Numerical Analysis Library
Vol. 1

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Important

The tape cartridge or disc containing the programs is very reliable, but being a mechanical device, is subject to wear over a period of time. To avoid having to purchase a replacement medium, we recommend that you immediately duplicate the contents of the tape onto a permanent backup tape or disc. You should also keep backup copies of your important programs and data on a separate medium to minimize the risk of permanent loss.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Table of Contents

COMMENTARY

Congratulations on your purchase of the most comprehensive Numerical Analysis package ever offered by Hewlett-Packard. Just by browsing through the table of contents for this program package you can see the computational power offered by the System 45.

This set of routines is divided into eight major sections: root finders, integration, ordinary differential equations, linear algebraic systems, eigen analysis, interpolation, functions, and Fourier transforms. They were designed to be used primarily as subroutines in your applications programs. However, drivers are provided so that they may be used as stand-alone programs.

The routines range in sophistication from elementary to state-of-the-art. In all cases, much attention has been paid to making the programs friendly, and easy to use. We are confident that you will find these routines useful and time-saving.

Dave Deane
Application Development

Introduction

I. DESCRIPTION

1. This package is a library of subprograms for performing the most widely used numerical techniques. These subprograms may be linked into memory at the end of the user program and called from that program. Except for error messages, these subprograms contain no input or output operations.

2. To get a desired subprogram:
   a. Insert the appropriate Numerical Analysis cartridge in the tape transport.
   b. Type GET "file name: T15", line no. (where file name is the name of the file in which the subprogram is stored, and line no. is a line number larger than the last line in the calling program.

   NOTE: If your 9845 has Option 600, the secondary tape transport, it may be used. In that case, the mass storage unit specifier must be ":T14".

3. Each subprogram has an associated driver, which elicits from the user the required inputs, links the subprogram and calls it, and prints the resulting outputs. These were written primarily to illustrate some of the techniques which may be used to call the subprograms, but they may be used as stand-alone programs.

4. Drivers and subprograms are contained in files with the same names:

   Driver file names have all capital letters.
   Subprogram file names have the first letter capital, followed by lower case letters.

   e.g., "MULLER" contains the driver for subprogram Muller contained in the file "Muller".

5. In the documentation of these routines, the expression (A MAX B) denotes the maximum value of A and B.

II. SYSTEM CONFIGURATION:

All the programs in this library will fit in any of the memory sizes available on the 9845B and the 9845C.

IV. ORDERING INFORMATION"

The following is a list of part numbers associated with this library:

Complete library (User instructions and two cartridges)	09845-10350
User Instructions	09845-10351
Cartridge 1	09845-10354
Cartridge 2	09845-10355

This section contains four different algorithms for finding roots of equations:

1) Bisect - an iterative rootfinder using the bisection method
2) Roota - a general rootfinder using a modified secant method
3) Muller - a general rootfinder using a quadratic method
4) Siljak - a polynomial rootfinder with complex coefficients.

Unfortunately, solving nonlinear equations is very difficult in practice. The methods given here are local methods. In the presence of good starting guesses, these work quite well. However, in the absence of good estimates, these techniques range from wild divergence to rapid convergence depending upon the particular user-defined function.

Hence, a word of caution. There is no such thing as a fool-proof nonlinear technique. When possible, graphic or analytic means should be used to get a rough estimate of the roots. In terms of speed stability, and precision, each of the above methods has its own advantages and disadvantages. With a number of different techniques to choose from, the user may find one that will most adequately meet his needs.

Subprogram Name: Bisect

This subprogram will search for solutions of $F(x)=\emptyset$ over an interval [a,b] where the user defines the continuous real-valued function F(x). The function may be algebraic of the form $(a_0+a_1x^{e_1}+a_1x^{e_2}+---+a_nx^{e_n})$ with a_i real and e_i rational (e.g. $x^3+3x^{3/2}+x$) or transcendental (e.g., sin(x)+cox(x)).

The user specifies the initial domain value, search increment, error tolerance, and maximum interval-halvings to be used.

Subprogram Utilization:

File Name - "Bisect"
Calling Syntax - CALL Bisect (A,B, Maxbi, Tol, Deltax, Root(*), F(*), Err(*), Nroots)

Input Parameters:

A Lower bound of search interval.
B Upper bound of search interval.
Maxbi Maximum number of bisections for each subinterval.
Tol Error tolerance for a root; a root x is accepted if |F(x)|<Tol*(1 MAX X).
Deltax Search increment; the subprogram begins at the lower bound and compares functional values at the ends of each subinterval, $[a+i\Delta x, a+(i+1)\Delta x]$, for a change of sign.
Nroots Number of roots to be found.

Subprograms required:

FNF(X) Given a domain value X, this function subprogram should return the range value of the user-defined function; this function subprogram must be supplied by the user.

Output parameters:

Root(*) Vector containing the roots of the function; the value 9.999999E99 is supplied for any roots not located.

F(*) Vector containing the functional evaluations of the roots; again, the value 9.999999E99 is supplied for any roots not found.

Err(*) Vector containing the estimated accuracy of the roots; the value 9.999999E99 is supplied for any roots not found.

Special Considerations and Programming Hints:

1. Because the number of roots located cannot be computed ahead of time, all roots are initialized to 9.999999E99 at the beginning of the subprogram. Thus, on output, any roots not found will contain the value 9.999999E99.

2. One of the advantages of this method is that whenever a root is found, both the accuracy of the root (contained in array ERR(*)) and the accuracy of the functional evaluation of the root (contained in array F(*)) are known precisely.

3. Clearly, with enough effort, one can always locate a root to any desired accuracy with this subprogram. But compared with some of the other methods contained in this section, the bi-section method converges rather slowly since it does not make full use of the information about F(X) available at each step. So, when possible, this method should not be used if there are a large number of roots to be found or if the rootfinder sub-program is to be called a number of times.

4. Upon entry into the subprogram, there is a bad data check. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Bisect"
   A = , B = , Maxbi =
   Tol = , Deltax = , Nroots =

   The user may correct the data from the keyboard (e.g., Tol = 1E-6, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

5. If, in a particular subinterval, the maximum number of iterations is exceeded, the subprogram will print the following warning message and the approximate root and accuracies so far obtained:

   "MAX # OF ITERATIONS REACHED ON ROOT #"
   "X BETWEEN" Left endpoint "AND" right endpoint.
   "F(X)=" Y.
   "ACCURACY TO "
   "Average value stored in root(*) as approximate X." The subprogram will then move to the next search interval.

Method and Formulae:

The subprogram Bisect will search for solutions of $F(x) = \emptyset$ over an interval [a,b] where the user defines the real-valued continuous function F(x). The function may be algebraic of the form $a_0+a_1x^{e_1}+a_2x^{e_2}+---+a_nx^{e_n}$ with a_i real and e_i rational (e.g. $x^3+3x^{3/2}+x$) or transcendental (e.g., sin (x)+cos(x)).

The user specifies the search increment Δx and the error tolerance for F(x). The program then begins at the left of the interval and compares functional values at the ends of the subinterval $[a, a+\Delta x]$. If the functional values are of opposite sign then the bisection method is used to locate the root. Each subinterval $[a+i\Delta x, a+(i+1)\Delta x]$ is examined for a possible root. At most, one root per interval will be located and if there are multiple roots per interval, none will be located. The user must also specify a maximum number of interval - halvings (Maxbi), so that an error tolerance that is not satisfied will result in the root localized to an interval of size $2^{-Maxbi}(b-a)$. The subprogram will examine $N = \text{int } \frac{b-a}{\Delta x}$ intervals.

References:

1. Stark, Peter A., Introduction to Numerical Methods (London: MacMillan Company, Collier-MacMillan Limited, 197Ø), pp. 95-96.

Driver Utilization:

File Name - "BISECT" , cartridge 1

The driver "BISECT" sets up the necessary input parameters for the subprogram Bisect and prints out the resulting outputs.

With each question, the user is only required to input the appropriate response. The driver also sets up the user-defined function sub-program.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "BISECT".
   a. Type: GET "BISECT"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Bisect ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, Bisect contained in file "Bisect" will be linked on after the driver.

5. When "HAS FUNCTION ALREADY BEEN DEFINED (Y/N)?" is displayed:
   a. Enter Y if the function has already been defined on the previous run.

b. Press: CONT.

c. Go to 7.

or

a. Enter N if the function has not yet been defined.

b. Press: CONT.

c. Go to 6.

6. When "FUNCTION [e.g., F=(X-3)*(X+4)]?" is displayed:

a. Type in the user-defined function in the above format.

b. Press: CONT.

c. At this point, the user-defined function will be properly entered in function subprograms FNF(X). This subprogram will then be stored on the mass storage device in file "KRYWEN" and then linked on to the program after subprogram Bisect. (File "KRYWEN" must have been created previously to running this program. Statement: CREATE "KRYWEN", 4,84 does this. File "KRYWEN" has already been created on the NUMERICAL ANALYSIS cartridge.)

d. Go to 7.

7. When "# OF ROOTS TO BE FOUND?" is displayed:

a. Enter the number of roots to be found.

b. Press: CONT.

8. When "LOWER BOUND?" is displayed:

a. Enter the lower bound of the search interval.

b. Press: CONT.

9. When "UPPER BOUND?" is displayed:

a. Enter the upper bound of the search interval.

b. Press: CONT.

10. When "MAXIMUM # OF BISECTIONS?" is displayed:
    a. Enter the maximum number of bisections allowed in searching for any one root in a subinterval.
    b. Press: CONT.

11. When "ERROR TOLERANCE?" is displayed:
    a. Enter the error tolerance desired. Here $|F(x)| \leq Tol*(1\ MAX\ X)$. is the tolerance criterion.
    b. Press: CONT.

