副プログラムで定義される関数を演算子として含むことができ、それらに対する微分も行うことができる。

3.2 計算グラフの内部表現

中間変数 v_i が式(1)で表されているとき、v_i に対応する計算グラフの節点 v_i をその節点名を持つアトムで表す。計算グラフはこれらの節点を表すアトムの性質リストを用いて次のように表される。

①演算子を性質名 OP の性質リストに記述する。定数演算子および入力変数演算子については、個々の演算子を記述せずにそれぞれ CONST および IN とする。

②被演算子を性質名 ARGS の性質リストに記述する。定数演算子および入力変数演算子には被演算子はないが、それぞれ定数の値および入力変数名を記述する。また、単項演算子の被演算子はその被演算子に対応する節点名を、多項演算子の被演算子はそれらの被演算子に対応する節点名のリストを記述する。

③その節点から出る辺の終点の節点名を性質名 TO の性質リストに記述する。

3.3 計算グラフの生成部

式で与えられる関数は構文解析部によって基本演算に分解され、その演算に対する節点が計算グラフ生成部で生成される。その際、x+0 => x, x*1 => x, x+(-y) => x-y, x/x => 1 などの簡約化がなされる。

また、既に生成されている節点に対応する演算に対して新たに節点は生成されない。演算子を ϕ, 被演算子を $u_1, u_2, \cdots, u_l$ とするとき、これは次のように行われる。

算法E

①節点 u_1 から出る辺の終点の節点名のリストLを性質名 TO の性質リストから得る。

②Lの各節点 v 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を新しく生成し、終了する。

③節点 v の性質名 OP の性質リストからその節点の演算子を得る。それが ϕ 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 の性質名 ARGS の性質リストの値が被演算子 $u_1, u_2, \cdots, u_l$ と一致すれば、この演算が既に節点 v 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。

3.4 微分部

与えられた式は構文解析部と計算グラフ生成部により逐次計算グラフに表現されていく。式中には微分演算子を含むことができ、その時点での計算グラフに対して偏導関数の計算過程が導出され、この計算過程も計算グラフ生成部により計算グラフに追加される。

微分演算子は被演算子として被微分関数を表す式と微分変数を持つ（3.6 節に述べるが微分変数としてベクトル変数、行列変数などを記述できる）。被微分変数に対応するいくつかの節点と微分変数に対応するいくつかの節点とを結ぶすべての経路上の節点と有向辺からなる部分計算グラフに対して微分演算を行う。部分計算グラフを抽出するための算法Ｆと算法Ｇを次のように定義する。

算法F

①与えられたいくつかの中間変数に対応する節点のそれぞれにマークを付ける。

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。性質名 TO の性質リスト中にマークの付けられた節点がある節点にマークを付ける。

算法G

①与えられたいくつかの中間変数に対応する節点のそれぞれにマークを付ける。

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。性質名 ARGS の性質リスト中にマークの付けられた節点がある節点にマークを付ける。

算法Ａおよび算法Ｂを表す正順の微分演算子に対しては、算法Ｆを用いて被微分変数に接続する節点をマークし、マークされている節点に対してそれぞれ算法Ａおよび算法Ｂを行う。算法Ｃおよび算法Ｄを表す逆順の微分演算子に対しては、算法Ｇを用いて微分変数に接続する節点をマークし、マークされている節点に対してそれぞれ算法Ｃおよび算法Ｄを行う。

3.5 出力部

たとえば、偏導関数の値のみ必要で関数値は不要の場合など、計算グラフに表されている計算過程のすべてについては数値計算をする必要がない場合がある。

また、入力変数が独立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、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、入力変数の中のいくつかの独立変数と出力変数の中のいくつかの従属変数を指定し、次のように計算過程を導出する。

①算法Fを用いて、従属変数に関係する部分計算グラフを抽出する。

②さらに算法Gを用いて独立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を分離する。

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一回のみ用いられる演算、すなわち、有向辺が一本だけ出ている節点に対応する演算は代入文として生成せず、その有向辺の終点の節点に対応する演算に組み込んで出力する。

3.6 構文解析部

スカラー関数の勾配ベクトルやヘッセ行列などを扱うとき、その成分の数が多くなると成分を記述することが煩わしくかつ間違いを犯しやすくなる。Kalaba et al.[18] は行列を扱うサブルーチン列によって問題を記述する方法を示しているが、自動化されていないため記述性はあまりよくない。そこで、ベクトルや行列など記法を用いて式を記述でき、かつ、式をベクトル変数や行列変数などで微分する微分演算子を式中に含むことができるプログラミング言語を開発した。この言語は数式処理言語のように、式の入力、行列の宣言、微分操作、ファイルの入出力操作などを記述することができ、前述の計算グラフ生成部、微分部、出力部を効率よく操作することができる。

