RM-78-45

DIF: AUTOMATIC DIFFERENTIATION OF FORTRAN-CODED POLYNOMIALS

W. Orchard-Hays
I. Butrimenko

September 1978

Preface

In considering possible implementation of a software system for nonlinear programming problems, one of the first practical difficulties met was the necessity to incorporate, for each model and version, a set of nonlinear multivariate functions and their partial derivatives. These functions are not known to the system designer and even simple ones are not readily expressed in convenient input formats; at least, no standard, widely accepted formats of this kind exist.

Essentially the only standard available for such a purpose is the FORTRAN language, and it appears most reasonable to require a system user to program his functions in FORTRAN. Although this creates another problem -- how to link compiled subroutines to a large, integrated application system -- that is not the difficulty in this paper. The linking of user-supplied subroutines to a standard control program or package of routines is fairly standard practice for many purposes.

If it is reasonable to expect a user to write (possibly several), functions of several variables in FORTRAN, it is not so reasonable to also require him to write subroutines for all the partial derivatives. At the same time, differentiating a function defined only by a subroutine during execution of an algorithm is close to an impossibility. The answer seemed to be to provide a separate preprocessor which accepts a FORTRAN routine in source code along with an indication of desired partials, and produces other FORTRAN source code which, when compiled and executed, computes the required derivatives. That is what the set of routines (also written in FORTRAN) described in this paper do.

Summary

The program described creates the first derivative functions of given function of limited complexity, namely generalized polynomials, but involving possibly many variables. Neither nested functions nor general rational forms are handled. However, partials to such functions can usually be readily programmed utilizing those forms produced automatically. Similarly, if a function of many variables is nonlinear in only a few, only derivatives of nonlinear terms need to be created; the constant derivatives can be added with ordinary DO-loops or similar standard programming techniques.

Three main files are used, indicated by integer variables IN, IFIN and IOUT. Using these symbols to denote the files, IN contains the FORTRAN source code representing one or more functions f(X) to be differentiated, where X is a vector. The code must contain special comment lines delimiting and identifying each function, File IFIN contains additional comment lines which are really statements in a stylized language which specifies differentiation, referring to the functions defined in IN. File IFIN can be a preprogrammed (but incomplete) FORTRAN routine which includes the differentiation statements at appropriate points. File IFOUT, the output, contains the program from IFIN elaborated with statements to compute the derivatives specified. The comment lines are retained and serve as useful comments in the final routine.

After an explanatory foreword, usage of the program is explained. This can be regarded as a users manual. Thereafter, detailed explanations of the method, including flowcharts, are given.

DIF

Automatic Differentiation of FORTRAN-coded Polynomials

FOREWORD

The program described in this document for automatically creating the derivative functions of given functions is not intended for functions of arbitrary complexity. The initial version described here handles only a highly restricted class of functions, namely generalized polynomials (i.e. exponents may be any nonzero real numbers) in many variables. It is the "many variables" that is important. The idea is related to non-linear programming models of the type in which the functions may be linear in most variables or where each nonlinear function involves a relatively small subset of the variables. Such models may be partially represented in the matrix form of linear programming, and supplemented by nonlinear partial functions. For example, the constraint functions G_i might have the form

$$G_i = g_i(X_1,\ldots,X_p) + \sum_{j=p+1}^{n} a_{ij}X_j$$

Thus, the matrix of partial derivatives at a point $X = \overline{X}$ would be

$$\left[\left(\frac{\partial g_i}{\partial X_j}\,\middle|_{\substack{X=\overline{X}\\ j\leq p}}\right)\ \left(a_{ij}\right)_{j>p}\right]$$

that is, the last n - p columns are constant.

If i has the range 1,2,...,50 and p = 15, then 50 functions g_i in up to 15 variables each must be written in some programming language--and here FORTRAN is assumed. (The remaining parts of the G_i are readily evaluated with simple DO-loops). This is a tedious and error-prone task. If one must additionally write by hand the (up to) 750 functions $\frac{\partial g_i}{\partial x_j}$, the burden is almost intolerable. It is this part which the program described automates.

We are considering adding a few common functions to the allowable terms of the g_i. If the program in its present form is found useful, we will consider this more earnestly. However, once the possibility of nested functions or general rational forms is opened up, the complexity increases enormously. For example, consider the relatively simple function

$$\exp(X_1 \cdot X_2 + \sin(X_1 \cdot X_3 + X_2)) \quad .$$

Differentiating this with respect to, say, X_1, introduces entirely new dimensions of complexity in the differentiating program over what is now required. It is not clear how much value there would be in providing functions restricted to the form

$$fn(t(X))$$

where t(X) is a product of powers of the X_j. This would be relatively easy to add for fn = exp, sin, cos, log and perhaps one or two more.

General rational forms in polynomials do not appear feasible at present. This is largely due to the difficulty of multiplying

polynomials symbolically. However, it is readily programmed in the skeleton routine by the user. For example, suppose p(X) and q(X) are polynomial and g(X) = p(X)/q(X). The derivatives of p(X) and q(X) can be obtained automatically with respect to as many variables as necessary. Suppose they are evaluated in the variables DP(J) and DQ(J) and the polynomials in P and Q. Then the FORTRAN statement

```
DG(J) = (DP(J) * Q - P * DQ(J))/(Q * Q)
```

can be incorporated in a subsequent DO-loop to obtain the $\frac{\partial g}{\partial x_j}$. This is the purpose of the copy function in the program, since the definitions of P and Q are required as well as their derivatives.

This program, written for the PDP-11 in FORTRAN, was designed for differentiating a commonly used class of polynomial functions when a differentiable function can be written in a form such as

$$f(x) = \sum_{i=1}^{m} c_i \prod_{j=1}^{k} x_j^{n_j}$$

where m, k are integer constants and c_i, n_j might take on both real and integer, positive and negative values.

The differentiation program, denoted as DIF, is organized as a main program with three fundamental files being involved. The numerical identifiers for these files are in the variables called IN, IFIN, IOUT.

As a result of running DIF, a sequence of formulae in the form of FORTRAN expressions are written to the file IOUT which represents the derivatives of the given functions with respect to the given variables. Obviously, after the file IOUT is created it can be compiled and run as an independent program (or subroutine). Consequently, any legal FORTRAN program name can be used to denote this file.

To facilitate the program's use, the section "outer specification for DIF" follows below. It is intended for those who want to use the program and are not interested in the programming problems. The rest of the document is intended for those who are either interested in the method proposed or perhaps might wish to change the program for some specific problems.

Outer Specifications for DIF

The functions to be differentiated are supposed to be defined in the form of conventional FORTRAN expressions, which are written to the file IN. This may be an executable program itself or simply a set of function definitions.

The identification of the beginning of a specific function is organized as a comment card to be followed by the expression mentioned above. The comment card must be written in the following form:

```
(col)
1    5
CFN, f
```

where the first four characters are used as the beginning identifier and f is a sequence of maximum four characters, which must

include a left-hand parenthesis if the function name is indexed. There is no restriction to the dimension size of the provided functions. Any number of continuation lines can be used to specify a given function. (A maximum of 400 nonblank characters is currently in effect. This is readily changed by enlarging a dimension parameter and recompiling DIF).

To identify the end of the appropriate function expression, it is followed by an ordinary comment card where only the character in the first column is important.

The program requires that all the functions be specified in an analogous way in succession. In the following example of IN file, three function, F, G, G3 are supposed to be given with function G to be indexed.

Example 1:

```
(col)
1
CFN,F
     F = -30.*X(1)**(-3.54)+10.*X(1)**(-2)*X(2)**2*X(3)
       + 40.*X(1)*X(2)-39.*X(2)**2
C
CFN,G(1)
     G(1) = 20.*X(1)**2*X(2)**3*X(3)**2-30.*X(1)*X(2)
C
CFN,G(2)
     G(2) = 10.*X(1)**4*X(2)**5-20.*X(1)**2*X(2)**2
C
CFN,G3
     G3 = 45.*X(1)**3-24*X(1)**2*X(2)**3*X(3)
```

All the variables X(1),X(2),...,X(N) with respect to some of which the differentiation is to be performed, are denoted as illustrated

and X might be a sequence of maximum three characters, the first of which must be X.

The file IFIN controls the sequence of actions. All information about the functions to be differentiated and variables with respect to which the differentiation is to be performed is specified by appropriate control statements. All existing lines of this file are copied verbatim to IOUT, interspersed as appropriate with the output from DIF. This facility enables one to write to IOUT all additional information, for instance to define the dimensionality of variables handled and provide any printing wanted.

