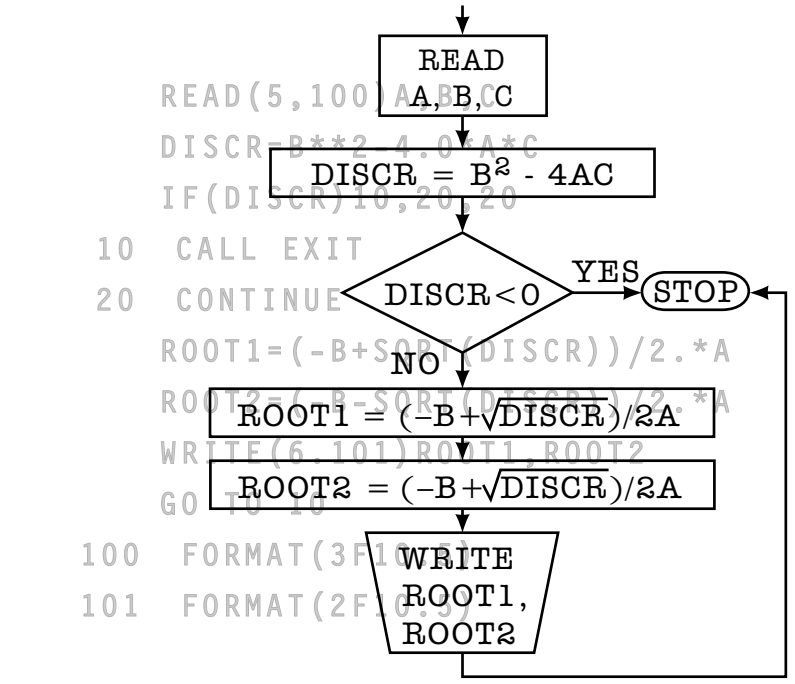
PHYSNET

MISN-0-347

ADVANCED FEATURES OF FORTRAN



Project PHYSNET Physics Bldg. Michigan State University East Lansing, MI

ADVANCED FEATURES OF FORTRAN

by
Robert Ehrlich
George Mason University

Title: **Advanced Features of Fortran**

Author: Robert Ehrlich, George Mason University, Physics Department

Version: 2/1/2000 Evaluation: Stage 0

Length: 1 hr; 20 pages

Input Skills:

1. Vocabulary: double precision, round-off error, triple precision (MISN-0-370).
2. Write (or modify) and run simple programs using elementary FORTRAN, such as the `COMMENT`, `IF`, `GOTO`, `CALL EXIT`, `CONTINUE`, `END`, `READ`, and `WRITE` statements (MISN-0-346).

Output Skills (Knowledge):

K1. Vocabulary: echo check, matrices, nested loops.

Output Skills (Rule Application):

R1. Recognize and write valid FORTRAN statements involving arrays with multiple and variable indices, including the use of the `DIMENSION` statement.

R2. Write valid `DO` loops (simple and nested), and recognize and correct invalid `DO` loops.

R3. Recognize and write valid `FORMAT` statements, including the `F`, `E`, `I`, `H`, `A`, and `X` formats, and use the `FORMAT` statement for carriage control.

R4. Use the `DATA` statement to initially assign the value of any variable.

R4. Use the `TYPE` statement to set the precision of a variable, declare a complex variable, or override the standard FORTRAN designations of real and integer variables.

Output Skills (Project):

P1. Enter and run "canned" programs to numerically calculate the maximum of one- and two-variable functions.

External Resources (Required):

1. A computer with FORTRAN.

THIS IS A DEVELOPMENTAL-STAGE PUBLICATION
OF PROJECT PHYSNET

The goal of our project is to assist a network of educators and scientists in transferring physics from one person to another. We support manuscript processing and distribution, along with communication and information systems. We also work with employers to identify basic scientific skills as well as physics topics that are needed in science and technology. A number of our publications are aimed at assisting users in acquiring such skills.

Our publications are designed: (i) to be updated quickly in response to field tests and new scientific developments; (ii) to be used in both classroom and professional settings; (iii) to show the prerequisite dependencies existing among the various chunks of physics knowledge and skill, as a guide both to mental organization and to use of the materials; and (iv) to be adapted quickly to specific user needs ranging from single-skill instruction to complete custom textbooks.

New authors, reviewers and field testers are welcome.