12. When "SEARCH INCREMENT?" is displayed:
    a. Enter the search increment, i.e., the subprogram begins at the lower bound and compares functional values at the ends of each subinterval, $[a+i\Delta x, a+(i+1)\Delta x]$, for a change of sign.

13. The roots x, their functional values F(X), and the accuracy to which the root is found will be printed for all roots found over the interval. The value 9.999999E99 is inserted for any roots not found.

EXAMPLE

```
F=SIN(X)-COS(X)/(1+X*X)


# OF ROOTS TO BE FOUND= 4
LOWER BOUNDS= 0
UPPER BOUNDS= 10
MAXIMUM # OF BISECTIONS= 20
ERROR TOLERANCE= .00000001
SEARCH INCREMENT= .1

     ROOT                FUNCTION VALUE              ACCURACY

 6.238996E-01            -9.425000E-09            1.907350E-07
 3.228892E+00             1.621320E-08            1.907400E-07
 6.307698E+00            -1.625780E-08            1.907000E-07
 9.435884E+00             8.955260E-08            3.051750E-06
```

Subprogram Name: Roota

Given a first guess, this subprogram will search for a solution of $F(x)=\emptyset$ where the user defines the continuous real-valued function $F(x)$.

Roots are found by marching along at a given step size until a change of sign is encountered. Then a modified secant method is employed to determine the zero of the function.

The user is required to specify the error tolerances for the root and for the function evaluation, as well as the step size and the maximum number of steps and iterations allowed.

Subprogram Utilization:

File Name - "Roota"

Calling Syntax - CALL Roota (Root, Err, Frsges, Step, Istpmx, Itrms, Tola, Tolf)

Input Parameters:

Frsges	Initial guess for the root.
Step	Step size for marching when looking for an interval in which the function changes sign; step size must be positive.
Istpmx	Maximum number of allowed steps in marching to find an interval in which the function changes sign.
Itrmx	Maximum number of iterations allowed in subprogram work after an interval has been found in which the function changes sign.

Tola	Tolerance for the root; i.e., if X_1 and X_2 are two consecutive approximations for the root, the average of X_1 and X_2 is accepted as a root if $\|X_1-X_2\| \leq (1 \text{ MAX } (\|X_1\| \text{ MAX } \|X_2\|))*\text{Tola}$.
Tolf	Tolerance for the function; i.e., an approximation X is accepted as a root if $\|F(X)\| \leq$Tolf.

Subprograms Required:

Chkit	Checks if a root is found or bracketed.
FNF(X)	User-defined function which, when given a domain value X, returns the value F(X).
FN Incres	Checks if too many marching steps have been taken in subprogram Roota.
Monton	Checks that the function is well-behaved in the search interval.
Work	Finds the root of the user-defined real-valued continuous function F by a modified secant method; requires that the functional values of the end-points of the interval be of opposite sign.

Output Parameters:

Root	Root of F(X)
Err	F(Root)

Special Considerations and Programming Hints:

1. Upon entry into subprogram Roota, there is a bad data check. If the check detects any "nonsense" data, an error message will be printed and the program pauses:

   "ERROR IN SUBPROGRAM Roota"

   Step = Istpms = Itrmx =

   Tola = Tolf =

   The data may be corrected from the keyboard (e.g., Tola = 1E-6, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. This method is slower than subprogram Muller, but has the advantage of having less trouble with functions that have somewhat unusual behavior, or functions that have periodic oscillations.

3. Subprogram Monton is used to check that the function is monotonic in the interval under consideration. If a change in direction is encountered, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Monton."

   "Y=F(X) IS NOT MONOTONIC OR IS VERY FLAT."

   "SUGGEST TRYING SMALLER STEP SIZE OR DIFFERENT FIRST GUESS."

   X1 = X2 = X3 =

   Y1 = Y2 = Y3 =

   There are many possible causes of this error, the most common being that (X2,Y2) is a relative minimum or maximum. Choosing X1 or X3 as the first guess and a step size less than |X1-X2| may solve this problem.

If the data indicates that the function is very flat in the interval, an initial guess on either side of the interval may help.

Methods and Formulae:

Subprogram Roota begins by initializing X2 = Frsges. Then Subprogram Chkit is called. In Chkit a new X is generated and ABS (F(X)) < = TOLF is tested. If it is true, then X is accepted as the root. Otherwise a step to the left and a step to the right are tested. If a root is not found and if F does not charge sign, then a test is made to see which way to march.

(Y2-Y1)*Y1 < Ø then march right
(Y2-Y1)*Y1 > Ø then march left.

If a root or a change of sign is not found after Istpmx steps, an error message is printed and the program pauses.

If a change of sign is found, then Chkit calls subprogram Work, where Root is found by a modified secant method which requires 2 ordinates to be of opposite sign.

$$x_2=(y_3*x_1-y_1*x_3)/(y_3-y_1)$$

A root is accepted if

ABS(F(X2)) < = TOLF

or

ABS(XA-XB) < = [1 MAX(ABS(XB)MAX ABS(XA))]*Tola
where XA, XB are consecutive attempts at finding a root.

Subprogram Monton is invoked repeatedly to test that F is strictly increasing or decreasing and that x_1, x_2, x_3 are in ascending order.

Driver Utilization:

File Name - "ROOTA" , cartridge 1

The driver "ROOTA" sets up the necessary input parameters for the subprogram Roota and prints out the resulting outputs.

With each question, the user is only required to input the appropriate response. The driver also sets up the user-defined function subprogram.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "ROOTA"
   a. Type: GET "ROOTA"
   b. Press: EXECUTE.

3. Press: RUN

4. When "HAS Roota ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Roota contained in file "Roota" will be linked on after the driver.
   d. Go to 5.

5. When "HAS FUNCTION ALREADY BEEN DEFINED (Y/N)?" is displayed:
   a. Enter Y if the function has already been defined on a previous run.
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter N if the function has not yet been defined.
   b. Press: CONT.
   c. Go to 6.

6. When "FUNCTION [e.g., F=(X-3)*(X+4)]?" is displayed:
   a. Type in the user-defined function in the above format.
   b. Press: CONT.
   c. At this point, the user-defined function will be properly entered in function subprogram FNF(X). This subprogram will then be stored on the mass storage device in file "KRYWEN" and then linked onto the program after subprogram Roota. (File "KRYWEN" must have been created previously to running this program. Statement: CREATE "KRYWEN", 4,84 does this. File "KRYWEN" has already been created on the NUMERICAL ANALYSIS cartridge.)
   d. Go to 7.

7. When "FIRST GUESS?" is displayed:
   a. Enter a first guess.
   b. Press: CONT.

8. When "STEP SIZE?" is displayed:
   a. Enter the step size for marching when looking for an interval in which the function changes sign; this must be positive.
   b. Press: CONT.

9. When "MAXIMUM NUMBER OF STEPS?" is displayed:
   a. Enter the maximum number of allowed steps in marching to find an interval in which the function changes sign.
   b. Press: CONT.

10. When "MAXIMUM NUMBER OF ITERATIONS?" is displayed:
   a. Enter the maximum number of iterations allowed in subprogram Work after an interval has been found in which the function changes sign.
   b. Press: CONT.

11. When "TOLERANCE FOR ROOT?" is displayed:
   a. Enter the tolerance for the root; i.e., if X_1 and X_2 are two consecutive approximations for the root, the average of X_1 and X_2 is accepted as a root if
   $$|X_1-X_2| \leq (1 \text{ MAX } (|X_1| \text{ MAX } |X_2|))*(\text{Tolerance for root})$$
   b. Press: CONT.

12. When "TOLERANCE FOR FUNCTION EVALUATION?" is displayed:
   a. Enter the tolerance for the function, i.e., an approximation X is accepted as a root if $|F(X)| \leq$ (Tolerance for the function).
   b. Press: CONT.

13. The root X as well as the functional value F(X) will be printed.

EXAMPLE

```
F=X*LOG(X)-1


FIRST GUESS= 1
STEP SIZE= .25
MAXIMUM NUMBER OF STEPS= 30
MAXIMUM NUMBER OF ITERATIONS= 30
TOLERANCE FOR ROOT= .00000001
TOLERANCE FOR FUNCTION= .000001


ROOT= 1.763223E+00     FUNCTION VALUE=-5.031000E-08
```

Subprogram Name: Work

This subprogram is used by subprogram Roota, a root finder subprogram. Work finds the root of a user-defined real-valued continuous function by a modified secant method. It requires that the functional values of the endpoints of the search interval be of opposite sign.

Subprogram Utilization:

File Name - contained in file "Roota"

Calling Syntax - CALL Work (Root, Err, T1, Z1, T3, Z3, Itrmx, Tola, Tolf, Kounti)

Input Parameters:

T1, T3	Endpoints of search interval.
Z1, Z3	F(T1), F(T3)
Itrmx	Maximum number of iterations allowed in searching for a root
Tola	Tolerance for the root; i.e., if X_1 and X_2 are two consecutive approximations for the root, the average of X_1 and X_2 is accepted as a root if $\|X_1-X_2\| \leq (1 \text{ MAX } (\|X_1\| \text{ MAX } \|X_2\|))*\text{Tola}$.
Tolf	Tolerance for the function, i.e., an approximation X is accepted as a root if $\|F(X)\| \leq$ Tolf.
Kounti	Counter on number of iterations.

Subprograms Required:

FNF(X)	User-defined function which, when given a domain value X, returns the value F(X).

Output Parameters:

Root	Root of F(X)
Err	F(Root)

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, Z1 and Z3 must be of opposite sign. If not, a BAD DATA check is printed.

   "ERROR IN SUBPROGRAM WORK."
   "Z1 and Z2 ARE NOT OF OPPOSITE SIGN."
   Z1 = Z3 =

2. If subprogram Roota is being used, Work is called only in subprogram Chkit. Chkit specifically checks that Z1 and Z3 are opposite sign. So this error will not occur.