In IFIN file, single statements of the following form appear

1

CDIF~~v($\hat{I}$,$\hat{J}$) = f($\hat{I}$), $\hat{I}$ = $\hat{i}$, $\hat{m}$, J = $\hat{j}$, $\hat{k}$

where $\hat{x}$ indicates that x is optional, and ~ indicates a blank (blanks are ignored), and the whole expression represents the following:

The function f, if it is not indexed or alternatively, each of the functions f(I); I = i,i+1,...,m, if they are indexed, are supposed to be differentiated with respect to variables X(J); J = j,j+1,...,k. The result of the differentiation is written as the associated variables v(J), or alternatively v(I,J). If m does not appear the foregoing refers to the indexed function with the fixed index number. If k does not appear, the differentiation is performed with respect to only one variable X(j).

The above mentioned assumptions about f and X hold for this file. The variable V is identified by a symbolic name of one to four letters or numbers, the first of which must be a letter.

In the example of IFIN file that follows, each of the control statements is referred to each of the functions given in Example 1.

Example 2:

```
1
CDIF C(J) = F, J = 1,2
CDIF A (I,J) = G(I), I = 1,2, J = 1,3
CDIF C(J) = G3, J = 1,3  .
```

The first statement controls the creation of two functions, which will be evaluated as variables C(1) and C(2) as the results of the differentiation of function F (see Example 1) with respect to vari variables X(1), X(2) respectively. These created functions are written onto file IOUT during the running of program DIF. The second statement above controls the generation of functions to be evaluated in variables A(1,1), A(1,2), A(1,3) as the result of differentiation function G(1) with respect to variables X(1), X(2) and X(3) respectively. The variables A(2,1), A(2,2) and A(2,3) are associated with derivatives of the function G(2) with respect to variables X(1), X(2) and X(3). The last example is concerned with the differentiation of function G3 with respect to variables X(1), X(2) and X(3), and the results of the differentiation to be assigned to C(1), C(2) and C(3) respectively.

An example of IOUT created after running DIF subject to the files IFIN (Appendix 1) and IN (Appendix 2) is given in the accompanying Appendix 3.

Obviously, to make IOUT in full accord with accepted FORTRAN conventions, IFIN should contain all appropriate DIMENSION, END, STOP, WRITE, READ and FORMAT statements if any. Some of them are included for illustration in the example program (Appendix 1, 3).

For the source files involved, DIF, IN, IFIN, and IOUT, the following names are used: DIM.F, D.F, C.F and O.F. If for some

reason, one of the last three should be changed, it can easily be done, changing one of the variables NAMFIL, NAMFIN or NAMOUT in DIM.F. All the functions involved must occur in the same order in the IN and IFIN files, i.e. IN file is usually searched only in a forward direction. However, in case of error, if the function name written in one of the control statements in IFIN is not encountered in IN file, an error message is printed out and IN file is rewound. In this case, control transfers to process the next function in IFIN and the rest will be handled in the regular way.

The Method Used and DIF Flow Chart

For differentiating a polynomial function, the following method was found expedient. The whole expression is broken down into separate terms. A term is associated with the part of an expression confined between two successive plus or minus signs. In the first term, a plus sign might not appear; in the last term, the end of expression means the end for this term. Each term is then broken down into separate factors. A factor is associated with a part of the term confined between two successive multiplication signs. The first factor of each term is confined, however, between plus/minus and multiplication (or plus/minus, if the term consists of only one factor) signs. In a similar manner, the last factor of the term is situated between multiplication and plus/minus or end of expression. A similar mechanism is used to process both terms and factors with the difference that the expression derivative is defined as the sum of derivatives of the terms and the term derivative is defined as the product of factor derivatives. If the variable with respect to which the differentiation is to be performed, is not encountered at the term level,

this term is cancelled. If this variable is not encountered at the factor level, this factor without any changes is stored as a potential multiplier for the term derivative. If the factor contains such a variable and is written in a form X(I)**N, the potential multiplier for the term derivative is stored as N*X(I)**(N-1).

The first time a particular function to be processed is encountered, the whole expression (except blank characters) to define this function and written on the IN file is stored in the array ICH(660). The function index, if any, is stored as a variable NF. To define the end of the current expression, a blank character is written after it is finished. For any subsequent analysis associated with the same function name, and index, the stored expression is used instead of reading.

Below is given the flow chart of the program to create the derivatives of all allowable given functions with respect to the appropriate variables. Because of its size, this flow chart is broken into separate flow charts. The control information is indicated by the subsequent reading of control statements in IFIN.

After such a statement as

CDIF~~v(Î,Ĵ) = f(Î), Î = î, m̂, J = j, k̂

appears, the string of characters, denoted here by the letter v, is retained and is called LEFT(4). If the string contains less than four characters, the rest of LEFT is filled with blanks, for example:

C(J)	C~~~ to LEFT
AC1(I,J)	AC1~ to LEFT

The string of characters denoted above by the letter f, including the left-hand parenthesis if the function name is indexed, is retained in NAMF(4), similarly blank-filled if less than four characters. This string is later used for matching against similar ones in the IN file. For example:

F	F~~~ to NAMF
G1	G1~~ to NAMF
F10(I)	F10(to NAMF

Both j and k are integers. If k appears, then it must be greater than j. The value j is the index of the first variable with respect to which the function f is to be differentiated. If k appears, then the function is to be differentiated with respect to all variables with index j,j+1,...,k.

If the function is indexed, i indicates the index of the first function, with the function name stored in NAMF, to be differentiated with respect to all variables indicated by j and k if any. If m appears, all the functions f(i),f(i+1),...,f(m) are subsequently to be differentiated with respect to all given variables. Obviously, i and m are integers. If m appears then it must be greater than i. The current index value, taking on values i,i+1,...,m is stored as a variable NF and used along with NAMF for matching against similar ones in the IN file.

The flow chart for reading IFIN control statements is shown in Figure 1. As a result of this, NAMF is organized to store the function name with the last significant character "(" and a variable I9I is set to 2, if the function is indexed. If not, I9I = 1. i and m are converted into variables NUIBEG and NUIEND

to be used later as the initial and terminal parameters, respectively, to assign the control variable I276 (Figure 2) for a DO-loop, the execution of the range of which is repeated for each function name combined with its index, if any. If m does not appear, the variable NUIEND is assigned the value represented by NUIBEG. If i does not appear, the variables NUIBEG and NUIEND are assigned the value 1; j and k are converted into variables JBEG and JEND to be used later as the initial and terminal parameters to assign the control variable NAMX (Figure 2) for DO-loop, nested in the previous one with the execution range repeated for each given variable. If k does not appear, the variable JEND is assigned the value represented by JBEG.

After the current function to be differentiated and the variable, with respect to which differentiation is to be performed, are defined, the desired function is searched for. This process is shown in the flow chart (Figure 2). First of all, the coincidence of the given function name and its index value, if any, is checked against the one stored in the array ICHA and the variable NF respectively. If no match is found, the beginning of the given function f is searched by subsequent readings of IN file to find the comment statement

CFN, f .

The function name and index value for the function must coincide with one in the IFIN file. If this coincidence is reached, the subsequent process to read the function definition and execute the main body of the differentiation program becomes effective. If not, and an END statement is encountered, the next

function is handled after rewinding IN file and printing an appropriate error message.

An indication of using ICHA (K10 = 2) or alternatively reading of IN (K10 = 1) for subsequent execution of the main body of the differentiation program is recorded (see Figure 2).

The flow chart of the initial stage for differentiating the current function f(I276) with respect to a current variable x(NAMX) is shown in Figure 3.

The initialization work can be separated into four parts. The first one (4) is concerned with the control of the left-hand side expression and simultaneous setting of those initial assumptions which may be done before the right-hand side expression is processed. The second one (5) is associated directly with analyzing each term which is done in the beginning and repeated with each reading of a plus/minus sign. The third part (6) is connected with control of the factor, which is distinguished by reading a multiplication sign and by the term beginning. The last part (7) is related directly to raising to a power.

On the first step (4), if I was not encountered in the control statement (I9I = 1) the following expression is written to IOUT file:

```
      7
      V(J) = 0
```

If I was encountered (I9I = 2) the FORTRAN expression written to IOUT file is:

```
      7
      V(I,J) = 0
```

By the following process, the derivative of each factor is searched for under the assumption that the term considered involves a variable x(NAMX). Until the current term is finished, all derivatives of each factor are written to a temporary file ISCR as continuation multiplication statements to the initial expression:

$$7$$

$$DIFACT = 1$$

In the course of the current term process (5) the identification that a variable x(NAMX) is involved on any factor level is registered as a variable IXX = 1. If in none of the factors a variable x(NAMX) is encountered, IXX = 0 is not changed. In the course of the current factor process (6) the identification that the raising to a power is involved, is registered as a variable IST = 1. If a variable or coefficient without raising to a power is encountered, IST = 0.