PROJECT STAFF

Andrew Schnepp	Webmaster
Eugene Kales	Graphics
Peter Signell	Project Director

ADVISORY COMMITTEE

D. Alan Bromley	Yale University
E. Leonard Jossem	The Ohio State University
A. A. Strassenburg	S. U. N. Y., Stony Brook

Views expressed in a module are those of the module author(s) and are not necessarily those of other project participants.

ADVANCED FEATURES OF FORTRAN

by

Robert Ehrlich
George Mason University

1. Overview

In another module some of the basic elements of the FORTRAN language are described.[1] Using those basic elements it should be possible to write and run simple FORTRAN programs. More complex programs require a knowledge of some of the features described in the present module, especially the important `DO` and `DIMENSION` statements. These statements and others will be illustrated at the end of this module in a program that computes values for arbitrary functions of one and two variables and allows you to find the function's maximum value.

2. Arrays and the `Dimension` Statement

2a. Definition of an Array. It is sometimes convenient when dealing with a number of closely related variables not to give each one a different name, but to distinguish among them by an index. This is entirely analogous to the use of subscripts in mathematical notation. In FORTRAN, indices must be enclosed in parentheses. For example, these statements might appear in a program:

```
Q(1) = 2.0
Q(2) = 5.2
```

The two variables `Q(1)` and `Q(2)` are called "the elements of the array `Q`." The parentheses are crucial: if two variables are named `Q1` and `Q2` then they do not thereby constitute elements of an array and they do not thereby have any relationship to either the array `Q` or to each other.

2b. Storage of Arrays. Each element of an array corresponds to a different location in the computer memory. Normally the elements of an array are stored in a contiguous block of memory locations. Thus, the index in parentheses refers to a particular memory location within the block. For this reason, the index must always be a positive integer that

[1] See "Review of Elementary FORTRAN" (MISN-0-346).

does not exceed the total number of memory locations within the block reserved for the particular variable.

2c. The `DIMENSION` Statement. The number of locations to be reserved for each array must be specified in a `DIMENSION` statement, usually placed near the beginning of a program. As an example, consider this `DIMENSION` statement:

```
DIMENSION X(10), Q(5), Z(1000)
```

This statement specifies that 10 memory locations should be reserved for the array `X`, 5 for the array `Q`, and 1000 for the array `Z`. In this case, it would therefore be perfectly legitimate to have the following array elements referred to in a program which contained this `DIMENSION` statement: `X(8)`, `Q(5)`, `Z(982)`. However, any references to `X(11)`, `Q(9)`, `Z(1027)`, `X(0)`, `Q(-2)`, or `Z(3.64)` would not be legitimate.

2d. Variable Array Indices. The indices used to refer to an element of an array may be variables or mathematical expressions as well as constants. For example, the following are completely valid statements:

```
X(J) = 2
X(2*K+1) = 10
```

provided that `J` and `K` have been assigned numerical values in some previous statements and that the values of the expressions in parentheses are within the ranges specified by the appropriate `DIMENSION` statements. On many computers the most complicated expression permitted for the index of an array is $c_1 \times v + c_2$, where c_1 and c_2 represent two fixed point constants and v represents a fixed point variable. Thus, for example, the following statement would not be permitted:

```
X(K**2+2*K) = 10
```

2e. Multiple Array Indices. Just as variables in mathematics may have more than one subscript, so also variables in FORTRAN may have more than one index. The number of indices specifies the number of dimensions of the array. For example, the elements of a two-dimensional array `Y` might include `Y(1,1)`, `Y(1,6)`, `Y(2,3)`, and `Y(3,5)`. The size of arrays of more than one dimension must also be specified in a `DIMENSION` statement. The statement

```
DIMENSION Y(4,10)
```

specifies that the first index of the two-dimensional array `Y` can take on values $1, 2, 3, 4$, and the second index can take on values $1, 2, ..., 10$. The statement therefore specifies that a block of $4 \times 10 = 40$ memory locations be reserved for the array `Y`. One-dimensional arrays are sometimes called "vectors" and two-dimensional arrays are called "matrices." On many computers arrays of more than three dimensions are not permitted.