   If subprogram Work is being used independently, then the user should check both the logic in the calling program and the specific behavior of the function in the interval [T1,T3].

3. If Itrmx, the maximum number of iterations is exceeded, an error message is also printed.

   "ERROR IN SUBPROGRAM Work. MAXIMUM # OF ITERATIONS EXCEEDED."
   Kounti = Itrmx =

   In this case, there are two possible solutions. First, increase Itrmx, the maximum number of iterations:

   a. Type: Itrmx = new maximum
   b. Press: EXECUTE
   c. Press: CONT.

The program will resume, incorporating the new value of Itrmx. There is a reasonable limit here, though. Itrmx greater than 50 would probably not make sense. If convergence is not obtained after 50 iterations, there is some other problem. The user should check to see if F(X) is exhibiting some unusual behavior in the vicinity of the looked-for root. It might even be necessary to switch to a simple bisection method, subprogram Bisect, in this subinterval to find the root.

4. In general, the modified secant method will be faster and obtain greater accuracy than the bisection method. If a large number of roots needs to be located, subprogram Roota or Muller should be used.

Methods and Formulae:

See subprogram Roota.

Subprogram Name: Chkit

This subprogram is used in subprogram Roota. Chkit checks if a root has been found or bracketed.

Subprogram Utilization:

File Name - contained in file "Roota", cartridge 1

Calling Syntax - CALL Chkit (Root, Err, T1, Z1, Step, T2, Z2, Itrmx, Tola, Tolf, Rtn)

Input Parameters:

T1,T2	Endpoints of search interval.
Z1, Z2	F(T1), F(T2).
Step	Step size for marching when looking for an interval in which the function changes sign; step size must be positive.
Itrmx	Maximum number of iterations allowed in subprogram Work after an interval has been found in which the function changes sign.
Tola	Tolerance for the root, i.e., if X_1 and X_2 are two consecutive approximations for the root, the average of X_1 and X_2 is accepted as a root if $\|X_1-X_2\| \leq (1 \text{ MAX } (\|X_1\| \text{ MAX } \|X_2\|))*\text{Tola}$.
Tolf	Tolerance for the function; i.e., an approximation X is accepted as a root if $\|F(X)\| \leq$ Tolf.
Rtn	Rtn = 1 implies root located. Rtn = -1 implies root not yet located.

Subprograms Required:

FNF(X)	User-defined function which, when given a domain value X, returns the value F(X).
Work	Finds the root of the user-defined real-valued continuous function F by a modified secant method.

Output Parameters:

Root	Root of F(X)
Err	F(Root)

Special Considerations and Programming Hints:

See subprogram Roota

Methods and Formulae:

Upon entry into Chkit, the search interval is incremented one step size. A check is performed to see if the function value of the new endpoints satisfies the error tolerance, TOLF. If so, the root has been located and the subprogram is exited. If not, a check is made to see if the root has been bracketed. If so, subprogram Work is called. If not, the subprogram is exited.

Subprogram Name: Monton

This suprogram is used in subprogram Roota, a root finder. Monton checks the three points (X_1,Y_1), (X_2,Y_2), (X_3,Y_3) to see that Y=F(X) is strictly increasing or decreasing. If not, then an error message is printed.

This subprogram insures that the function is well-behaved in the search interval.

Subprogram Utilization:

File Name - contained in file "Roota", cartridge 1

Calling Syntax - CALL Monton (Tolf, X1, X2, X3, Y1, Y2, Y3)

Input Parameters:

Tolf	Provides tolerance for flatness also used as tolerance for function evaluation in other subprograms.
X1,X2,X3	Three current values of search interval.
Y1,Y2,Y3	F(X1), F(X2), F(X3)

The input parameters are all left unchanged and there are no output parameters for this subprogram.

Special Considerations and Programming Hints:

1. Upon entering this subprogram, a check is made to see that X1 < X2 < X3. If not, an error message is printed:

   "ERROR IN SUBPROGRAM Monton."
   "ORDINATES X1,X2,X3 ARE NOT IN INCREASING ORDER."
   X1 = X2 = X3 =

If this error occurs, it suggests that the user-defined function is extremely ill-behaved.

2. Monton's primary purpose is to check that the function is monotonic in the interval under consideration, i.e. $Y_1<Y_2<Y_3$ or $Y_1>Y_2>Y_3$.

There is also cause for alarm if the function is almost flat in the interval, i.e., Y_1, Y_2 and Y_3 are extremely close in value. If this is the case, then an error message is printed.

"ERROR IN SUBPROGRAM Monton."
"Y=F(X) IS NOT MONOTONIC OR IS VERY FLAT."
"SUGGEST TRYING SMALLER STEP SIZE OR DIFFERENT FIRST GUESS"
X1 = X2 = X3 =
Y1 = Y2 = Y3 =

There are many possible causes of this error, the most common being that (X2,Y2) is a relative minimum or maximum. Choosing X1 or X3 as the first guess may solve this problem. Another possibility is to choose a smaller step size.

Methods and Formulae:

Monton performs two checks:

i) that $x_1<x_2<x_3$
and
ii) that (y3-y2)*(y2-y1)>=Tolf

This second inequality tests that F(x) is strictly increasing or decreasing in the interval $[x_1,x_3]$ and that the values y_1, y_2, y_3 are not too close in value. The tolerance for the function, Tolf, is used for that purpose here.

Subprogram Name: FN Incres

This subprogram is used in subprogram Roota, a root finder, and is used to check if too many marching steps have been taken in searching for a change of sign.

Subprogram Utilization:

File Name - contained in file "Roota", cartridge 1

Calling Syntax - FN Incres (I, Istpmx, X1, X2, X3, Y1, Y2, Y3)

Input Parameters:

I	Number of steps taken upon entering subprogram.
Istpmx	Maximum number of steps allowed in marching to find an interval in which the function changes sign.
X1,X2,X3	Three points in the interval [X1,X3]
Y1,Y2,Y3	Y1=F(X1), Y2=F(X2), Y3=F(X3)

Output:

FN Incres I+1

Special Considerations and Programming Hints:

1. If the maximum number of steps, Istpmx, has been exceeded, an error message is printed and the program halts.

   "ERROR IN INCRES IN Roota"
   "NO CHANGE OF SIGN AFTER STEPS"
   X1 = Y1 =
   X2 = Y2 =
   X3 = Y3 =

   If the user wishes to continue with a larger maximum number of steps, Istpmx, he may do the following:

   i) Type: Istpmx = new maximum
   ii) Press: EXECUTE
   iii) Press: CONT.

The program will resume with the new maximum number of steps. If Istpmx = 2Ø does not work, the user may want to attempt Istpmx = 5Ø. If this is not effective, it would not make sense to try Istpmx = 100 or Istpmx = 1000. There is some other difficulty.

Subprogram Name: Muller

Given a vector of initial guesses, this subprogram will search for solutions of F(X)=Ø where the user defines the continuous real-valued function F(X). Roots are found by Muller's method, a real, quadratic root solver.

The user is required to specify the number of roots to be found, the error tolerance for the function evaluations and the number of significant digits desired in the roots, as well as the spread criteria for multiple roots and the maximum number of iterations per root.

Subprogram Utilization:

File Name - "Muller", cartridge 1

Calling Syntax - CALL Muller (Root(*), NRoots, Itmax, Tolf, Eps, Eta, Digits)

Input Parameters:

*Root(*)	Vector containing initial guesses of roots.
Nroots	Number of roots to be found.
*Itmax	Maximum number of iterations per root.
Tolf	Function tolerance; i.e., X is accepted as a root if $\|F(X)\| \leq$ Tolf.
Eps	Spread tolerance for multiple roots.
Eta	Restart value for multiple roots.

Note: If the Ith root (Root(I)) has been computed and it is found that $|Root(I)-Root(J)| \leq Eps$ where Root(J) is a previously computed root, then the computation is restarted with a guess equal to Root (I) + Eta.

Digits	Number of significant digits desired in root.

Subprograms Required:

FNF(X)	User-defined function which, when given a domain value X_1 returns the value F(X).

Output Parameters:

*Root(*)	Vector containing roots of the function.
*Itmax	Number of iterations required to find final root.

* Itmax and Root(*) are both input and output parameters.

Special Considerations and Programming Hints:

1. Upon entry into subprogram Muller, there is a bad data check. If the check detects any "nonsense" data, an error message will be printed:

   "ERROR IN SUBPROGRAM Muller."
   Nroots = Itmax = Tolf =

   The data may be corrected from the keyboard (e.g., Tolf = 1E-8, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. If the maximum number of iterations, Itmax, is exceeded in searching for any one root, I, an error message will be printed.

   "ERROR IN SUBPROGRAM Muller."
   "MAXIMUM # OF ITERATIONS EXCEEDED ON Root" I and Root (I) = 9.999999E99.

   The subprogram continues execution with a search for the next root.

3. Muller's technique has the quickest rate of convergence of all the methods contained in this section. If the user has good initial estimates of the roots, this method is the one to use.

   But, there are times this method gives misleading results. For example, if the first fifty positive roots of a particular function are desired and the user does not have good initial estimates, Muller may not generate the desired results. Some roots may be skipped over; or some negative roots may be generated.