The initial value of the current index value for X, NX = 0, is written. Obviously, NX is an integer number. The initial value of the exponent EXP is supposed (7) to be zero. The identification of the exponent as an integer number is registered as a variable K143 = 1, or alternatively, if the exponent is a real number, K143 = 2. The counter parameters for the current number characterize the number of subsequent significant digits (M), an indication of the current number as an integer (IDEC=-1), or alternatively, a real number (IDEC is equal to the number of significant digits before the point in this case), and last, the sign indication. If the current number is positive, ISIGN = 1, if it is negative, ISIGN = -1. Initially (7.3) M = 0, IDEC = -1, and ISIGN = 1.

After the initialization is performed the function comes into the process (Figure 4) as the separate characters IS. Some characters which carry double information are stored until the next step as a variable ISN. For instance, plus/minus being referred to the beginning of the next term simultaneously characterize the end of the current term and are associated with the final term handling (Figure 6). Similarly, if the character multiplication sign is encountered (IS=*), the next character is read and stored as a variable ISN to define whether it was a raising to a power (ISN=*) or the end of the current factor (ISN≠ *). The flow chart for this case is shown in Figure 5.

The greater part of the characters, however, are processed immediately after they are read. In case the variable ISN is different from zero, the reading process is omitted and the current character IS is set to be equal ISN. If ISN = 0, in accordance with the reading conditions specified in Figure 2, the current character is identified using ICHA(K10 = 2), or alternatively, reading the current line from IN file (K10 = 1), which is stored in ICHAR. The identification of the function definition is registered as the first comment card encountered by subsequent reading of IN file. The entire function definition except for blank and continuation characters is copied to ICHA and used for subsequent processing of the same function. In this case the end of function expression is identified as the first blank character.

For the first letter X encountered (IS = X) the whole string to denote a variable name followed by a left-hand parenthesis is stored in NAMMX(3), blank-filled if less than three characters. NAMMX is used later for output.

The function name with its index, if any, followed by the equality sign is ignored.

The current value of the variable index is to be found in NX and used for output associated with the current factor. For a new factor NX is updated.

When any digit or point is encountered, it is registered in the counter.

The indication of the end of an integer index value for any variable is accepted by the counter as the first right-hand parenthesis encountered after IS = X was registered. When evaluating the current factor features (Figure 5), if raising to a power was encountered IST = 1, the indication of the end of exponent is received by the counter. As this takes place, the output to create an appropriate FORTRAN continuation statement is successively subject to the existence or nonexistence of:

1) any variable involved in the current factor;
2) raising to a power;
3) specified variable with respect to which the differentiation is to be presented.

If no variable is involved under the current factor treatment clearly this comprises only the coefficient value. After this value TERM is computed using the counter it is output to the temporary file ISCR using the following FORMAT

```
FORMAT("&~~~~~*(",F10.5,")")
```

If the specified variable with respect to which the differentiation is to be performed is involved in the current factor

NX = NAMX and raising to a power was not encountered, the derivative of this factor is 1 and output is omitted. With raising to a power EXP, the derivative of the current factor is written to ISCR as a continuation line for the initial statement DIFACT = 1 using the following format

```
FORMAT("&~~~~~*(",F5.2,")*",3A1,"(",I2,")**(",F5.2,")")
```

with the list

```
EXP,(NAMMX(I),I=1,3), NX, (EXP-1)
```

This assumes EXP is a real number.

For simplicity, the output for the cases EXP-1 = 1 or EXP as an integer is not considered here separately.

If any other variable NX ≠ NAMX is encountered, then with the raising to a power EXP the derivative of the current factor is written to ISCR as a continuation line using the following format

```
FORMAT("&~~~~~*",3A1,"(",I2,")**(",F5.2,")")
```

with the list

```
(NAMMX(I),I=1,3), NX, EXP.
```

If NX ≠ NAMX with no raising to a power is processed, then the derivative of the current term is written using the following format

```
FORMAT("&~~~~~*",3A1,"(",I2,")")
```

with the list

```
(NAMMX(I),I=1,3), NX   .
```

After the current factor is terminated, control is transferred to an appropriate initialization for the next factor (6).

To identify the end of the current term a variable K240 is set to 2 if plus/minus or end of expression is encountered (Figure 6). The term is checked for the presence of the variable with respect to which the differentiation is performed. If this condition is met IXX = 1, the final analysis of the last factor is performed followed by the addition of the derivative of this term to the previous value V for the function derivative (Figure 7). Again, if V is one-dimensional (I9I = 1), the following record is written to the file ISCR:

```
7
V(J) = V(J) + DIFACT   .
```

If V is two-dimensional (I9I = 2), this takes the form:

```
7
V(I,J) = V(I,J) + DIFACT   .
```

The identification of the last record is written to the temporary file ISCR. Then all the records from the temporary file (after its rewinding) are transferred to the file IOUT. The temporary file is rewound again with the result that the next record associated with the next term is written from the beginning of the file ISCR.

If the variable, with respect to which the differentiation is performed, is not involved IXX $\neq$ 1 (Figure 6) in the current

term, all the information on this term is not essential. After rewinding ISCR, a check is made whether the whole expression has already been processed. If this condition is not met IEND $\neq$ 1, control is tranferred to the appropriate initialization for the next term (5). If the differentiation of the current function with respect to the given variable is accomplished IEND = 1, the next variable with respect to which the differentiation is to be performed, if any, is selected (3). Otherwise, the next function is looked for and so on.

Comparison of Results Obtained with Manual Derivative Preparation

For the examples taken from J. Abadie's program [1], the comparison was carried out between manually organized programs and those automatically obtained by DIF program.

In addition to undoubtedly greater ease of use in the application with automatic differentiation, program size along with compilation and running time were compared.

Some size and compilation time increases for automatically obtained programs should not be considered as a shortcoming of the method because the compilation is a single-shot process.

The difference in size ranges from 5324 to 6348, and from 5164 to 5620. The compilation time increased from 2.6 seconds to 4.9 seconds for the user and from 3.3 seconds to 4.2 seconds for the system.

Any increase in size and compilation time are dominated by practically the same running time for both cases. All the aforesaid indicates that the automatic differentiation is preferable to manually coded programs.

REFERENCES

[1] Colville, A.R. "A Comparative Study of Nonlinear Programming Codes", IBM New York, Scientific Center Report No. 320-2949, June 1968.