3. The `Do` Statement

3a. `DO` Loops. A loop in a program can be created using the `GOTO` and `IF` statements: an alternative method is to use the `DO` statement. We can illustrate the meaning of the `DO` statement through an example:

```
DO 5 K = 7,87
```

This statement instructs the computer to execute repeatedly all the statements that follow, up to and including statement number 5, setting the loop index `K` equal to 7 initially and increasing it by one each time through the loop, until it reaches 87, which is the last time through the loop. Thus the loop is to be executed 81 times. From the meaning of the `DO` statement, it should be clear that the following two sets of instructions are equivalent:

```
C  EXAMPLE OF A LOOP C EXAMPLE OF A LOOP
C  USING AN IF STATEMENT C USING A DO STATEMENT
C                                C
      N=6                            DO 2 N=7,100
   1  N=N+1                            .
        .                              .
        .                              .
        .                           2  CONTINUE
      IF(N-100)1,1,2                   .
   2  CONTINUE                         .
        .                              .
        .
```

3b. Sample Program Using a `DO` Loop. To illustrate how a `DO` statement might be used in a program, we show a program which reads

in numerical grades for some number of students and computes an average grade. As indicated in this example, the upper and/or lower limit on the loop index specified in the `DO` statement may be either a variable or a constant. Many `COMMENT` statements have been used to explain the role of the statements in this program:

```
C  READ A NUMERICAL VALUE FOR N, THE NUMBER OF
   GRADES:
      READ(2,100)N
C  SET THE SUM OF THE GRADES TO ZERO INITIALLY:
      SUM=0.
C  REPEAT ALL INSTRUCTIONS UP TO STATEMENT NO.\,10,
C  FOR J=1,2...,N:
      DO 10 J=1,N
C  READ A DATA CARD CONTAINING A GRADE:
      READ(2,101)GRADE
C  ADD THIS GRADE TO THE SUM:
  10  SUM=SUM+GRADE
C  LET FN BE THE FLOATING POINT EQUIVALENT OF N:
      FN=N
C  COMPUTE THE AVERAGE GRADE:
      AVG=SUM/FN
C  PRINT THE RESULT:
      WRITE(3,102)AVG
C  STOP THE PROGRAM.
      CALL EXIT
 100  FORMAT(I10)
 101  FORMAT(F10.5)
 102  FORMAT(19H THE AVERAGE GRADE=,F10.5)
      END
```

3c. Premature Exit from a `DO` Loop. If a loop defined by a `DO` statement contains one or more `IF` statements, the program may execute the loop fewer times than is specified in the `DO` statement. An example of such a premature exit from the loop is given in the program below. Note that when a premature exit occurs, the latest value of the loop counter is saved and may be used in another part of the program.

```
C   THIS PROGRAM WAS WRITTEN TO READ 100 CARDS, EACH
C   CONTAINING A NUMBER, AND TO INDICATE WHICH IS THE
C   FIRST CARD OUT OF SEQUENCE (THAT IS, THE FIRST
C   CARD THAT HAS A NUMBER SMALLER THAN THE PRECEDING
```

```
C   CARD).
C
      DIMENSION NUM(100)
      READ(2,100)NUM(1)
      DO 10 J=2,100
      READ(2,100)NUM(J)
C
C   THE FOLLOWING STATEMENT CAUSES A JUMP OUT OF THE
C   LOOP (TO STATEMENT 20) IF NUM(J) IS LESS THAN
C   NUM(J−1), THE PREVIOUS NUMBER:
      IF(NUM(J)−NUM(J−1))20,10,10
 10   CONTINUE
      GO TO 30
 20   WRITE(3,100)J
 30   CALL EXIT
100   FORMAT(I10)
      END
```

3d. Nested `DO` Loops. In many programs a loop may be contained within another loop. Such "nested loops" can easily be created using `DO` statements as indicated in this example:

```
                C     AN EXAMPLE OF NESTED DO LOOPS
                C
          ----------    DO 20 J=1,5
        |               .
        |               .
        |               .
        |         --    DO 10 K=1,25
        |       |       .
outer   |  inner|       .
loop    |  loop |       .
        |       |-- 10  CONTINUE
        |               .
        |               .
        |               .
        | ----------  20 CONTINUE
```

In this example the computer is instructed to execute the outer loop 5 times: `J` $= 1, 2, ..., 5$. Each time through the outer loop it is instructed to execute the inner loop 25 times: `K` $= 1, 2, ..., 25$. Thus the inner loop is executed a total of $5 \times 25 = 125$ times. Any number of loops may

be contained within a loop: this includes the possibility of loops within loops within loops. Some possibilities are shown schematically in Figure 1. Figure 1c indicates that nested loops may have a single last statement in common.