3.6.1 定数

整数型定数、実数型定数、倍精度実数型定数、文字列型定数を記述できる。整数型定数、実数型定数、倍精度実数型定数に対しては計算グラフの節点が生成される。文字列型定数は、文字列の前後を”で囲んだものである。文字列中の”は”” として記述する。

3.6.2 変数の型とその宣言

任意の階数のテンソルを次のように宣言する。

MATRIX a(n_1,n_2,・・・,n_j),・・・;

ここでａはｊ階のテンソルを表す変数名、n_1,n_2,・・・,n_j は添字の範囲が１からそれぞれn_1,n_2,・・・,n_j であることを表す。添え字の範囲の組（n_1,n_2,・・・,n_j）を型と呼ぶことにする。このように変数を宣言すると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。０階のテンソル変数（スカラー変数）および代入文により添字の範囲が決定できる場合は宣言を省略できる。変数ａの添字が i_1, i_2,・・・,i_j である要素を a(i_1,i_2,・・・,i_j)、要素全体をａと引用する。また、a(i_1,i_2,・・・,i_k), $1 \leq k < j$ と引用することによって、型が（n_{k+1},n_{k+2},・・・,n_j）であるj-k 階のテンソル変数として扱うことができる。

3.6.3 演算子および関数

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、対応する各要素の演算に対して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。スカラー倍については各要素との積に対して計算グラフの節点が生成される。第１被演算子が型（n_1,n_2,・・・,n_j）のテンソル、第２被演算子が型（n_j）のベクトルのとき、その積はテンソルの最後の添字で表されるベクトルと第２被演算子のベクトルとの内積として定義され、その計算過程が計算グラフに追加され、演算結果の型は（n_1,n_2,・・・,n_{j-1}）となる。第２被演算子が型（n_j,n_{j+1}）の行列のとき、その積はテンソルの最後の二つの添字で表される行列と第２被演算子の行列との積として定義され、その計算過程が計算グラフに追加され、演算結果の型は（n_1,n_2,・・・,n_{j-1},n_{j+1}）となる。

③除算の演算子を／とする。テンソルをスカラーで割る場合についてのみ定義され、その計算過程が計算グラフに追加される。

④巾乗算の演算子を ** とする。スカラーのスカラー乗および正方行列の正整数乗のみについて定義され、その計算過程が計算グラフに追加される。

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関数を VP とする。その計算過程は計算グラフに追加される。

⑥行列の転置を求める演算子を TP とする。

⑦テンソルの階数を上げる演算子を VEC とする。任意の数の同一の型を持つ式を被演算子として持つ。式の数を n，式の型を（n_1,n_2,・・・,n_j）とするとき、演算

結果の型は（$n, n_1, n_2, \cdots, n_j$）となる。

⑧スカラー変数のスカラー関数 SQRT,CBRT,EXP,LOG,LOG10,SIN,COS,TAN,COTAN, ASIN,ACOS,ATAN,SINH,COSH,TANH を記述できる。これらの関数を演算子とする節点が生成される。

⑨微分演算子には正順全微分算法（算法A）を表す DFUV、正順偏微分算法（算法B）を表す DFU、逆順微分算法（算法C）を表す DFDV および逆順偏微分算法（算法D）を表す DFD がある。これらの演算子の第１被演算子には微分される式を、第２被演算子には微分する変数を記述する。演算子 DFU および DFD において、微分される式の型を（$n_1, n_2, \cdots, n_j$），微分する変数の型を（$n_1', n_2', \cdots, n_j'$）とするとき、結果の式の型は（$n_1, n_2, \cdots, n_j, n_1', n_2', \cdots, n_j'$）となる。演算子 DFUV は微分する変数と同じ型の式を第３被微分演算子に持ち、結果の式の型は微分される式の型と同一である。また、演算子 DFDV は微分される式の型と同じ型の式を第３被微分演算子に持ち、結果の式の型は微分する変数と同一である。これらの微分演算によって生成された計算過程は計算グラフに追加される。

⑩丸め誤差を推定する関数を ERR とする。伊理ほか[12]および土谷ほか[15]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。

$$((1/3)\cdot\sum_{i=1}^{k} \partial f/\partial v_i \cdot v_i^2)^{1/2}\,\varepsilon . \qquad (8)$$