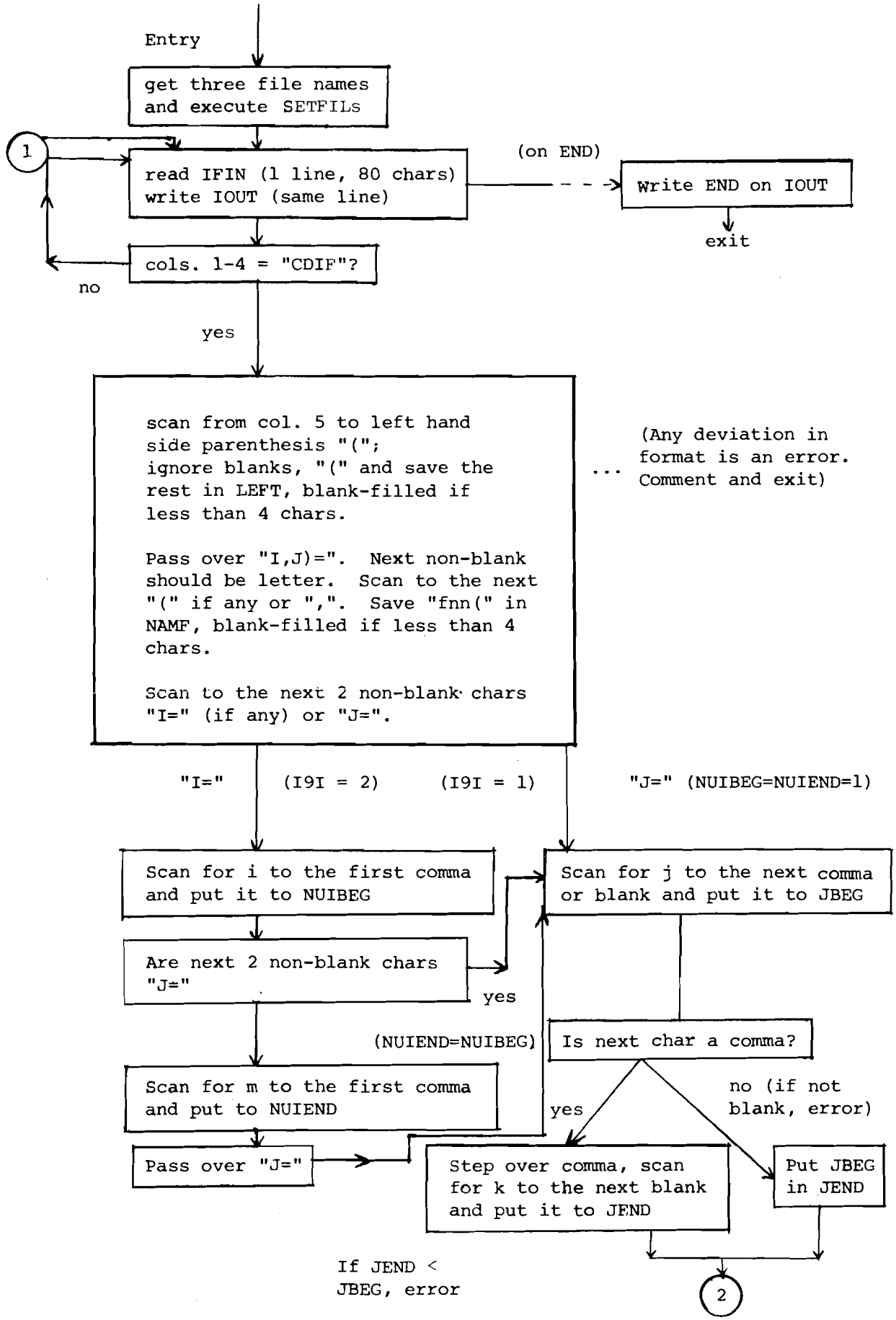

Entry
get three file names and execute SETFILs
1
read IFIN (1 line, 80 chars) write IOUT (same line)
(on END)
Write END on IOUT
exit
cols. 1-4 = "CDIF"?
no
yes
scan from col. 5 to left hand side parenthesis "("; ignore blanks, "(" and save the rest in LEFT, blank-filled if less than 4 chars.
Pass over "I,J)=". Next non-blank should be letter. Scan to the next "(" if any or ",". Save "fnn(" in NAMF, blank-filled if less than 4 chars.
Scan to the next 2 non-blank chars "I=" (if any) or "J=".
... (Any deviation in format is an error. Comment and exit)
"I=" (I9I = 2)
(I9I = 1) "J=" (NUIBEG=NUIEND=1)
Scan for i to the first comma and put it to NUIBEG
Scan for j to the next comma or blank and put it to JBEG
Are next 2 non-blank chars "J="
yes
(NUIEND=NUIBEG)
Is next char a comma?
Scan for m to the first comma and put to NUIEND
yes
no (if not blank, error)
Pass over "J="
Step over comma, scan for k to the next blank and put it to JEND
Put JBEG in JEND
If JEND < JBEG, error
2

Figure 1

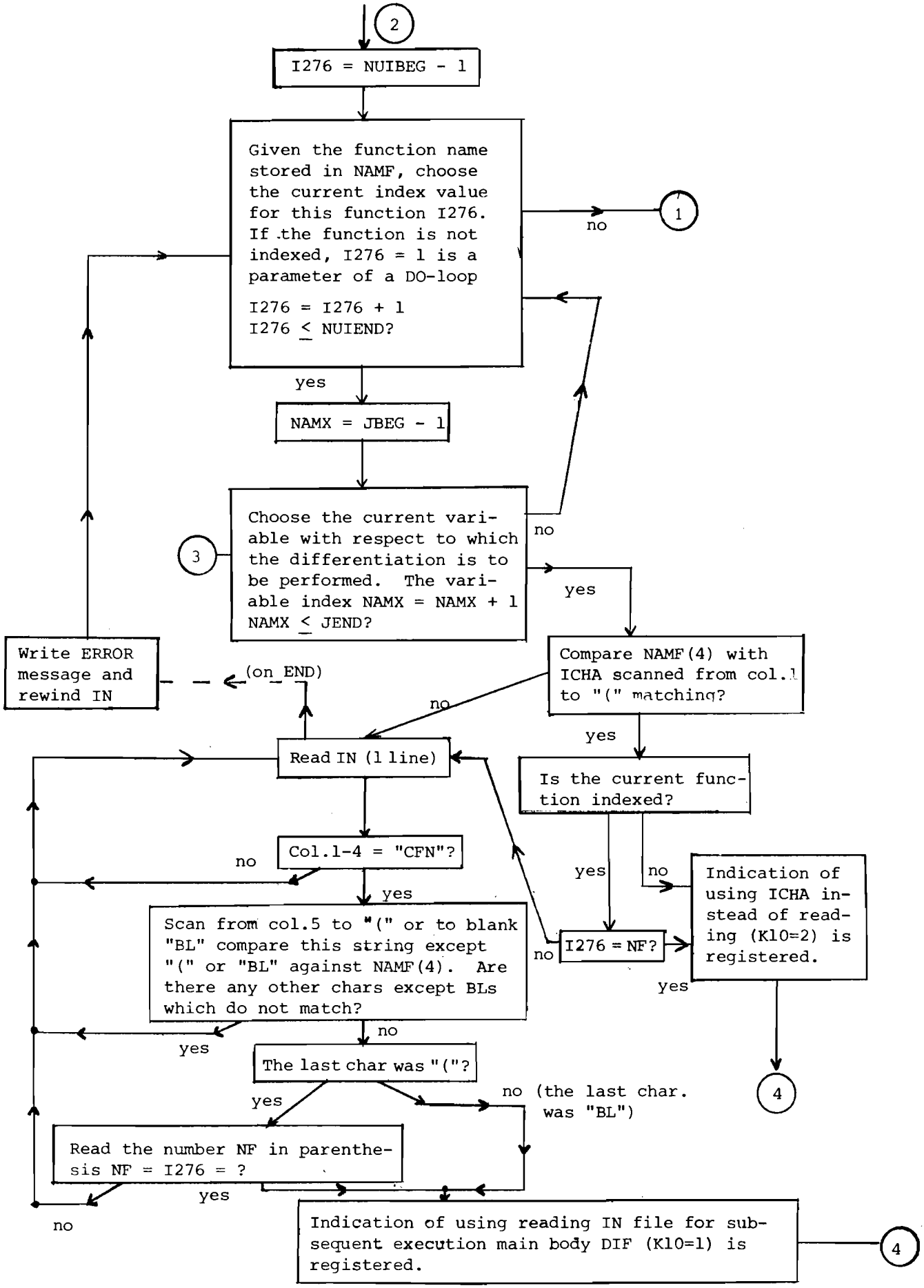

2
I276 = NUIBEG - 1
Given the function name stored in NAMF, choose the current index value for this function I276. If the function is not indexed, I276 = 1 is a parameter of a DO-loop
I276 = I276 + 1
I276 ≤ NUIEND?
no
1
yes
NAMX = JBEG - 1
Choose the current variable with respect to which the differentiation is to be performed. The variable index NAMX = NAMX + 1
NAMX ≤ JEND?
no
3
yes
Write ERROR message and rewind IN
(on END)
Compare NAMF(4) with ICHA scanned from col.1 to "(" matching?
no
yes
Read IN (1 line)
Is the current function indexed?
Col.1-4 = "CFN"?
no
yes
yes
no
Indication of using ICHA instead of reading (K10=2) is registered.
Scan from col.5 to "(" or to blank "BL" compare this string except "(" or "BL" against NAMF(4). Are there any other chars except BLs which do not match?
no
I276 = NF?
no
yes
yes
no
The last char was "("?
yes
no (the last char. was "BL")
4
Read the number NF in parenthesis NF = I276 = ?
yes
no
Indication of using reading IN file for subsequent execution main body DIF (K10=1) is registered.
4

Figure 2

(4)

Initialize

1. Indication that the function definition is not finished IEND = 0 (alternatively IEND = 1).
2. Indication that the equality sign was not encountered IEQ = 0 (alternatively IEQ = 1).
3. Indication of the current char. number to be read K = 0, and the normalized current char number in the current line KNOR = 0.
4. Indication of the current non-blank char number to be read K60 = 0.
5. WRITE to IOUT the initialization of V(I,J) if I9I = 1

```
          WRITE (IOUT, 121) (LEFT(I), I = 1,4), NAMX
121       FORMAT ("        "4A1,"(",  ,") = 0")
```

with result that the statement is written

```
          V(J) = 0
```

If I9I = 2

```
          WRITE (IOUT,123) (LEFT(I), I = 1,4), I276, NAMX
123       FORMAT ("       "4A1,"(",I3,",I3,") = 0"
```

with result that the statement is written

```
          V(I,J) = 0
```

(5)