For example: assume we want to locate the first 6 positive roots of F = Sin(X) and we have poor initial estimates:

```
F=SIN(X)


# OF ROOTS TO BE FOUND= 6
MAXIMUM # OF ITERATIONS= 20
TOLERANCE FOR FUNCTION= .000001
SPREAD TOLERANCE FOR MULTIPLE ROOTS= .00000001
RESTART VALUE FOR MULTIPLE ROOTS= .0001
# OF SIGNIFICANT DIGITS IN ROOT= 6


INITIAL GUESS( 1)= 1.000000E+00
INITIAL GUESS( 2)= 2.000000E+00
INITIAL GUESS( 3)= 3.000000E+00
INITIAL GUESS( 4)= 4.000000E+00
INITIAL GUESS( 5)= 5.000000E+00
INITIAL GUESS( 6)= 6.000000E+00




ROOTS:

   Root( 1)= 2.620439E-08
   Root( 2)= 3.141593E+00
   Root( 3)= 6.283185E+00
   Root( 4)= 9.424778E+00
   Root( 5)= 1.256637E+01
   Root( 6)=-3.141593E+00
```

On the other hand, assume that our initial guesses are good:

```
F=SIN(X)


# OF ROOTS TO BE FOUND= 6
MAXIMUM # OF ITERATIONS= 20
TOLERANCE FOR FUNCTION= .000001
SPREAD TOLERANCE FOR MULTIPLE ROOTS= .00000001
RESTART VALUE FOR MULTIPLE ROOTS= .0001
# OF SIGNIFICANT DIGITS IN ROOT= 6


INITIAL GUESS( 1)= 3.000000E+00
INITIAL GUESS( 2)= 6.000000E+00
INITIAL GUESS( 3)= 9.000000E+00
INITIAL GUESS( 4)= 1.200000E+01
INITIAL GUESS( 5)= 1.500000E+01
INITIAL GUESS( 6)= 1.800000E+01




ROOTS:

   Root( 1)= 3.141593E+00
   Root( 2)= 6.283185E+00
   Root( 3)= 9.424778E+00
   Root( 4)= 1.256637E+01
   Root( 5)= 1.570796E+01
   Root( 6)= 1.884956E+01
```

Methods and Formulae:

Given a vector of initial guesses, subprogram Muller will search for solutions of F(X)=Ø where the user defines the continuous real-valued function F(X).

The algorithm is based on an extension of Muller's method by WERNER L. FRANK. This procedure does not depend on any prior knowledge of the location of the roots nor on any special starting process. All that is required is the ability to evaluate F(X) for any desired value of X. Multiple roots can also be obtained.

Given an initial guess, Root(I), supplied by the user,

$P = .9*Root(I)$

$P_1 = 1.1*Root(I)$

$P_2 = Root(I)$

if Root(I)≠Ø and P = -1, P1 = 1, P2 = Ø if Root (I) = Ø. From these three starting values, one chooses Rt, the next approximation to the root, as one of the zeros of the second degree polynomial which passes through the functional values F(P), $F(P_1)$ and $F(P_2)$. Successive approximations are obtained by repeating the quadratic fit:

$$Rt_{i+3} = Rt_{i+2} + (Rt_{i+2} - Rt_{i+1})*d_{i+3}$$

where $$d_{i+3} = \frac{-2*F(Rt_{i+2})*(1+d_{i+2})}{b_{i+2} \pm \left[b_{i+2}^2 - 4*F(Rt_{i+2})*d_{i+2}*(1+d_{i+2})*\{F(Rt_i)*d_{i+2} - F(Rt_{i+1})*(1+d_{i+2}) + F(Rt_{i+2})\}\right]^{1/2}}$$

and $$b_{i+2} = F(Rt_i)*d_{i+2}^2 - F(Rt_{i+1})*(1+d_{i+2})^2 + F(Rt_{i+2})*(1+2*d_{i+2})$$

The sign in the denominator is chosen to give d_{i+3} the smaller magnitude.

Having found (R-1) roots, one determines the Rth root by solving the equation $F_R(X)=\emptyset$ where:

$$F_R(X) = \frac{F(X)}{\prod_{i=1}^{R-1} (X-X_i)} \qquad (R=2,3,\cdots).$$

The process halts when successive iterants pass one of the two convergence tests:

$|H| < |Rt|*10^{-Digits}$ where H=Rt-previous Rt
Digits = number of significant digits desired in root.

or

$|F(Rt)| <$ Tolf and $|F(Rt)| <$ Tolf where Tolf is the desired function tolerance.

If the maximum number of iterations is exceeded on any root, an error message will be printed and the root will default to the value 9.999999E99.

References:

1. D.E. Muller, "A method for Solving Algebraic Equations Using an Automatic Computer", MTAC 10 (1956).
2. W.L. Frank, "Finding Zeros of Arbitrary Functions", J.ACM 5(1958).
3. B. Leavenworth, "Algorithm 25: Real Zeros of an Arbitrary Function", Comm. of ACM 3(11) 1960, 602.

Driver Utilization:

File Name - "Muller", cartridge 1

The driver "MULLER" sets up the necessary input parameters for the subprogram Muller as well as prints the resulting outputs.

With each question, the user is only required to answer with the appropiate response. The driver also sets up the user-defined functions subprogram.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "MULLER".
   a. Type: GET "MULLER"
   b. Press: EXECUTE

3. Press: RUN

4. When "HAS Muller ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Muller contained in file "Muller" will be linked on after the driver.

5. When "HAS FUNCTION ALREADY BEEN DEFINED (Y/N)?" is displayed:
   a. Enter Y if the function has already been defined on the previous run.
   b. Press: CONT.
   c. Go to 7

or

   a. Enter N if the function has not yet been defined.
   b. Press: CONT.
   c. Go to 6.

6. When "FUNCTION [e.g.,F=(X-3)*(X+4)]?" is displayed:
   a. Type in the user-defined function in the above format.
   b. Press: CONT.
   c. At this point, the user-defined function will be properly entered in function subprogram FNF(X). This subprogram will then be stored on the mass storage device in file "KRYWEN" and then linked on to the program after subprogram Muller. (File "KRYWEN" must have been created previously to running this program. Statement: CREATE "KRYWEN", 4,84 does this. File "KRYWEN" has already been created on the NUMERICAL ANALYSIS cartridge.)
   d. Go to 7.

7. When "# OF ROOTS TO BE FOUND?" is displayed:
   a. Enter the number of roots to be found.
   b. Press: CONT.

8. When "MAX # OF ITERATIONS?" is displayed:
   a. Enter the maximum number of iterations permitted per root.
   b. Press: CONT.

9. When "TOLERANCE FOR FUNCTIONAL EVALUATION?" is displayed:
   a. Enter the tolerance for the functional evaluation, i.e., X is accepted as a root if $|F(X)| \leq$ this tolerance.
   b. Press: CONT.

10. When "SPREAD TOLERANCE FOR MULTIPLE ROOTS?" is displayed:
   a. Enter the spread tolerance for multiple roots. If the Ith root, Root(I), has been computed and it is found that $|Root(I)-Root(J)| \leq$ this tolerance where Root(J) is a previously computed root, then the computation is restarted with a guess equal to Root(I)+restart value.
   b. Press: CONT.

11. When "RESTART VALUE FOR MULTIPLE ROOTS?" is displayed:
    a. Enter the restart value as explained in #10.
    b. Press: CONT.

12. When "# OF SIGNIFICANT DIGITS?" is displayed:
    a. Enter the number of significant digits desired for a root to be accepted, i.e., a root is accepted if two successive approximations to a given root agree in the first "DIGITS" digits.
    b. Press: CONT.

13. When "INITIAL GUESS (I)?" (I=1 to # OF ROOTS) is displayed:
    a. Enter the initial guesses on the roots to be found.
    b. Press: CONT.

14. Values will be printed for all roots found. The value 9.999999E99 is inserted for any roots not found.

EXAMPLE

```
F=SIN(X)


# OF ROOTS TO BE FOUND= 4
MAXIMUM # OF ITERATIONS= 20
TOLERANCE FOR FUNCTION= .00000001
SPREAD TOLERANCE FOR MULTIPLE ROOTS= .0000000001
RESTART VALUE FOR MULTIPLE ROOTS= .0001
# OF SIGNIFICANT DIGITS IN ROOT= 8


INITIAL GUESS( 1)= 0.000000E+00
INITIAL GUESS( 2)= 2.000000E+00
INITIAL GUESS( 3)= 4.000000E+00
INITIAL GUESS( 4)= 6.000000E+00




ROOTS:

   Root( 1)= 0.000000E+00
   Root( 2)= 3.141593E+00
   Root( 3)= 6.283185E+00
   Root( 4)= 9.424778E+00
```

Subprogram Name: Siljak

This subprogram will find all roots, Z, of polynomials of the form

$$a_0+ib_0+(a_1+ib_1)Z+(a_2+ib_2)Z^2+---+(a_n+ib_n)Z^n=\emptyset.$$

Roots are found by expressing the polynomials in terms of Siljak functions and using the method of steepest descent to determine the zeros.

The user is required to enter the complex coefficients of the polynomial as well as its degree. Tolerances for the root and for function evaluations are the maximum number of iterations are also needed.

Subprogram Utilization:

File Name - "Siljak", cartridge 1

Calling Syntax - CALL Siljak (N, Rcoef(*), Icoef(*), Tola, Tolf, Itmax, Rroot(*), Iroot(*)).

Input Parameters:

N	Degree of polynomial; number of roots to be found.
Rcoef(*)	Vector containing the real parts of the coefficients of the polynomial where the subscript corresponds to the exponent of the variable; subscripted from 0 to N. e.g., for $a_0+ib_0+(a_1+ib_1)Z+(a_2+ib_2)Z^2+---+(a_n+ib_n)Z^n=\emptyset$, Rcoef(3) would be the real part of the

	coefficient of the Z^3 term, a_3.
Icoef(*)	Vector containing the imaginary parts of the coefficients of the polynomial where the subscript corresponds to the exponent of the variable; subscripted from Ø to N.
Tola	Tolerance for the root.
Tolf	Tolerance for the function evaluation.
Itmax	Maximum number of iterations permitted in searching for any one root.

Output Parameters:

Rroot(*)	Vector containing the real parts of the roots of the polynomial; subscripted from 1 to N.
Iroot(*)	Vector containing the imaginary parts of the roots of the polynomial; subscripted from 1 to N.