ここで、εは浮動小数点演算で $(1+\varepsilon)\neq 1$ となるような最小の数とする。関数 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追加する。関数 ERR の第１被演算子には誤差評価の対象となる式を、第２被演算子にはその式における独立変数を記述する。第３被演算子にはεを表すスカラー変数を記述する。演算結果の型は第１被演算子の型と同一である。

⑪関数副プログラムなどで定義される関数は

OPERATOR op(n),・・・;

と宣言することによって演算子として記述できる。ここでｎはその演算子 op の被演算子の数を表す。被演算子にはスカラー型の式のみ記述できる。また、演算子 op の演算結果の型がスカラー型でないときは、OPERATOR 宣言のあとで MATRIX 宣言を行うことにより演算結果の型を宣言できる。演算子 op の被演算子の数をｎ、演算結果の型を（$n_1, n_2, \cdots, n_j$）とするとき、op($i_1, i_2, \cdots, i_k$)($arg_1, \cdots, arg_n$)，$1 \leq k \leq j$ と引用することによって、演算結果の型は（n_{k+1},

n_{k+2},・・・,n_j）となる。また、op(arg_1,・・・,arg_n）と引用することによって、演算結果の型は（n_1,n_2,・・・, n_j）となる。被演算子の数がn、演算結果の型が（n_1, n_2,・・・,n_j）の演算子 op を DFU またはDFD で微分すると、自動的に被演算子の数がn、演算結果の型が（n_1,n_2,・・・,n_j,n）の演算子 op' が宣言される。ここで、op' は op の演算子名の先頭に D をつけた演算子名である。また、DFUV で微分すると、自動的に被演算子の数がn、演算結果の型が（n_1,n_2,・・・,n_j）の演算子 op' が宣言される。ここで、op' は op の演算子名の先頭に V をつけた演算子名である。DFDV での微分は現時点では定義されていない。

3.6.4 文

プログラムは以下に示す文を；で区切って記述する。

① CLEAR 文はその時点での計算グラフを消去し、初期状態にする。

② FREE 文は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。

③ IN 文は IN のうしろにファイル名を表す文字列型定数を記述する。IN 文を実行後、入力は指定されたファイルからなされる。ファイルからの入力を終了後、入力は IN 文の次の文から再開される。ファイル中に IN 文を含んでもよい。

④ OUT 文は OUT のうしろにファイル名を表す文字列型定数を記述する。OUT 文を実行後、出力は指定されたファイルに対してなされる。いくつかのファイルに OUT 文で切り替えながら出力することができる。

⑤ SHUT 文は SHUT のうしろにファイル名を表す文字列型定数を記述する。SHUT 文を実行後、出力は標準出力に対してなされる。

⑥ ON 文は ON のうしろに FORT または ECHO を記述する。FORT は計算過程の出力を FORTRAN 形式で出力することを、 ECHO は入力を標準エラー出力にエコーすることを示す。

⑦ OFF 文は OFF のうしろに FORT または ECHO を記述する。FORT は計算過程の出力をFORTRAN 形式で出力せず、⑫に示す代入文の形式で出力することを、ECHO は入力を標準エラー出力にエコーしないことを示す。

⑧ WRITE 文は WRITE のうしろに文字列型定数を記述する。WRITE 文によってその文字列型定数が出力される。

⑨ OUTCODE 文は

OUTCODE(x_1,x_2,・・・,x_n)f_1,f_2,・・・,f_m;

のように記述する。これによって独立変数を x_1,x_2,・・・,x_n としたときの従属変数 f_1,f_2,・・・,f_m の計算過程が出力される。。

⑩ MATRIX 文については 3.6.2 節に示した。

⑪ OPERATOR 文については 3.6.3 節に示した。

⑫代入文は次のように記述する。

1) a:=e

ここで a は変数名、e は式とする。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 の型は自動的に式 e の型として宣言される。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。

2) a(i_1,i_2,・・・,i_k):=e

ここで a は型（n_1,n_2,・・・,n_j）を持つ変数名、e は型（n_{k+1},n_{k+2},・・・,n_j）を持つ式、k は $1 \leq k \leq j$ 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。

4．まとめ

偏導関数値を計算する方法として、数値微分法、数式処理による方法があるがそれぞれ欠点がある。これらの方法に対して自動微分法が提案され、偏導関数値が正確に、かつ、高速に求められることが示された。この自動微分法の算法を第２節に示した。