3e. Improper `DO` Loops. An ambiguous structure such as the one shown below is not allowed, since neither loop is contained within the other.

```
      C THIS FORM IS NOT LEGITIMATE!
      C
 -------      D0 20 J=1,5
|     --     DO 10 K=1,25
|    |          .
|    |          .
|    |          .
| ---|-- 20    CONTINUE
     |          .
     |          .
     |          .
     |-- 10    CONTINUE
                .
                .
```

Using the loop index to the left of an equal sign anywhere within a loop is also not allowed, since this redefines its value and the loop is not executed the proper number of times. For a similar reason the variable names used as loop indices in nested `DO` loops must be different. However, instructions outside of a loop may use the variable name used as the loop index in some different way.

In the following examples, (a) and (b) are not legitimate `DO` loops, while (c) and (d) are legitimate.

```
C  AN IMPROPER USE OF C AN IMPROPER USE OF
C  THE LOOP INDEX WITHIN C THE SAME LOOP INDEX
C  A DO LOOP           C  IN NESTED DO LOOPS
      DO 10 J=1,3           DO 10 J=1,5
        .                     .
        .                     .
      J=1                   DO 6 J=1,3
 10   CONTINUE                .
```

```
      .                          .
      .                      6   CONTINUE
      .                     10   CONTINUE
      .                          .
        (a)                        (b)
```

```
C  IT IS OK TO USE THE C IT IS OK TO USE THE
C  SAME VARIABLE AS A C LOOP COUNTER
C  LOOP INDEX AS      C  VARIABLE NAME IN
C  LONG AS THE LOOPS C SOME OTHER WAY IN
C  ARE NOT NESTED.    C  INSTRUCTIONS THAT ARE
       DO 10 J=1,10   C  OUTSIDE THE LOOP.
         .            C
         .                 DO 10 J=1,10
   10  CONTINUE             .
       DO 20 J=1,30         .
         .               10  CONTINUE
         .                   J=1
   20  CONTINUE             .
         .                  .
         .                  .
           (c)                (d)
```

4. Format Statements

4a. Common `FORMAT` Statements. A `FORMAT` statement contains a string of format specifications separated by commas. For example:

```
100 FORMAT(F11.5,E15.5,I6,29H ARE THE VALUES OF A,B,and N)
```

This statement contains four types of format specifications:

- `F` floating point variables.
- `E` floating point variables, expressed in exponential form.
- `I` integer (fixed point) variables.
- `H` specific strings of characters of any length, used to print labels or titles on the output.

The meaning of the four format specifications in the sample `FORMAT` statement is:

`F11.5`: specifies that the quantity to be read or written is a floating point number occupying 11 spaces and has five digits after the decimal

point. For example, -45.12347: in this case, two blank spaces must be added in front of the number, in order that it occupy a total of 11 spaces counting the sign and the decimal point. If the same number were written in `F6.2` format it would appear as -45.12, with no leading blanks. Finally, if we attempted to write the number in `F5.2` format, we might get `*****`, meaning "it can't be done."

`E15.5`: specifies that the quantity to be read or written is a floating point number in exponential (power of ten) form. It occupies a total of 15 spaces with 5 digits after the decimal point, for example,

$0.89993E-12$ (which is 0.89993×10^{-12}).

Note that in this case four blank spaces must appear in front of the number in order that it occupy a total of 15 spaces.

`I6`: specifies that the quantity to be read or written is an integer which takes up 6 spaces, for example, -12345.

`29H` ARE THE VALUES OF `A`, `B`, AND `N`: the "`29H`" specifies that the 29 characters: "`ARE THE VALUES OF A, B, AND N`" are to be written exactly as they appear. Unlike the three preceding formats, the `H` format does not refer to any variable that appears in a `READ` or a `WRITE` statement; it is simply used to write out a specific string of characters. On some computers the characters to be written need only be enclosed in single quotation marks, avoiding the need for a character count, i.e., 29 in this case.

A `FORMAT` statement, as noted elsewhere[2], may be referred to by one or more `READ` or `WRITE` statements. Two other types of format specifications, the `A` format and the `X` format, are discussed in Section 5.