Special Considerations and Programming Hints:

1. Upon entry into subprogram Siljak, there is a bad data check. If the check detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Siljak."
   N = , Tola = , Tolf = , Iter =

   The data may be corrected from the keyboard (e.g., Tola = 1E-6, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. The subprogram has been set to quarter the interval size at most 2Ø times. If this maximum is exceeded, an error message is printed:

   "ERROR IN SUBPROGRAM Siljak."
   "THE INTERVAL SIZE HAS BEEN QUARTERED 2Ø TIMES AND THE TOLERANCE FOR FUNCTIONAL EVALUATIONS IS STILL NOT MET."

Tolf = , U = , V =

The subprogram is then exited with all known information. All roots not found will contain the value 9.999999E99.

If more than 20 quarterings are required, there is probably some unusual behavior in the function.

3. If the maximum number of iterations has been exceeded, the following error message will be printed and Siljak will be exited with all known information:

"ERROR IN SUBPROGRAM Siljak."
"MAXIMUM # OF ITERATIONS HAS BEEN EXCEEDED."
L = Itmax =

Again, all roots not located will contain the value 9.999999E99.

There are two possible solutions here. First, increase Itmax, the maximum number of iterations allowed; or second, allow larger error tolerances, Tola and Tolf. The iteration counter, L, is reset after each root is located, so attempting to solve a polynomial of large degree should not cause this error.

Methods and Formulae:

This subprogram will find all the roots Z of polynomials of the form $a_0+ib_0+(a_1+ib_1)Z+(a_2+ib_2)Z^2+---+(a_n+ib_n)Z^n=\emptyset$.

Roots are found by expressing the polynomial in terms of Siljak functions and using the method of steepest descent to determine the zeros. Once a root is found, the polynomial is reduced by synthetic division and the process is repeated. The last root is computed algebraically. The algorithm is very accurate and stable; it will virtually always find the roots and the user is not required to provide an initial value. Multiple roots are found at some slightly reduced accuracy, and higher order polynomials may show some loss of accuracy as more roots are found. In general, the program will find "normally" spaced roots accurate to better than 6 decimal places. Newton's method could find the roots faster, but convergence is not guaranteed, and with Siljak's method, no a priori information such as the derivative is necessary.

$$F(Z)=\sum_{k=0}^{n}(a_k+ib_k)Z^k=\emptyset$$

Siljak functions x_k and y_k are defined by

$Z^k=x_k+iy_k$ and may be calculated recursively

$x_0=1, x_1=.1, y_0=\emptyset, y_1=1$

$x_{k+2}=2x\ x_{k+1}-(x^2+y^2)x_k$

$y_{k+2}=2x\ y_{k+1}-(x^2+y^2)y_k$

where x + iy are the root approximations

$$\mu=\sum_{k=0}^{n}(a_kx_k-b_ky_k)$$

$$\frac{\partial\mu}{\partial x}=\sum_{k=1}^{n}k\ (a_kx_{k-1}-b_ky_{k-1})$$

$$\frac{\partial v}{\partial x}=\sum_{k=1}^{n}k\ (a_ky_{k-1}+b_kx_{k-1})$$

References:

1. Moore, J.B., "A Convergent Algorithm for Solving Polynomial Equations", Journal of the Association for Computing Machinery. vol. 14, No. 2 (April, 1967), pp. 311-315.

Driver Utilization:

File Name - "SILJAK", cartridge 1

The driver "SILJAK" sets up the necessary input parameters for the subprogram Siljak as well as prints out the resulting outputs

With each question, the user is only required to input the appropriate response.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge with the machine turned on.

2. Get file "SILJAK."
   a. Type: GET "SILJAK"
   b. Press: EXECUTE.

3. Press: RUN

4. When "HAS Siljak ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Siljak contained in file "Siljak" will be linked on after the driver.

5. When "DEGREE OF POLYNOMIAL?" is displayed:
   a. Enter the degree of the polynomial; this is also the number of roots to be found.
   b. Press: CONT.

6. When "MAX # OF ITERATIONS?" is displayed:
   a. Enter the maximum number of iterations allowed per root.
   b. Press: CONT.

7. When "TOLERANCE FOR ROOTS?" is displayed:
   a. Enter the tolerance desired for the roots; a root is accepted if the difference in value of the root approximations of two successive iterations is less than this tolerance.
   b. Press: CONT.

8. When "TOLERANCE FOR FUNCTIONAL EVALUATIONS?" is displayed:
   a. Enter the tolerance for the functional evaluations; a root X is accepted if $|F(X)| \leq$ this tolerance.
   b. Press: CONT.

9. When "Rcoef(I)=?" (For I=1 to degree of polynomial) is displayed:
   a. Enter the appropriate real part of the coefficient; each subscript corresponds to the exponent of the variable; i.e., $(a_0+b_0i)+(a_1+b_1i)Z+(a_2+b_2i)Z^2+---+(a_n+b_ni)Z^n$.
   b. Press: CONT.

10. When "Icoef(I)=?" (For I=1 to degree of polynomial) is displayed:
   a. Enter the appropriate imaginary part of the coefficient as in 9.
   b. Press: CONT.
   c. Go to 9 if there are more coefficients to be entered.

11. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if a change in the coefficients is desired.
   b. Press: CONT.
   c. Go to 12.

or

a. Enter N if no changes in the coefficients are desired.
b. Press: CONT.
c. Go to 13.

12. When "COEFFICIENT NUMBER?" is displayed:
a. Enter the number of the coefficient to be changed.
b. Press: CONT.

13. When "Rcoef(I)?" (where I is the number of the coefficient to be changed) is displayed:
a. Enter the new value of Rcoef(I).
b. Press: CONT.

14. When "Icoef(I)?" (where I is the number of the coefficient to be changed) is displayed:
a. Enter the new value of Icoef(I).
b. Press: CONT.
c. Go to 11.

15. The real and imaginary parts of the roots will be printed. Any roots not found will contain the value 9.999999E99.

Example

```
DEGREE OF POLYNOMIAL= 4
MAX # OF ITERATIONS= 20
TOLERANCE FOR ROOTS= .00000001
TOLERANCE FOR FUNCTIONAL EVALUATIONS= .000001


COEFFICIENTS: [(Rcoef(0)+Icoef(0)*I)+(Rcoef(1)+Icoef(1)*I)*X^1+...]

         REAL              IMAGINARY

   -1.600000E+01        0.000000E+00
    0.000000E+00        0.000000E+00
    0.000000E+00        0.000000E+00
    0.000000E+00        0.000000E+00
    1.000000E+00        0.000000E+00




ROOTS:
         REAL              IMAGINARY


   -2.000000E+00        0.000000E+00
    2.000000E+00        0.000000E+00
    0.000000E+00       -2.000000E+00
    0.000000E+00        2.000000E+00
```

This section contains routines for numerically evaluating integrals. There are many possible reasons for using numerical integration, three of which are:

1) The analytical form of a simple integral can be quite complicated. For example,

$$\int dx/\sqrt{(x^2\ (x^2+25)^3)} = -1/(50x^2\ \sqrt{(x^2+25)}) - 3/(2(5^4)\ \sqrt{x^2+25})$$

$$+(3/(2(5^5)))\log(5+\ \sqrt{x^2+25}/x)$$

The evaluation of this expression may involve more calculations than evaluation by numerical methods.

2) Many integrals cannot be expressed in finite form. For example, $\int e^{-x^2} dx$.

3) The integrand may not be known explicitly, but may be expressed as a set of collocation points.

NUMERICAL INTEGRATION: SIMPSON'S RULE

Subprogram Name: Simp

This subprogram approximates $\int_a^b F(X)dx$ for the user-defined continuous function F(X) on the interval [a,b]. The user is required to specify the maximum number of iterations permitted and the error tolerance between successive calculations of the integral.

Subprogram Utilization:

File Name - "Simp", cartridge 1

Calling Syntax - CALL Simp (Low, Up, Int, Flg, Itmax, Tol)

Input Parameters:

Low	Lower bound of interval of integration.
Up	Upper bound of interval of integration.
Flg	Should intermediate results be printed: Flg = 1 implies yes Flg = -1 implies no
Itmax	Maximum number of interval halvings.
Tol	Error tolerance between successive calculations of integral.

Subprograms Required:

FNF(X)	Function to be integrated must be defined in function subprogram FNF(X).

Output Parameters:

Int	Value of integral.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, a BAD DATA CHECK is performed. If the program detects "nonsense" data, the program execution will pause and the following error message will be printed:

   "ERROR IN SUBPROGRAM Simp"

   Up = , Low = , Flg =

   Itmax = , Tol =

   The data may be corrected from the keyboard (e.g., Tol = 1E-4, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. If the maximum number of iterations is exceeded, the following error message will be printed:

   "MAXIMUM # OF ITERATIONS EXCEEDED."

   Int = , Intold =

   Tol = , Itmax =

   At this point, the tolerance or the maximum number of iterations may be increased from the keyboard (e.g., Itmax = 30, EXECUTE) or the value of the integral may be accepted as is.

Methods and Formulae:

This subprogram will approximate $\int_a^b F(X)dx$ for the user-defined function F(X) which must be defined in a function subprogram. The function must be continuous over the interval [a,b].

The method used is Simpson's one-third rule with truncation error $O(h^4)$ where h is the interval size.

The stopping criterion for this method is either a maximum number of interval halvings or successive computations of the integral differing by less than some user-supplied error tolerance.

Simpson's one-third Rule:

$$\int_a^b f(x)dx \approx \frac{h}{3}[f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+\cdots+4f(a+(n-1)h)+f(a+nh)]$$

where n = number of intervals,

$h = \frac{(b-a)}{n}$ = interval size

References:

1. Beckett, Royce and Hurt, James, Numerical Calculations and Algorithms (New York: McGraw Hill, 1967), pp. 166-169.

Driver Utilization:

File Name - "SIMP"

The driver "SIMP" sets up the necessary input parameters for the subprogram Simp and prints the resulting value of the integral. Intermediate integral values may also be printed at the user's option.