自動微分法に基づいて偏導関数値の計算を行うためには、関数を計算過程に分解することなど種々の作業が必要である。そこで関数を記述するだけでこれらの作業を自動的に行うシステムが作られてきた。このようなシステムの実現方法にはいろいろあるが、著者は前処理時に計算グラフを作成し偏導関数の計算過程を導出する前処理システムを作成した。このシステムによって出力された計算過程は、実行時に計算グラフを操作しないこと、および、演算の再利用および簡約化がなされていることから高速に実行されるという特徴があった。さらに本研究では第３節に示したように前処理システム用のプログラミング言語を定義し、システムを改良した。その結果、ベクトルや行列などを用いて式を記述でき、また、式中に微分演算子を記述できるなど、記述性がたいへんよくなった。

付録１にプログラム例を示す。付録２には付録１のプログラムによって前処理システムが出力した FORTRAN のプログラムを示す。

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。

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 Programing 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., Raskhoo, N. and Tishler, A.: Nonlinear Least Squares via Automatic Derivative Evaluation, Appl. Math. and Comp., 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, The 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, Comp. and Maths. with Appls., Vol. 12A, No. 11, pp. 1091-1103 (1986).
14) 吉田利信，山下稔：計算グラフを用いた数値計算のための数式処理システム，情報処理学会第33回全国大会，pp. 1877-1878 (1986).
15) 土谷隆，笹山晋一：非線形方程式系の解法に対する高速微分法の応用，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. 96-116 (1987).
16) 山下稔，吉田利信：計算グラフを用いた数値計算のための数式処理システムの設計，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. 145-153 (1987).
17) 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).
18) Kalaba, R., Plum, T. and Tesfatsion, L.: Automation of Nested Matrix and Derivative Operations, Appl. Math. and Comp., Vol.23, pp.243-268 (1987).
19) Wexler, A. S.: Automatic Evaluation of Derivatives, Appl. Math. and Comp., Vol.24, pp. 19-46 (1987).
20) 吉田利信：偏導関数の計算過程を導出するための前処理システムの作成，電子情報通信学会コンピュテーション研究会，Vol. COMP87-28, pp.21-27 (1987).

付録１．入力プログラム例

```
on fort;
out "katsura.for";
write "      PROGRAM KATURA";
write "      IMPLICIT REAL*8 (A-H,O-Z)";
write "      DIMENSION R(4)";
write "      EPS=1.0E-29";
write "      J=1";
write "      K=40";
write "      WRITE(*,'(1X,A)') 'INPUT INITIAL VALUES OF X1,X2,X3,X4'";
write "      READ(*,*) X1,X2,X3,X4";
write "   10 CONTINUE";
 x:=vec(x1,x2,x3,x4);
 f:=vec(2*(x4*x4+x3*x3+x2*x2)+x1*(x1-1),
        2*x3*(x4+x2)+x2*(x1-1+x1),
        x2*(x4+x2+x4)+x3*(x1-1+x1),
        2*(x4+x2)+2*x3+(x1-1));
 ff:=f*f;
 r:=-dfd(ff,x);
outcode(x)ff,r;
write "      WRITE(*,'(1X,I5,E18.10)') J,FF";
write "      IF(FF .LT. EPS) GOTO 20";
write "      IF(K .EQ. 40) THEN";
write "       P1=R(1)";
write "       P2=R(2)";
write "       P3=R(3)";
write "       P4=R(4)";
write "       K=0";
write "      ELSE";
 p:=vec(p1,p2,p3,p4);
 y:=dfuv(-r,x,p);
 b:=(r*y)/(p*y);
outcode(p)b;
write "       P1=R(1)-B*P1";
write "       P2=R(2)-B*P2";
write "       P3=R(3)-B*P3";
write "       P4=R(4)-B*P4";
write "       K=K+1";
write "      END IF";
 a:=(r*p)/(p*y);
outcode(p)a;
write "      X1=X1+A*P1";
write "      X2=X2+A*P2";
write "      X3=X3+A*P3";
write "      X4=X4+A*P4";
write "      J=J+1";
write "      GO TO 10";
write "   20 WRITE(*,'(1X,I5,E18.10/1X,4E18.10)') J,FF,X1,X2,X3,X4";
write "      END";
shut "katsura.for";
```