1. An indication that beginning term does not include any factor with variable X(NAMX)
   IXX = 0 (is set to 1 if any).
2. An indication that the current term is being processed K240 = 1. Alternatively, if either the chars "+/-", referred to the beginning of the next term, or end of expression are encountered, that characerizes the end of the current term and K240 is set to 2.
3. An indication that the next char to read ISN = 0 (alternatively ISN = *, or +/-, if those are referred to the beginning of the nect term).
4. An indication that the current char IS is not * and K10 ≠ 1
   K194 = 1 (K194 is set to 2, when IS = * and K10 = 1
5. WRITE to ISCR the initial value of DIFACT = 1

```
          WRITE (ISCR, 28)
28        FORMAT ("     DIFACT = 1")
```

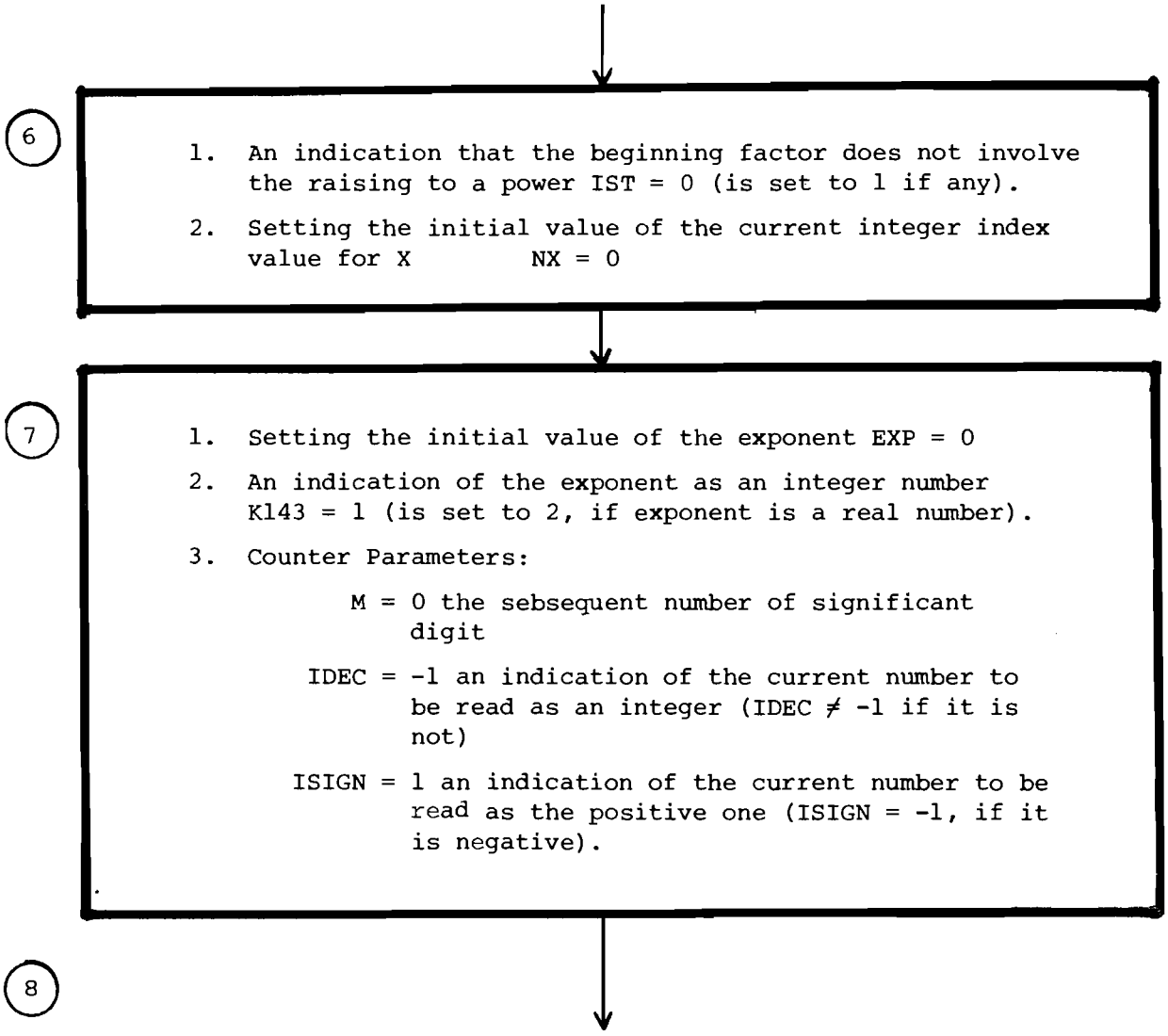

6
1. An indication that the beginning factor does not involve the raising to a power IST = 0 (is set to 1 if any).
2. Setting the initial value of the current integer index value for X NX = 0
7
1. Setting the initial value of the exponent EXP = 0
2. An indication of the exponent as an integer number Kl43 = 1 (is set to 2, if exponent is a real number).
3. Counter Parameters:
M = 0 the sebsequent number of significant digit
IDEC = -1 an indication of the current number to be read as an integer (IDEC ≠ -1 if it is not)
ISIGN = 1 an indication of the current number to be read as the positive one (ISIGN = -1, if it is negative).
8