With each question, the user is only required to enter the appropriate response. The driver also sets up the user-defined function subprogram.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "SIMP".
   a. Type: GET "SIMP"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Simp ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Simp contained in file "Simp" will be linked on at the end of the driver.

5. When "HAS FUNCTION ALREADY BEEN DEFINED (Y/N)?" is displayed:
   a. Enter Y if the function has already been defined on the previous run.
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter N if the function has not yet been defined.
   b. Press: CONT.
   c. Go to 6.

6. When "FUNCTION [e.g., F=(X-3)*(X+4)]?" is displayed:
   a. Type in the user-defined function to be integrated in the format F =___.
   b. Press: CONT.
   c. At this point, the user-defined function will be properly entered in function subprogram FNF(X). This subprogram will then be stored on the mass storage device in file "KRYWEN" and then linked onto the program after subprogram Simp. (File "KRYWEN" must have been created previously to running this program. Statement: CREATE "KRYWEN", 4,84 does this. File "KRYWEN" has already been created on the NUMERICAL ANALYSIS cartridge.)
   d. Go to 7.

7. When "LOWER BOUND?" is displayed:
   a. Enter the lower bound of the interval.
   b. Press: CONT.

8. When "UPPER BOUND?" is displayed:
   a. Enter the upper bound of the interval.
   b. Press: CONT.

9. When "PRINT INTERMEDIATE DATA POINTS (Y/N)?" is displayed:
   a. Enter Y if intermediate data points are desired.
   b. Press: CONT.
   c. Go to 10.

or

   a. Enter N if intermediate data points are not desired.
   b. Press: CONT.
   c. Go to 14.

10. When "MAX # OF INTERVAL HALVINGS?" is displayed:
   a. Enter the maximum number of interval halvings. The evaluation of the integral will be made on 2 subintervals, then 4,16,32,---, halving the interval size on each iteration.
   b. Press: CONT.

11. When "ERROR TOLERANCE?" is displayed:
   a. Enter the error tolerance. The value of the integral is accepted if the difference in value of two successive approximations is less than this tolerance.
   b. Press: CONT.

12. The program will print the value of the integral as well as intermediate data points if they were requested.

EXAMPLE

```
F=X*X*SIN(3*X)


LOWER BOUND= 0
UPPER BOUND= 1.0471975512
MAX # OF INTERVAL HALVINGS= 10
ERROR TOLERANCE= .000001

# INTERVAL=    2      INTEGRAL= 1.913968E-01
# INTERVAL=    4      INTEGRAL= 2.170216E-01
# INTERVAL=    8      INTEGRAL= 2.173795E-01
# INTERVAL=   16      INTEGRAL= 2.173921E-01
# INTERVAL=   32      INTEGRAL= 2.173927E-01


INTEGRAL= 2.173927E-01
```

HEWLETT hp PACKARD

FILON'S METHOD

Subprogram Name: Filon

This subprogram approximates $\int_a^b F(X)\sin(tx)dx$ or $\int_a^b F(X)\cos(tx)dx$ for the user-defined continuous function F(X) on the interval [a,b]. Both integrals are found using Filon's method, a special case of Simpson's rule for oscillating functions.

The user is required to specify an array of function values, the number of points of evaluation, the number of oscillatiions, t, and the endpoints of the interval.

Subprogram Utilization:

File Name - "Filon", cartridge 1

Calling Syntax - CALL Filon (F(*), T,A,B,Nb,Key,Int)

Input Parameters:

F(*)	Vector containing table of functional values calculated at Nb equidistant points from A to B; subscripted from 1 to Nb.
T	Number of oscillations of the trigonometric function, i.e., sin(tx) or cos(tx).
A,B	Endpoints of interval.
Nb	Number of equidistant points from A to B; Nb must be odd and Nb > T+1.
Key	Key = 1 implies cos integral is evaluated. Key = -1 implies sin integral is evaluated.

Subprograms Required:

SUB	Cnsts (Alpha, Beta, Gamma, Theta) Subprogram which computes the three constants for use in Filon integration formula.

Output Parameters:

Int	Value of the integral.
Snat	Sin(at)
Snbt	Sin(bt)
Theta	T*H

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, a BAD DATA CHECK if performed. If the program detects "nonsense" data, the program execution will pause and the following error message will be printed:

   "ERROR IN SUBPROGRAM Filon"
   T = , A = , B =
   Nb = , Key =

   The data may be corrected from the keyboard (e.g., Nb=71, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. In order to use Filon's Formula, an odd number of points, Nb, is required. If the subprogram detects an even number of points, an error message is printed and execution pauses.

   "ERROR IN SUBPROGRAM Filon"
   "ODD # OF POINTS REQUIRED."
   Nb =

   Again, the data may be corrected from the keyboard. When CONT is pressed, the program will resume execution at the next line.

3. For a good approximation of the integral, Theta values (Theta = T*H) should be kept less than 1. So, for example, if the user wants to integrate $\int_0^1 x^2 \sin(10x)dx$, T=10. Nb should be chosen greater than 11 since:
$10H<1 \quad H<\frac{1}{10} \quad \frac{(B-A)}{Nb-1} < \frac{1}{10} \quad \frac{1}{Nb-1} < \frac{1}{10} \quad Nb > 11.$
Care should also be taken not to choose too large an Nb since this increases the running time of the program.

4. The vector F(*) must be dimensioned in the calling program: DIM F(1:Nb) where Nb is the number of equidistant points of the interval of integration.

Methods and Formulae:

This subprogram computes $\int_a^b F(X)\cos(tx)dx$ or $\int_a^b F(X)\sin(tx)dx$. Both integrals are found using Filon's method, a special case of Simpson's Rule.

It is assumed that F is an array of functional values calculated at Nb(odd) equidistant points from a to b and h is the interval size, $h = \frac{(b-a)}{Nb-1}$.

$$\int_a^b F(X)\sin(tx)dx \approx h[\alpha S_\alpha + \beta S_\beta + \gamma S_\gamma]$$

$$\int_a^b F(X)\cos(tx)dx \approx h[\alpha C_\alpha + \beta C_\beta + \gamma C_\gamma]$$

where $\alpha = [t^2h^2 + th\sin(th)\cos(th) - 2\sin(th)^2]/(th)^3$

$\beta = 2[th(1+\cos(th)^2 - 2\sin(th)\cos(th)]/(th)^3$

$\gamma = 4[\sin(th) - th\cos(th)]/(th)^3$

$S_\alpha = -[f(b)\cos(bt) - f(a)\cos(at)]$

$S_\beta = .5[f(a)\sin(at) - f(b)\sin(bt)]$

$$+ \sum_{i=1}^{\frac{Nb-1}{2}} f(2_i+1)\sin[(a+2ih)t]$$

$$S_\gamma = \sum_{i=1}^{\frac{Nb-1}{2}} f(2i)\sin[(a+(2i-1)h)t]$$

$$C_\alpha = f(b)\sin(bt)-f(a)\sin(at)$$

$$C_\beta = .5[f(a)\cos(at)-f(b)\cos(bt)] + \sum_{i=1}^{\frac{Nb-1}{2}} f(2i+1)\cos[(a+2ih)t]$$

$$C_\gamma = \sum_{i=1}^{\frac{Nb-1}{2}} f(2i)\cos[(a+(2i-1)h)t]$$

Driver Utilization:

File Name - "FILON", cartridge 1

The driver "FILON" sets up the necessary input parameters for the subprogram Filon and prints the resulting value of the integral.

With each question, the user is only required to enter the appropriate response. The driver also sets up the user-defined function subprogram.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "FILON"
   a. Type: GET "FILON"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Filon ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Filon contained in file "Filon" will be linked on at the end of the driver.

5. When "HAS FUNCTION ALREADY BEEN DEFINED (Y/N)?" is displayed:
   a. Enter Y if the function has already been defined on the previous run.
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter N if the function has not yet been defined.
   b. Press: CONT.
   c. Go to 6.

6. When "FUNCTION[e.g., F=(X-3)*(X+4)]?" is displayed:
   a. Type in the user-defined function, i.e., F(X) in $\int$ F(X) sin (tx) dx or $\int$ F(X) cos (tx) dx in the format F =____.
   b. Press: CONT.
   c. At this point, the user-defined function will be properly entered in function subprogram FNF(X). This subprogram will then be stored on the mass storage device in file "KRYWEN" and then linked onto the program after subprogram Filon. (File "KRYWEN" must have been created previously to running this program. Statement: CREATE "KRYWEN", 4,84 does this. File "KRYWEN" has already been created on the NUMERICAL ANALYSIS cartridge.
   d. Go to 7.

7. When "# OF POINTS OF INTEGRATION [ODD # REQUIRED]?" is displayed:
   a. Enter the number of points of integration. This must be odd. See the Special Considerations section of Filon for details on the number of points necessary for a reasonable estimate.
   b. Press: CONT.

8. When "SIN OR COS (SIN = -1, COS = 1)?" is displayed:
   a. Enter -1 if the integrand is of the form F(X)sin(tx).
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter 1 if the integrand is of the form F(X)cos(tx).
   b. Press: CONT.
   c. Go to 7.