付録２．付録１のプログラムに対する出力

```
      PROGRAM KATURA
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(4)
      EPS=1.0E-29
      J=1
      K=40
      WRITE(*,'(1X,A)') 'INPUT INITIAL VALUES OF X1,X2,X3,X4'
      READ(*,*) X1,X2,X3,X4
   10 CONTINUE
      V0014=X1-1
      V0016=2*(X4*X4+X3*X3+X2*X2)+X1*V0014
      V0017=X3*2
      V0018=X2+X4
      V0020=X1+V0014
      V0022=V0017*V0018+X2*V0020
      V0023=X4+V0018
      V0026=X2*V0023+X3*V0020
      V0029=V0014+V0017+2*V0018
      V0037=2*V0029
      V0038=2*V0026
      V0039=X2*V0038
      V0040=2*V0022
      V0043=X2*V0040+X3*V0038
      V0047=V0017*V0040+V0039+2*V0037
      V0050=2*V0016
      V0054=2*V0050
      V0055=X4*2
      V0058=V0054*V0055+V0039+V0047
      V0063=V0017*V0054+2*(V0037+V0018*V0040)+V0020*V0038
      V0064=X2*2
      V0070=V0054*V0064+V0047+V0020*V0040+V0023*V0038
      V0073=X1*V0050+V0037+V0043+V0043+V0014*V0050
      FF=V0016*V0016+V0022*V0022+V0026*V0026+V0029*V0029
      R(1)=-(X1*V0050+V0037+V0043+V0043+V0014*V0050)
      R(2)=-(V0054*X2*2+V0047+V0020*V0040+V0023*V0038)
      R(3)=-(V0017*V0054+2*(V0037+V0018*V0040)+V0020*V0038)
      R(4)=-(V0054*X4*2+V0039+V0047)
      WRITE(*,'(1X,I5,E18.10)') J,FF
      IF(FF .LT. EPS) GOTO 20
      IF(K .EQ. 40) THEN
       P1=R(1)
       P2=R(2)
       P3=R(3)
       P4=R(4)
       K=0
      ELSE
       V0092=2*P3
       V0093=P2+P4
       V0097=2*P1
```

```
      V0102=P4+V0093
      V0113=2*(P1+V0092+2*V0093)
      V0114=2*(V0023*P2+X2*V0102+V0020*P3+X3*V0097)
      V0117=V0038*P2+X2*V0114
      V0118=2*(V0018*V0092+V0017*V0093+V0020*P2+X2*V0097)
      V0125=V0040*P2+X2*V0118+V0038*P3+X3*V0114
      V0131=V0040*V0092+V0017*V0118+V0117+2*V0113
      V0136=2*(2*(V0055*P4+V0017*P3+V0064*P2)+V0014*P1+X1*P1)
      V0137=V0050*P1
      V0142=2*V0136
      V0148=V0055*V0142+V0054*2*P4+V0117+V0131
      V0157=V0054*V0092+V0017*V0142+2*(V0113+V0040*V0093+V0018*V0118)+
     &  V0038*V0097+V0020*V0114
      V0170=V0064*V0142+V0054*2*P2+V0131+V0040*V0097+V0020*V0118+V0038*
     &  V0102+V0023*V0114
      V0174=V0137+X1*V0136+V0113+V0125+V0125+V0137+V0014*V0136
      B=(-(V0073*V0174+V0070*V0170+V0063*V0157+V0058*V0148))/(P1*V0174+
     &  P2*V0170+P3*V0157+P4*V0148)
       P1=R(1)-B*P1
       P2=R(2)-B*P2
       P3=R(3)-B*P3
       P4=R(4)-B*P4
       K=K+1
     END IF
     V0092=2*P3
     V0093=P2+P4
     V0097=2*P1
     V0102=P4+V0093
     V0113=2*(P1+V0092+2*V0093)
     V0114=2*(V0023*P2+X2*V0102+V0020*P3+X3*V0097)
     V0117=V0038*P2+X2*V0114
     V0118=2*(V0018*V0092+V0017*V0093+V0020*P2+X2*V0097)
     V0125=V0040*P2+X2*V0118+V0038*P3+X3*V0114
     V0131=V0040*V0092+V0017*V0118+V0117+2*V0113
     V0136=2*(2*(V0055*P4+V0017*P3+V0064*P2)+V0014*P1+X1*P1)
     V0137=V0050*P1
     V0142=2*V0136
     A=(-(V0073*P1+V0070*P2+V0063*P3+V0058*P4))/(P1*(V0137+X1*V0136+
     & V0113+V0125+V0125+V0137+V0014*V0136)+P2*(V0064*V0142+V0054*2*P2+
     & V0131+V0040*V0097+V0020*V0118+V0038*V0102+V0023*V0114)+P3*(V0054
     & *V0092+V0017*V0142+2*(V0113+V0040*V0093+V0018*V0118)+V0038*V0097
     & +V0020*V0114)+P4*(V0055*V0142+V0054*2*P4+V0117+V0131))
     X1=X1+A*P1
     X2=X2+A*P2
     X3=X3+A*P3
     X4=X4+A*P4
     J=J+1
     GO TO 10
  20 WRITE(*,'(1X,I5,E18.10/1X,4E18.10)') J,FF,X1,X2,X3,X4
     END
```