Figure 3

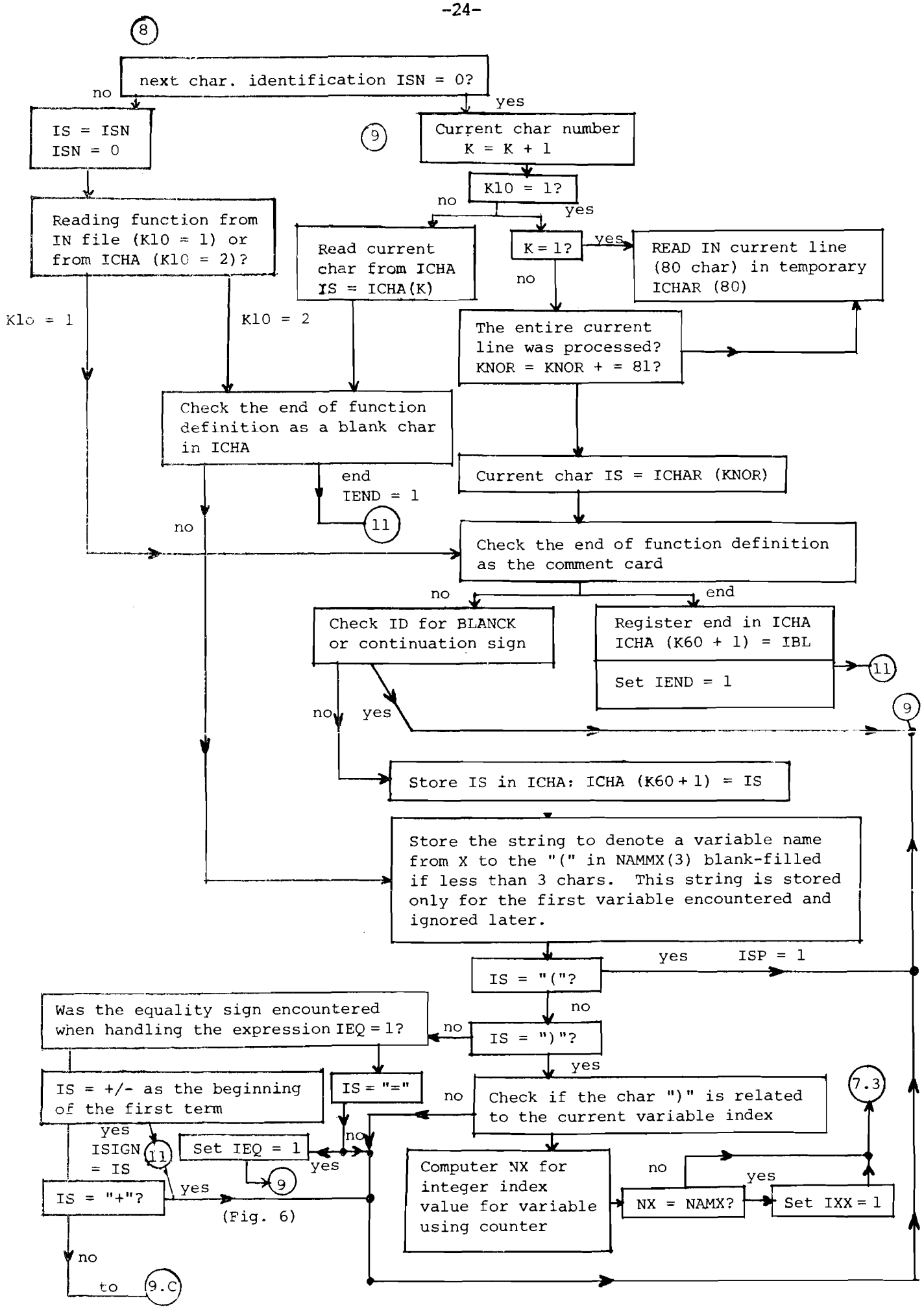
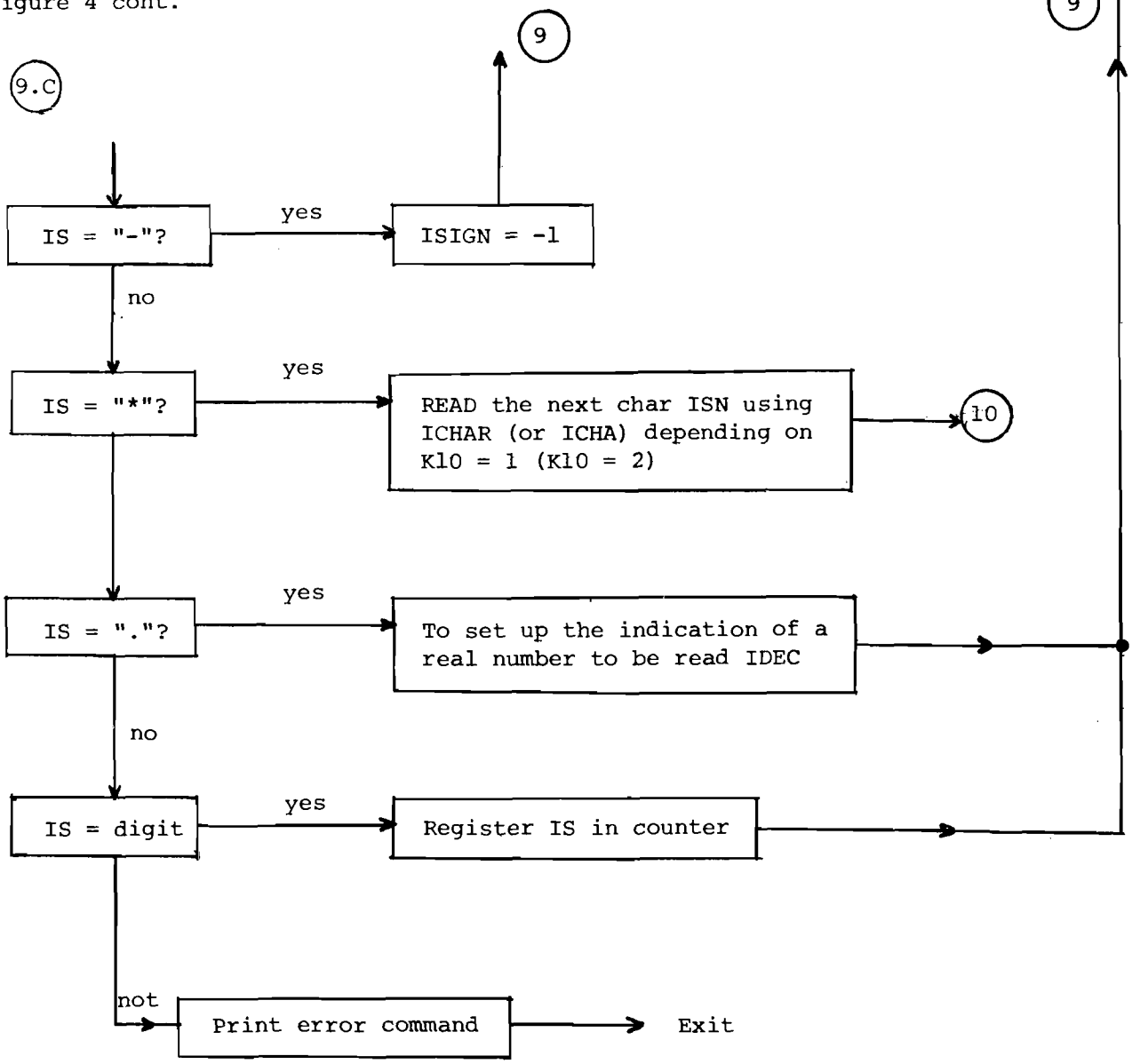

8
next char. identification ISN = 0?
no
yes
IS = ISN
ISN = 0
9
Current char number
K = K + 1
K10 = 1?
no
yes
Reading function from IN file (K10 = 1) or from ICHA (K10 = 2)?
Read current char from ICHA
IS = ICHA(K)
K = 1?
yes
READ IN current line (80 char) in temporary ICHAR (80)
no
K10 = 1
K10 = 2
The entire current line was processed?
KNOR = KNOR + = 81?
Check the end of function definition as a blank char in ICHA
end
IEND = 1
11
Current char IS = ICHAR (KNOR)
no
Check the end of function definition as the comment card
no
end
Check ID for BLANCK or continuation sign
Register end in ICHA
ICHA (K60 + 1) = IBL
Set IEND = 1
11
no
yes
9
Store IS in ICHA: ICHA (K60 + 1) = IS
Store the string to denote a variable name from X to the "(" in NAMMX(3) blank-filled if less than 3 chars. This string is stored only for the first variable encountered and ignored later.
IS = "("?
yes
ISP = 1
Was the equality sign encountered when handling the expression IEQ = 1?
no
IS = ")"?
no
yes
IS = +/- as the beginning of the first term
IS = "="
no
Check if the char ")" is related to the current variable index
7.3
yes
ISIGN = IS
11
Set IEQ = 1
no
yes
9
Computer NX for integer index value for variable using counter
no
yes
NX = NAMX?
Set IXX = 1
IS = "+"?
yes
(Fig. 6)
no
to
9.C

Figure 4

Figure 4 cont.