4b. Illustration of Common Format Specifications. Use of the `F`, `E`, `I`, and `H` format specifications is illustrated in this program:

```
C  ILLUSTRATION OF A FORMAT STATEMENT USING F,E,I,
C  AND H FORMAT SPECIFICATIONS
C
      A=4.0/3.0
      B=25000.
      N=67
      WRITE(3,100)A,B,N
```

[2]See "Review of Elementary FORTRAN" (MISN-0-346).

```
100  FORMAT(F10.5,E10.5,I4,29H ARE THE VALUES OF A,
     B, AND N)
     CALL EXIT
     END
```

When this program is executed, the values of `A`, `B`, and `N` will be written out on device number 3, according to the format specifications in `FORMAT` statement 100. Assuming that device number 3 is the printer, this line would be printed: `1.33333 0.25E+5 67 ARE THE VALUES OF A,B, AND N`

4c. Carriage Control in `FORMAT` Statements. As another example, suppose the `FORMAT` statement in the program were replaced by:

```
100  FORMAT(30H1 A          B          N//
     1F11.5,E15.5,I6)
```

The first format specification in this `FORMAT` statement indicates that the 30 characters (including blanks) which follow the "`30H1`" are to be printed exactly as they appear (this includes everything up to the two slashes). The "1" functions as a control character. There are three useful control characters, which are used to control the way a line is printed out on a printer, typewriter, or teletypewriter:

control character meaning:

(blank)	Print this line and advance to the next.
1	Skip to a new page, then print this line and advance to the next.
+	Print this line but don't advance to the next.

In the `FORMAT` statement we are presently considering, the computer is instructed to advance the printer to a new page and then print the heading: "`A B N`". The two slashes (`//`) which come after the `30H` format instruct the computer to skip two lines before printing anything further. Thus the actual numerical values for `A`, `B`, and `N` appear on the third line after the heading:

```
      A                B          N

  1.33333          0.25000E+5    67
```

Note that if the numerical value of A had actually needed the alloted eleven positions, the leading digit "`1`" would be the first character on the line and would cause the printer to (unintentionally) skip to a new

page and not print that leading digit. There are other types of formats in addition to the `F`, `E`, `I` and `H`; we shall discuss one of them in the next section.

5. Special Formats

5a. `DATA` Statement: An Alternative I/O. Two methods by which variables in a FORTRAN program can be given numerical values are the `READ` statement and the assignment statement: the `DATA` statement provides a third alternative. An example of a `DATA` statement is // `DATA A,B,C,KOUNT/1.0,2.0,5.7,13/` which is equivalent to the four assignment statements:

```
A = 1.0

B = 2.0

C = 5.7

KOUNT = 13
```

There is, however, one difference: the `DATA` statement(s) must always be placed at the beginning of a program, whereas the assignment statements may appear anywhere.

5b. Numerical and Non-Numerical Data. `DATA` statements can be used to define nonnumerical quantities as well as numerical ones. If we have this `DATA` statement in a program://

```
DATA X,Y,Z,DOG/'B','+',' ','FLEA'/
```

then during execution, the letter "`B`" is stored in the variable `X`, the character "`+`" is stored in the variable `Y`, the character (blank) is stored in the variable `Z`, and the sequence of letters "`FLEA`" is stored in the variable `DOG`.[3] The maximum number of characters that can be stored in one variable usually ranges from four to six depending on the word size of the computer. If letters or other characters are stored in a variable it does not make any sense to use the variable in performing arithmetic operations (e.g., it makes no sense to add the letter "`B`" to the character "+"). A non-numerical variable can be used in assignment statements which do not involve any arithmetic operations, as in this example:

[3] On some computers, it is necessary to use the `H` format specification instead of the single quotation marks, in which case this statement would be written `DATA X,Y,Z,DOG/1HB,1H+,1H,4HFLEA/`

```
      DATA DOG/'FLEA'/
      CAT=DOG
```

This `DATA` statement instructs the computer to store the letters "`FLEA`" in the variable `DOG`. The assignment statement then stores the value of `DOG` (which is "`FLEA`") in the variable `CAT`.

5c. The "`A`" Format is Used for Non-Numerical Data. In order to read or write quantities consisting of strings of characters we may use the `A` format, as shown in this example:

```
      DATA DOG/'FLEA'/
      CAT=DOG
      WRITE(3,110)CAT
110   FORMAT(A4)
      CALL EXIT
      END
```

During execution of this program the `WRITE` statement specifies that the value of the variable `CAT (''FLEA'')` be printed in `A4` format, as indicated in `FORMAT` statement 110. But since the first character on a line is assumed to be a control character, the `F` in "`FLEA`" is interpreted as a control character, and the printed message consists of the remaining characters "`LEA`". It is desirable, therefore, to have the printer skip one or more spaces on the line before the first character appears.