9. When "LOWER BOUND?" is displayed:
   a. Enter the lower bound of the interval.
   b. Press: CONT.

10. When "UPPER BOUND?" is displayed:
   a. Enter the upper bound of the interval.
   b. Press: CONT.

11. When "FREQUENCY?" is displayed:
   a. Enter the frequency, t, of F(X)sin(tx) or F(X)cos(tx).
   b. Press: CONT.

12. The program will then print the value of the integral.

EXAMPLE

```
EXAMPLE 1

DEFINE F(X) WHERE F(X)*SIN(T*X) OR F(X)*COS(T*X) IS TO BE INTEGRATED.


F=X*X


# OF POINTS OF INTEGRATION= 31
Key= 1
LOWER BOUND= 0
UPPER BOUND= 1.5707963268
FREQUENCY= 1


INTEGRAL= 4.674011E-01



EXAMPLE 2

DEFINE F(X) WHERE F(X)*SIN(T*X) OR F(X)*COS(T*X) IS TO BE INTEGRATED.


F=LOG(1+X)


# OF POINTS OF INTEGRATION= 31
Key=-1
LOWER BOUND= 0
UPPER BOUND= 6.2831853072
FREQUENCY= 10


INTEGRAL=-1.976604E-01







# OF POINTS OF INTEGRATION= 71
Key=-1
LOWER BOUND= 0
UPPER BOUND= 6.2831853072
FREQUENCY= 10


INTEGRAL=-1.976264E-01
```

HEWLETT hp PACKARD

CAUTIOUS ADAPTIVE ROMBERG EXTRAPOLATION

Subprogram Name: Cadre

This subprogram uses cautious adaptive Romberg extrapolation to approximate $\int_a^b F(X)\,dx$ for the user-defined function F(X) on the interval [a,b].

The user is required to specify the lower and upper bounds, the relative and absolute error tolerances and the user defined function subprogram.

Subprogram Utilization:

File Name - "Cadre", cartridge 1

Calling Syntax - CALL Cadre (A,B,Aerr,Rerr,Err,Flg,Cadre)

Input Parameters:

A	Lower bound of interval of integration.
B	Upper bound of interval of integration.
Aerr	Desired absolute error in the answer.
Rerr	Desired relative error in the answer; Rerr must be in the range (Ø,Ø.1); e.g., Rerr = Ø.1 indicates that the estimate of the intergral is to be correct to one digit, whereas Rerr = 1E-4 calls for four digits of accuracy.

Subprograms Required:

FNF(X)	Function to be integrated must be defined in the function subprogram FNF(X).

Output Parameters:

Err	Estimated bound on the absolute error of the integral.
Flg	An integer between 1 and 5 indicating what difficulties were met with specifically. =1, all is well =2, one or more singularities were successfully handled =3, in some subintervals, the estimate 'Vint' was accepted merely because 'Err' was small even though no regular behavior could be recognized.

=4, failure, overflow of stack Ts.
=5, failure, too small a subinterval is required. This may be due to too much noise in the function (relative to the given error requirements) or due to a poorly behaved integrand.

Cadre — Value of the integral.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, a BAD DATA CHECK is performed. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Cadre".
   A = , B = , Aerr =
   Rerr = ,

   The data may be corrected from the keyboard (e.g., Aerr = 1E-6, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. "Cadre" can, in many cases, handle jump discontinuities and certain algebraic discontinuities. See reference 1 for full details.

3. The relative error parameter Rerr must be in the interval [Ø,Ø.1]. For example, Rerr=Ø.1 indicates that the estimate of the integral is to be correct to one digit, whereas Rerr=1E-4 calls for four digits of accuracy.

4. The absolute error parameter, Aerr, should be nonnegative. In order to give a reasonable value for Aerr, the user must know the approximate magnitude of the integral being computed. In many cases, it is satisfactory to use Aerr=Ø. In this case, only the relative error requirement is satisfied in the computation.

5. In order to allow this subprogram to run on a machine with 8k bytes, DeBoor's original program has been modified in a number of places. Most error messages have been deleted and the size of some of the arrays was shortened. Reference 1 contains the original version of the algorithm.

6. The variable Flg indicates the suggested reliability of the solution. To quote DeBoor:
"A very cautious man would accept Int only if Flg is 1 or 2. The merely reasonable man would keep the faith even if Flg is 3. The adventurous man is quite often right in accepting Int even if Flg is 4 or 5."

Methods and Formulae:

The subprogram Cadre attempts to solve the following problem: given the real function f, two real numbers A and B, and two nonnegative numbers Aerr and Rerr, find a number Int such that

$$\left|\int_a^b f(x)dx - Int\right| <= MAX(Aerr,\ Rerr*\left|\int_a^b f(x)dx\right|)$$

For this, the subprogram employs an adaptive scheme, whereby Int is found as the sum of estimates for the integral of f(x) over suitably small subintervals of the given interval of integration. Starting with the interval of integration itself as the first such subinterval, Cadre attempts to find an acceptable estimate on a given subinterval by cautions Romberg extrapolation. If this attempt fails, the subinterval is divided into two subintervals of equal length, each of which is now considered separately. For the sake of economy, values of f(x), once calculated, are saved until they are successfully used in estimating the integral over some subinterval to which they belong. For a more detailed description of this algorithm, see reference 1.

References:

1. DeBoor, Carl, "CADRE: An Algorithm for Numerical Quadrature", Mathematical Software (John R. Rice, Ed.), New York, Academic Press, 1971, Chapter 7.

Driver Utilization:

File Name - "CADRE"

The driver "CADRE" sets up the necessary input parameters for the subporgram Cadre and prints the resulting value of the integral, the estimated error term and the suggested reliability of the results.

With each question, the user is only required to enter the appropriate response. The driver also sets up the user-defined function subprogram.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "CADRE".
   a. Type: GET "CADRE"
   b. Press: EXECUTE

3. Press: RUN

4. When "HAS Cadre ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subporgram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Cadre contained in file "Cadre" will be linked on at the end of the driver.

5. When "HAS FUNCTION ALREADY BEEN DEFINED (Y/N)?" is displayed:
   a. Enter Y if the function has already been defined on the previous run.
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter N if the function has not yet been defined.
   b. Press: CONT.

6. When "FUNCTION [e.g., F=(X-3)*(X+4)]?" is displayed:
   a. Type in the user-defined function to be integrated in the above format.
   b. Press: CONT.
   c. At this point, the user-defined function will be properly entered in function subprogram FNF(X). This subprogram will then be stored on the mass storage device in file "KRYWEN" and then linked onto the program after subprogram Cadre. (File "KRYWEN" must have been created previously to running this program. Statement: CREATE "KRYWEN", 4,84 does this. File "KRYWEN" has already been created on the NUMERICAL ANALYSIS cartridge.)

7. When "LOWER BOUND?" is displayed:
   a. Enter the lower bound of the interval of integration.
   b. Press: CONT.

8. When "UPPER BOUND?" is displayed:
   a. Enter the upper bound of the interval of integration.
   b. Press: CONT.

9. When "ABSOLUTE ERROR?" is displayed:
   a. Enter the absolute error tolerance. If the user is not sure of the magnitude of the integral, Aerr may be set equal to zero. The program will then ignore this error tolerance.
   b. Press: CONT.

10. When "RELATIVE ERROR?" is displayed:
   a. Enter the relative error tolerance. This must be in the range (Ø,Ø.1); e.g., Rerr=Ø.1 indicates that the estimate of the integral is to be correct to one digit, whereas Rerr=1E-4 calls for four digits of accuracy.
   b. Press: CONT.

11. The program will print the integral value, the estimated error term and the suggested reliability of the results.

```
EXAMPLE 1

F=X*X*SIN(3*X)


LOWER BOUND= 0
UPPER BOUND= 1.0471975512
ABSOLUTE ERROR= .0000000001
RELATIVE ERROR= .00000001


INTEGRAL= 2.173928E-01
ESTIMATED ERROR= 4.557969E-11
FLG=1
```

```
EXAMPLE 2

F=LOG(EXP(1)/X)


LOWER BOUND= .00000000001
UPPER BOUND= 1
ABSOLUTE ERROR= .00000001
RELATIVE ERROR= .000001


INTEGRAL= 2.000000E+00
ESTIMATED ERROR= 2.419857E-06
FLG=2
```

HEWLETT hp PACKARD

EQUALLY-SPACED DATA POINTS

Subprogram Name: Inteq

This subprogram approximates $\int_a^b F(X)dx$ where F(X) is represented by discrete function values $F(X_i)$ at equally spaced points X_i.

The user is required to supply the increment between intervals, the number of data points, which must be odd, and the functional value at each of the data points.

Subprogram Utilization:

File Name - "Inteq"

Calling Syntax - CALL Inteq (N,Inc,F(*),Int)

Input Parameters:

N	Number of data points; this must be odd.
Inc	The increment between equally-spaced data points.
F(*)	Vector containing the functional values in increasing order of domain value; subscripted from 1 to N.

Output Parameters:

Int	The integral over the interval [F(1), F(N)].

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there are two error checks:
   If there are not at least three data points, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Inteq"
   "AT LEAST THREE DATA POINTS ARE REQUIRED."
   N =

   If there is not an odd number of data points, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Integ"
   "ODD # OF DATA POINTS IS REQUIRED."
   N =

2. Vector F(*) must be dimensioned in the calling program: DIM F(1:N) where N is the number of data points being entered.

Methods and Formulae:

This subprogram approximates $\int_a^b f(x)dx$ where discrete values of f(x) are known at equally-spaced points over the interval [a,b]. Simpson's one-third rule is used to approximate the integral.

$$\int_a^b f(x)dx \approx \frac{h}{3}\{f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+\cdots +4f(a+(n-1)h)+f(a+nh)\}$$

where: n = number of intervals (number of data points minus one)

$$h = \frac{b-a}{n}$$

References:

1. Beckett, Royce and Hunt, James Numerical Calculations and Algorithms (New York: McGraw-Hill, 1967), pp. 166-169.

Driver Utilization:

File Name - "INTEQ", cartridge 1

The driver "INTEQ" sets up the necessary input parameters for the subprogram Inteq and prints the resulting value of the integral.

With each question, the user is only required to enter the appropriate response.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "INTEQ".
   a. Type: GET "INTEQ"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Inteq ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Inteq contained in file "Inteq" will be linked on at the end of the driver.