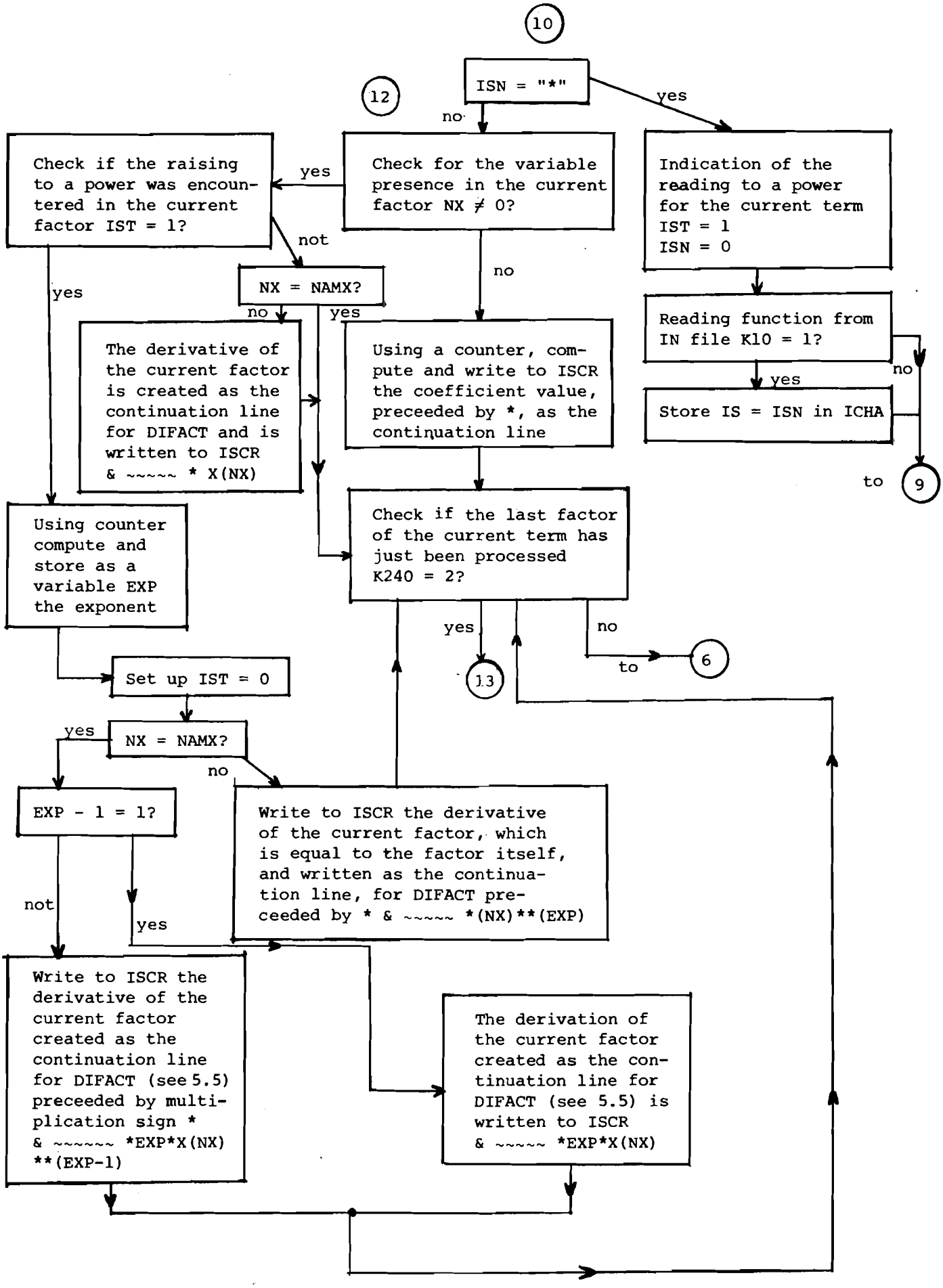

10
ISN = "*"
yes
no
12
Check for the variable presence in the current factor NX ≠ 0?
yes
Check if the raising to a power was encountered in the current factor IST = 1?
not
yes
Indication of the reading to a power for the current term
IST = 1
ISN = 0
NX = NAMX?
no
yes
no
Using a counter, compute and write to ISCR the coefficient value, preceeded by *, as the continuation line
Reading function from IN file K10 = 1?
yes
no
Store IS = ISN in ICHA
to 9
The derivative of the current factor is created as the continuation line for DIFACT and is written to ISCR
& ~~~~~ * X(NX)
Using counter compute and store as a variable EXP the exponent
Check if the last factor of the current term has just been processed
K240 = 2?
yes
no
13
to 6
Set up IST = 0
yes
NX = NAMX?
no
EXP - 1 = 1?
Write to ISCR the derivative of the current factor, which is equal to the factor itself, and written as the continuation line, for DIFACT preceeded by * & ~~~~~ *(NX)**(EXP)
not
yes
Write to ISCR the derivative of the current factor created as the continuation line for DIFACT (see 5.5) preceeded by multiplication sign *
& ~~~~~~ *EXP*X(NX)
**(EXP-1)
The derivation of the current factor created as the continuation line for DIFACT (see 5.5) is written to ISCR
& ~~~~~ *EXP*X(NX)

Figure 5

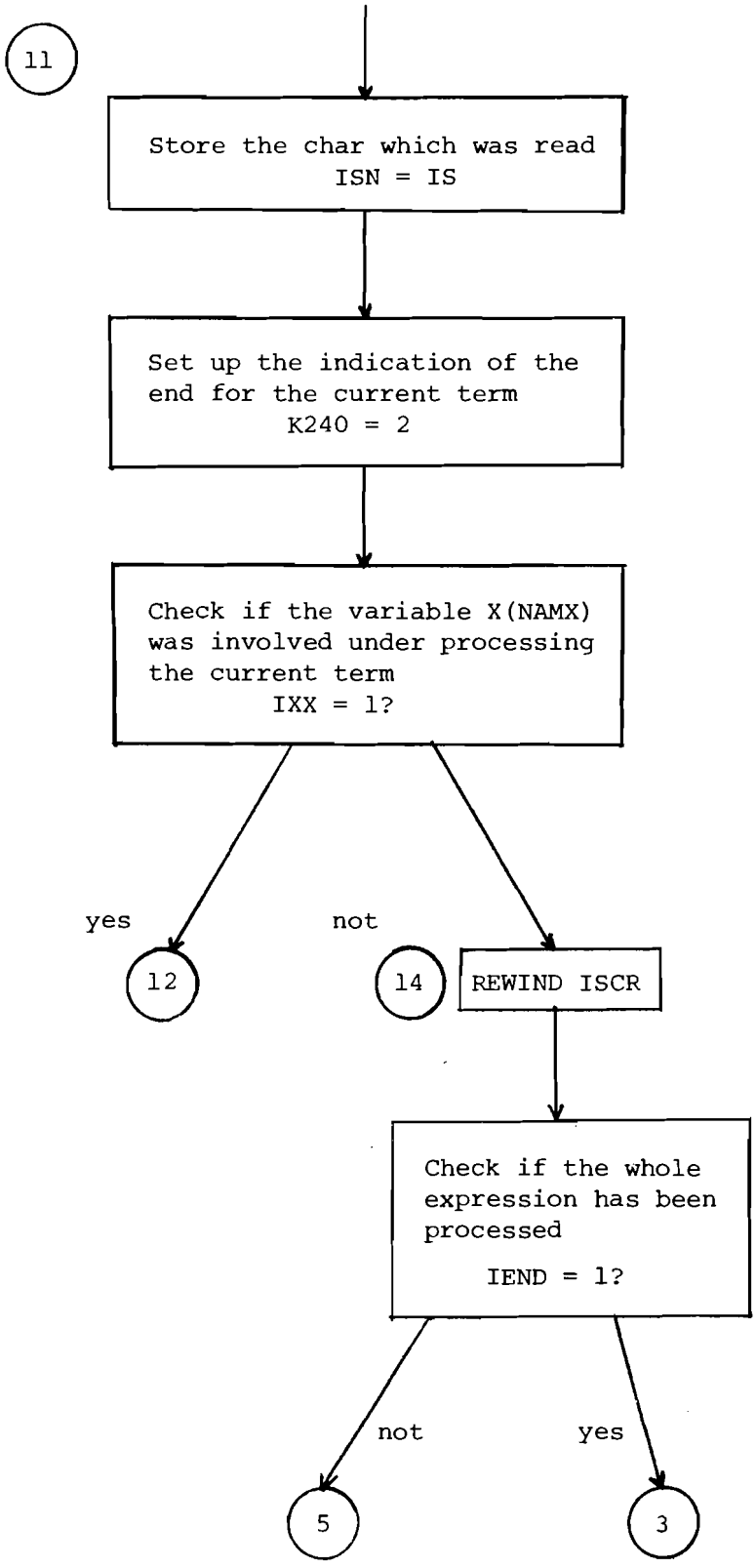

11
Store the char which was read
ISN = IS
Set up the indication of the
end for the current term
K240 = 2
Check if the variable X(NAMX)
was involved under processing
the current term
IXX = 1?
yes
not
12
14
REWIND ISCR
Check if the whole
expression has been
processed
IEND = 1?
not
yes
5
3