5d. The "`X`" Format Is Used To Print Blanks. Although the `H` format could be used to print blank spaces, we usually use the `X` format. For example, if we want the printer to skip the first 10 spaces on a line and then print the characters "`FLEA`" we need only modify the `FORMAT` statement to read:

```
110 FORMAT(10X,A4)
```

Thus the `X` format is used to skip over some number of spaces, leaving them blank. The `A` format is used to print a string of characters that is stored as a specific variable, the `H` format is used to print a string of characters not associated with a variable, and the `X` format is used to print a string of blank spaces.

6. Type Declaration Statements

6a. `DOUBLE PRECISION` Type Declaration. The method of specifying that calculations involving particular variables be treated using double precision is to include those variables in a `DOUBLE PRECISION` statement at the beginning of the program. For example, the statement

```
DOUBLE PRECISION X,Y,Z
```

would require that the variables `X`, `Y`, `Z`, be treated as double precision variables. If all the variables in an assignment statement are double precision variables then the result will be computed in double precision arithmetic. The `DOUBLE PRECISION` statement is one of a number of "type declaration" statements. Other important examples of such statements include: `REAL`, `INTEGER`, and COMPLEX.

6b. `REAL` and `INTEGER` Type Declarations. The `REAL` and `INTEGER` statements are used to declare that a particular variable should be treated as a real or as an integer variable. These statements are only necessary if you wish to override the FORTRAN convention wherein integer variable names begin with one of the letters `I`, `J`, `K`, `L`, `M`, `N`, and real variables begin with any other letter.

6c. `COMPLEX` Type Declaration. The `COMPLEX` statement is used to specify variables which are to be assigned both real and imaginary parts, permitting FORTRAN to do complex arithmetic.

6d. Location of Type Declarations. Type declaration statements of the kind we have been discussing must appear at the beginning of a program, ahead of any `DIMENSION` statements in which the type-declared variables appear.

7. Some Sample Problems

7a. Maximum of a One-variable Function . Let us assume that we wish to determine the maximum value of some mathematical function F of one variable x in some specific range: $x_1 < x < x_2$. The function $F(x)$ may be too complicated to find its maximum by setting its first derivative equal to zero. There are, however, simple algorithms for finding the maximum numerically. The simplest method, perhaps, is to calculate the function at a series of closely spaced x-values and see where it is largest. If greater accuracy is desired we can "zoom-in" on the region of the maximum by printing out values of the function in a smaller range

near the maximum number found so far. The following simple computer program reads in limits `X1`, `X2` specifying the domain over which the function $F(x)$ is to be calculated and then proceeds to calculate and print out 10 values for $F(x)$ for x-values equally spaced between `X1` and `X2`. For the sake of definiteness we have chosen a particular function.

$$F(x) = e^{-(x-\sqrt{2})^2}$$

Its maximum is at $x = \sqrt{2}$.

```
C  PROGRAM TO PRINT OUT THE VALUE OF A FUNCTION AT 10
C  POINTS BETWEEN X1 AND X2.
C
      READ(5,100)X1,X2
C
C  DX IS THE STEP SIZE
C
      DX=(X2−X1)/9.0
C
C  LOOP OVER 10 X−VALUES
C
      DO 10 J=1,10
      AJ=J−1
      X=X1+AJ∗DX
      F=EXP(−(X−SQRT(2.))∗∗2)
 10   WRITE(6,100)X,F
      CALL EXIT
100   FORMAT(2F10.5)
      END
```

If this program is run repeatedly, using successively more closely spaced limits `(X1, X2)`, we can zoom in on the maximum value of the function $F(x)$. Try it, starting with `X1 = 1`, `X2 = 2`. Start your second run using the x-values for the two largest F-values of the previous run.