Figure 6

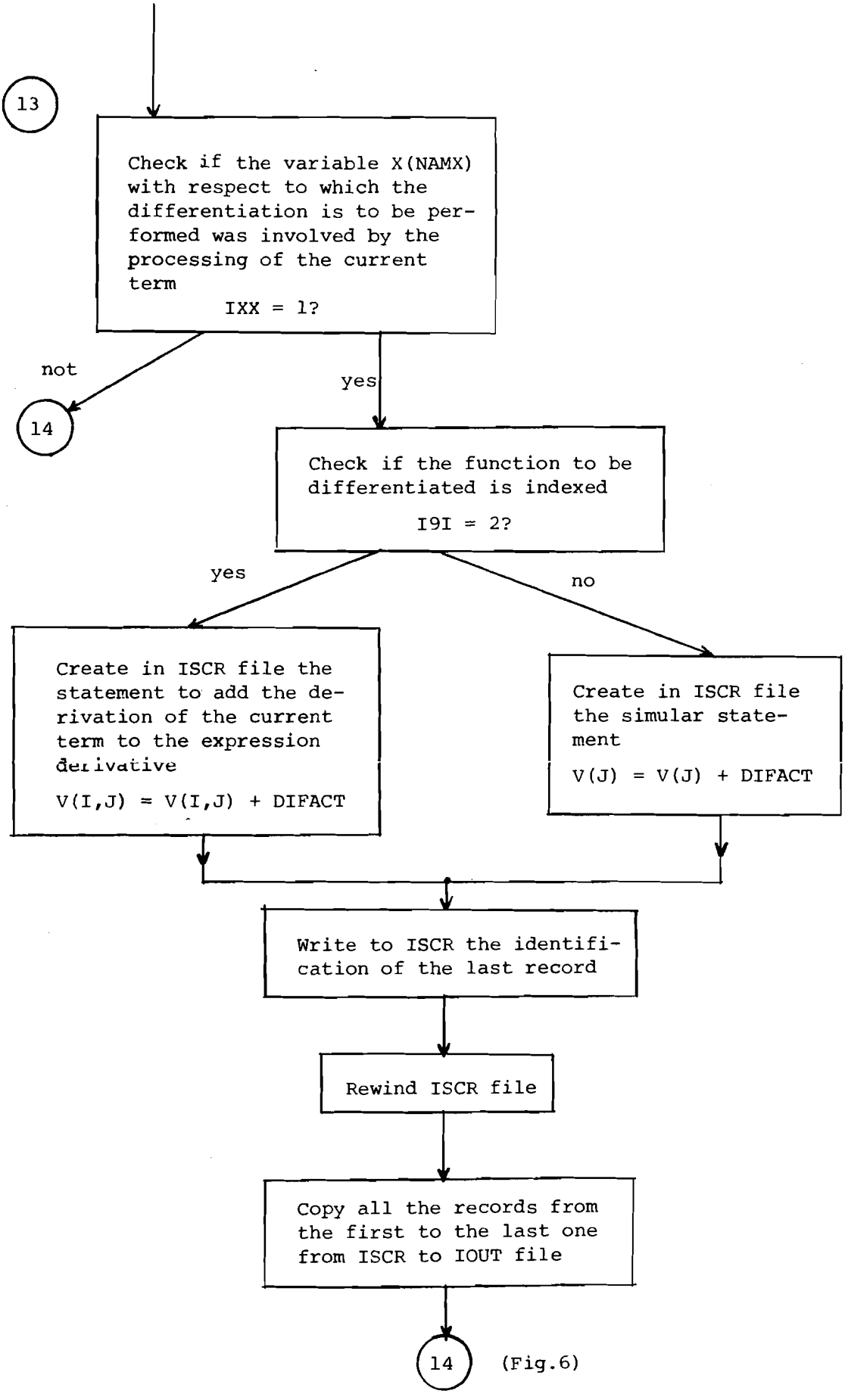

13
Check if the variable X(NAMX) with respect to which the differentiation is to be performed was involved by the processing of the current term
IXX = 1?
not
yes
14
Check if the function to be differentiated is indexed
I9I = 2?
yes
no
Create in ISCR file the statement to add the derivation of the current term to the expression derivative
V(I,J) = V(I,J) + DIFACT
Create in ISCR file the simular statement
V(J) = V(J) + DIFACT
Write to ISCR the identification of the last record
Rewind ISCR file
Copy all the records from the first to the last one from ISCR to IOUT file
14
(Fig.6)

Figure 7

APPENDIX I

```
MAR 16 12:59 1976  C,F PAGE 1


      DIMENSION X(5),A(5,5),C(5)
      IOUT=4
      NAMF=4HIOU
      CALL SETFIL(IOUT,NAMF)
      DO 10 I=1,5
10    XC(I)=I+1
CDIF C(J)=F,J=1,2
      WRITE(IOUT,20)(C(I),I=1,2)
20    FORMAT("DF/DX(1) =",F15.5,", DF/DX(2) =",F15.5)
CDIF A(I,J)=G(I),I=1,2,J=1,3
CDIF C(J)=G3,J=1,3
      STOP
      END
```

```
      DIMENSION C(5),A(5,5),X(5)
CFN,F
      F=-30.*X(1)**(-3.5)+10.*X(1)**(-2)*X(2)**2*X(3)+40.*X(1)*X(2)-39.*X(2)**2
C
CFN,G(1)
      G(1)=20.*X(1)**2*X(2)**3*X(3)**2-30.*X(1)*X(2)
C
CFN,G(2)
      G(2)=10.*X(1)**4*X(2)**5-20.*X(1)**2*X(2)**2
C
CFN,G3
      G3=45.*X(1)**3-24*X(1)**2*X(2)**3*X(3)
C
      END
```

```
MAR 16 13:06 1976  O.F PAGE 1


      DIMENSION X(5),A(5,5),C(5)
      IOUT=4
      NAMF=4HIOU
      CALL SETFIL(IOUT,NAMF)
      DO 10 I=1,5
10    XC(I)=I+1
CDIF C(J)=F,J=1,2
      C    (  1) =0
      DIFACT=1
&     *( -30.00000)
&     *(-3.54)*X   ( 1)**(-4.54)
      C    (  1) =C    (  1)+DIFACT
      DIFACT=1
&     *(  10.00000)
&     *(-2.00)*X   ( 1)**(-3)
&     *X   ( 2)**( 2)
&     *X   ( 3)
      C    (  1) =C    (  1)+DIFACT
      DIFACT=1
&     *(  40.00000)
&     *X   ( 2)
      C    (  1) =C    (  1)+DIFACT
      C    (  2) =0
      DIFACT=1
&     *(  10.00000)
&     *X   ( 1)**(-2)
&     *( 2.00)*X   ( 2)
&     *X   ( 3)
      C    (  2) =C    (  2)+DIFACT
      DIFACT=1
&     *(  40.00000)
&     *X   ( 1)
      C    (  2) =C    (  2)+DIFACT
      DIFACT=1
&     *( -39.00000)
&     *( 2.00)*X   ( 2)
      C    (  2) =C    (  2)+DIFACT
      WRITE(IOUT,20)(C(I),I=1,2)
20    FORMAT("DF/DX(1) =",F15.5,", DF/DX(2) =",F15.5)
CDIF A(I,J)=G(I),I=1,2,J=1,3
      A    (  1,  1)=0
      DIFACT=1
&     *(  20.00000)
&     *( 2.00)*X   ( 1)
&     *X   ( 2)**( 3)
&     *X   ( 3)**( 2)
      A    (  1,  1) =A    (  1,  1)+DIFACT
      DIFACT=1
&     *( -30.00000)
&     *X   ( 2)
      A    (  1,  1) =A    (  1,  1)+DIFACT
      A    (  1,  2)=0
      DIFACT=1
&     *(  20.00000)
&     *X   ( 1)**( 2)
&     *( 3.00)*X   ( 2)**( 2)
```

```
MAR 16 13:06 1976  D.F PAGE 2

&       *X  ( 3)**( 2)
        A   ( 1,  2) =A   ( 1,  2)+DIFACT
        DIFACT=1
&       *( -30.00000)
&       *X  ( 1)
        A   ( 1,  2) =A   ( 1,  2)+DIFACT
        A   ( 1,  3)=0
        DIFACT=1
&       *(  20.00000)
&       *X  ( 1)**( 2)
&       *X  ( 2)**( 3)
&       *( 2.00)*X  ( 3)
        A   ( 1,  3) =A   ( 1,  3)+DIFACT
        A   ( 2,  1)=0
        DIFACT=1
&       *(  10.00000)
&       *( 4.00)*X  ( 1)**( 3)
&       *X  ( 2)**( 5)
        A   ( 2,  1) =A   ( 2,  1)+DIFACT
        DIFACT=1
&       *( -20.00000)
&       *( 2.00)*X  ( 1)
&       *X  ( 2)**( 2)
        A   ( 2,  1) =A   ( 2,  1)+DIFACT
        A   ( 2,  2)=0
        DIFACT=1
&       *(  10.00000)
&       *X  ( 1)**( 4)
&       *( 5.00)*X  ( 2)**( 4)
        A   ( 2,  2) =A   ( 2,  2)+DIFACT
        DIFACT=1
&       *( -20.00000)
&       *X  ( 1)**( 2)
&       *( 2.00)*X  ( 2)
        A   ( 2,  2) =A   ( 2,  2)+DIFACT
        A   ( 2,  3)=0
CDIF C(J)=G3,J=1,3
        C   ( 1) =0
        DIFACT=1
&       *(  45.00000)
&       *( 3.00)*X  ( 1)**( 2)
        C   ( 1) =C   ( 1)+DIFACT
        DIFACT=1
&       *( -24.00000)
&       *( 2.00)*X  ( 1)
&       *X  ( 2)**( 3)
&       *X  ( 3)
        C   ( 1) =C   ( 1)+DIFACT
        C   ( 2) =0
        DIFACT=1
&       *( -24.00000)
&       *X  ( 1)**( 2)
&       *( 3.00)*X  ( 2)**( 2)
&       *X  ( 3)
        C   ( 2) =C   ( 2)+DIFACT
        C   ( 3) =0
```

```
MAR 16 13:06 1976  O.F PAGE 3

      DIFACT=1
&     *( -24.00000)
&     *X   ( 1)**( 2)
&     *X   ( 2)**( 3)
      C    (  3) =C    (  3)+DIFACT
      STOP
      END
```