7b. Maximum of a Two-variable Function . Now consider a general function of two variables $F(x, y)$ defined over some domain $x_1 < x < x_2$, $y_1 < y < y_2$. We can find its maximum numerically in the same manner by calculating the function on a two-dimensional grid of x, y values and seeing where it is greatest. The following program will calculate and print out the value of the Function $F(x, y)$ at a 10×10 array of points equally spaced inside the domain $x_1 < x < x_2$, $y_1 < y < y_2$. For the sake

of definiteness we have assumed the function.

$$F(x) = e^{-(x-\sqrt{2})^2-(y-\sqrt{3})^2}$$

Its maximum is at $x = \sqrt{2}, y = \sqrt{3}$.

```
C  PROGRAM TO PRINT OUT THE VALUE OF A FUNCTION AT A
C  GRID OF 10x10 POINTS FOR WHICH X IS IN THE RANGE
C  X1 TO X2 AND Y IS IN THE RANGE Y1 TO Y2.
      DIMENSION F(10)
      READ(5,100)X1,X2,Y1,Y2
C
C  COMPUTE STEP SIZE FOR X AND Y
C
      DX=(X2−X1)/9.0
      DY=(Y2−Y1)/9.0
C
C  LOOP OVER Y−VALUES
C
      DO 20 J=1,10
      AJ=J−1
      Y=Y1+AJ∗DY
C
C  LOOP OVER X−VALUES
C
      DO 10 K=1,10
      AK=K−1
      X=X1+AK∗DX
 10   F(K)=EXP(−(X−SQRT(2.)∗∗2−(Y−SQRT(3.)∗∗2)
 20   WRITE(6,101)(F(K),K=1,10)
      CALL EXIT
100   FORMAT(4F10.5)
101   FORMAT(1X,10F10.5)
      END
```

In this program a one-dimensional array $F(K)$ is used for printing since its line of values is printed out for each value of the y-variable. An alternative method would be to store the values for the function in a two dimensional array, $F(J, K)$, which would not have to be printed out until the entire array was calculated. That alternative, however, would unnecessarily increase the amount of memory required (100 locations vs. 10) without having any particular advantage, and should, therefore, be avoided. As in

the case of the one-variable function we can, in effect, "zoom in" on the maximum of the function by running the program repeatedly using `X1`, `X2`,`Y1`, `Y2` values that bracket the maximum of the preceeding run. Try it!

Acknowledgments

Preparation of this module was supported in part by the National Science Foundation, Division of Science Education Development and Research, through Grant #SED 74-20088 to Michigan State University.

A. Structured Fortran Specifications

1. In FORTRAN programs the main program is only a set of calls to subroutines so that the main program can serve as a Table of Contents. Each subroutine's name suggests its function.
2. Variables are generally transferred into and out of a subroutine through its argument list. Labeled `COMMON` is used grudgingly. Unlabeled `COMMON` is never used.
3. In a subroutine's argument list, all "input" variables are listed first. These are followed by three spaces, "in & out" variables, three spaces, "out" variables.
4. Each subroutine begins with `COMMENT` statements listing: (a) the function of the subroutine; (b) the author; (c) the name of the latest modifier and date of modification; and (d) a dictionary of variables and constants.
5. There is a hierarchical diagram showing the flow of variables from one subroutine to another. One subroutine may appear in several different places on the diagram.
6. Generally, each subroutine's list of FORTRAN statements is shorter than one page; preferably much shorter.
7. There are no `GOTO`'s, except for error exits, for operator intervention, or in rare strange circumstances.
8. Each successively nested loop is successively indented. Subroutine calls are preferred over using more than one level of nesting.

MODEL EXAM

1. Write the FORTRAN statements necessary to compute the elements of an array C from the elements of two other arrays A and B that are related to C by:

$$c_1 = a_{11}b_1 + a_{12}b_2 + a_{13}b_3$$
$$c_2 = a_{21}b_1 + a_{22}b_2 + a_{23}b_3$$
$$c_3 = a_{31}b_1 + a_{32}b_2 + a_{33}b_3$$

   Be sure to use: (a) DO loop(s); (b) DIMENSION statement(s); and (c) variable-indices. Note: readers familiar with matrix notation will recognize the multiplication of a three-dimensional vector B by a 3×3 matrix A to produce another 3D vector C.

2. State the default and over-ride rules for these FORTRAN variable types, including an example of each type of over-ride ataement: (a) integer; (b) real; (c) complex; and (d) double precision.

3. Show how each of the data values given below would be printed under control of the stated field specifications:

   a. I3: -16337, -384, -21, 0, 21, 384, 16337

   b. E9.4: -402.1, -9.4, -.02665, 0, .02665, 9.4, 402.1

Brief Answers:

Get some FORTRAN-knowledgable colleague to check your answers.