5. When "# OF DATA POINTS?" is displayed:
   a. Enter the number of data points. Since Simpson's Rule is employed, there must be an odd number of points.
   b. Press: CONT.
   c. At this point, subprogram Inteq 1 is called to dynamically set up array F(*) to hold the data points.

6. When "INCREMENT?" is displayed:
   a. Enter the increment between data points.
   b. Press: CONT.

7. When "F(I)?" (FOR I=1 TO N) is displayed:
   a. Enter the appropriate function value of the data point.
   b. Press: CONT.
   c. To to 6 if more points are to be entered.

8. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired.
   b. Press: CONT.

or

   a. Enter N if no changes are necessary.
   b. Press: CONT.
   c. Go to 11.

9. When "DATA POINT NUMBER?" is displayed:
   a. Enter the data point number of the point to be changed.
   b. Press: CONT.

10. When "F(I)?" is displayed:
   a. Enter the function value of the data point.
   b. Press: CONT.
   c. Go to 8.

11. The program will print the value of the integral.

```
EXAMPLE 1


# OF DATA POINTS= 11
INCREMENT= .1

FUNCTION VALUES:

   F( 1)= 0.000000E+00
   F( 2)= 1.000000E-01
   F( 3)= 2.000000E-01
   F( 4)= 3.000000E-01
   F( 5)= 4.000000E-01
   F( 6)= 5.000000E-01
   F( 7)= 6.000000E-01
   F( 8)= 7.000000E-01
   F( 9)= 8.000000E-01
   F(10)= 9.000000E-01
   F(11)= 1.000000E+00

INTEGRAL= 5.000000E-01



EXAMPLE 2

# OF DATA POINTS= 9
INCREMENT= .25

FUNCTION VALUES:

   F( 1)= 0.000000E+00
   F( 2)= 2.800000E+00
   F( 3)= 3.800000E+00
   F( 4)= 5.200000E+00
   F( 5)= 7.000000E+00
   F( 6)= 9.200000E+00
   F( 7)= 1.210000E+01
   F( 8)= 1.560000E+01
   F( 9)= 2.000000E+01

 INTEGRAL= 1.641667E+01
```

Subprogram Name: Intun

This subprogram approximates $\int_a^b F(X)$ where F(X) is represented by discrete functional values for unequally-spaced domain values X over the interval [a,b]. The user is required to input the data points and the tolerance desired.

Subprogram Utilization:

File Name - "Intun"

Calling Syntax - CALL Intun (N,Eps,X(*),Y(*),Int)

Input Parameters:

N	Number of data points; N must be at least 3.
Eps	Epsilon tolerance for the solution of the simultaneous equations used in the calculation of the integral.
X(*)	Vector containing the domain values of the data points subscripted from 1 to N; domain values must be in increasing order.
Y(*)	Vector containing the range values ofthe data points subscripted from 1 to N.

Output Parameters:

Int	Value of the integral.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Intun".

   N = , X(1) = , X(N) =

   The data may be corrected from the keyboard (e.g., N = 21, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. For a derivation of the equations and the flow chart upon which this program is based, see Greville's article in reference 1. Reference 2 explains the theory of spline functions in general with applications in various other areas.

3. Vectors X(*) and Y(*) must be dimensioned in the calling program:

   DIM X(1:N), Y(1:N) where N is the number of data points to be used.

4. The number of data points must be greater than 2, otherwise an error will occur in the dimension statement in line 210

   DIM B(2:N-1), S(N), G(2:N-1) of the subprogram.

Methods and Formulae:

This subprogram approximates $\int_a^b f(x)dx$ where f(x) is represented by discrete functional values for unequally-spaced domain values x over the interval [a,b].

The method implemented involves fitting a curve through the data points and integrating that curve. The curve used is the cubic natural spline function which derives its name from a draftsman's mechanical spline. If the spline is considered as a function represented by s(x), the second derivative s''(x) approximates the curvature. For the curve through data points (x_1,y_1), (x_2,y_2), ..., (x_n,y_n) we want $\int_{x_1}^{x_n}(s''(x))^2dx$ to be minimized in order to achieve the "smoothest" curve.

The spline function with minimum curvature has cubic polynomials between adjacent data points. Adjacent polynomials are joined continuously with continuous first and second derivatives as well as $s''(x_1)=s''(x_n)=\emptyset$.

The procedure to determine s(x) involves the iterative solution of a set of simultaneous linear equations by the method of successive overrelaxation. The accuracy to which these equations are solved is specified by the user. For a detailed discussion of the algorithm, see reference 2.

$$\int_{x_i}^{x_{i+1}} s(x)dx \approx \sum_{i=1}^{n-1} \frac{1}{2}(x_{i+1}-x_i)(y_i+y_{i+1}) - \frac{1}{24}(x_{i+1})^3\, s''(x_i)+s''(x_{i+1})$$

References:

1. Ralston and Wilf, Mathematical Methods for Digital Computers, Vol. II (New York: John Wiley and Sons, 1967) pp. 156-158.

2. Greville, T.N.E., Editor, "Proceedings of An Advanced Seminar Conducted by the Mathematics Research Center", U.S. Army, University of Wisconsin, Madison. October 7-9,1968. Theory and Application of Spline Functions (New York, London: Academic Press, 1969), pp. 156-167.

Driver Utilization:

File Name - "INTUN", cartridge 1

The driver "INTUN" sets up the necessary input parameters for the subprogram Intun as well as prints out the resulting outputs.

With each question, the user is only required to input the appropriate response.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "INTUN".
   a. Type: GET "INTUN"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Intun ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Intun contained in file "Intun" will be linked on after the driver.

5. When "# OF DATA POINTS?" is displayed:
   a. Enter the number of data points to be used in the computation.
   b. Press: CONT.
   c. At this point, subprogram Intun 1 is called to dynamically set up the data point array.

6. When "ERROR TOLERANCE?" is displayed:
   a. Enter the error tolerance desired for use in solving the system of equations used in Intun. A generally acceptable value would be 1E-6.
   b. Press: CONT.

7. When "X(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the X coordinate of the Ith data point. These values must be entered in increasing order.
   b. Press: CONT.

8. When "Y(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the Y coordinate of the Ith data point.
   b. Press: CONT.
   c. If more data points are to be entered, go to 7.

9. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the data.
   b. Press: CONT.
   c. Go to 10.

or

   a. Enter N if no changes are desired in the data.
   b. Press: CONT.
   c. Go to 13.

10. When "DATA POINT #?" is displayed:
    a. Enter the data point number of the point to be changed.
    b. Press: CONT.

11. When "X(I)?" is displayed:
    a. Enter the X coordinate of the Ith data point.
    b. Press: CONT.

12. When "Y(I)?" is displayed:
    a. Enter the Y coordinate of the Ith data point.
    b. Press: CONT.
    c. Go to 9.

13. The program will then print the endpoints of the interval of integration and the value of the integral.

EXAMPLE

```
EXAMPLE 1



# OF DATA POINTS= 5
ERROR TOLERANCE= .000001

DATA POINTS: [DOMAIN VALUES MUST BE IN INCREASING ORDER]

   X( 1)= 0.000000E+00     Y( 1)= 0.000000E+00
   X( 2)= 1.000000E+00     Y( 2)= 1.000000E+00
   X( 3)= 2.000000E+00     Y( 3)= 2.000000E+00
   X( 4)= 3.000000E+00     Y( 4)= 3.000000E+00
   X( 5)= 4.000000E+00     Y( 5)= 4.000000E+00

INTEGRAL FROM  0.000 TO  4.000 IS  8.000000E+00



EXAMPLE 2



# OF DATA POINTS= 8
ERROR TOLERANCE= .000001

DATA POINTS: [DOMAIN VALUES MUST BE IN INCREASING ORDER]

   X( 1)= 0.000000E+00     Y( 1)= 0.000000E+00
   X( 2)= 2.000000E-01     Y( 2)= 4.000000E-02
   X( 3)= 6.000000E-01     Y( 3)= 3.600000E-01
   X( 4)= 1.000000E+00     Y( 4)= 1.000000E+00
   X( 5)= 1.100000E+00     Y( 5)= 1.210000E+00
   X( 6)= 1.500000E+00     Y( 6)= 2.250000E+00
   X( 7)= 1.600000E+00     Y( 7)= 2.560000E+00
   X( 8)= 2.000000E+00     Y( 8)= 4.000000E+00

INTEGRAL FROM  0.000 TO  2.000 IS  2.669976E+00
```

HEWLETT hp PACKARD

ORDINARY DIFFERENTIAL EQUATIONS: RUNGE-KUTTA METHOD

Subprogram Name: Kutta

This subprogram is a locally fifth-order Runge-Kutta procedure for solving systems of first-order ordinary differential equations.

The user must supply the dimension of the system of differential equations, the beginning and end points of integration, the initial value vector and the maximum number of integration steps.

NOTE: Before running this subprogram, the user must supply the subprogram Func containing the user-defined system of equations to be solved. If the included driver is used, subprogram Func must be stored on the default mass storage device in file "Func".

Subprogram Utilization:

File Name - "Kutta", cartridge 1

Calling Syntax - CALL Kutta (Idm,A,H,B,Maxstp,Ynt(*),Y(*)).

Input Parameters:

Idm	Dimension of the system of differential equations.
A	Integration starting point.
H	Integration step size.
B	Integration end point.
Maxstp	Maximum number of integration steps.
Ynt(*)	Initial value vector subscripted from 1 to Idm.

Subprograms Required:

Sub Func(Ysv(*),X,Idm,F(*))	Contains user-defined system of equations; used to generate new function values, F(*) from old function values, Ysv(*); see the Special Considerations section for further details.

Output Parameters:

Y(*)	Y(I,N) is the solution of Ith component evaluated at X=A+(N-1)*H; Y(*) needs to be dimensioned DIM Y (Idm,Nb) where Nb is the number of points of integration.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a check for bad data. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Kutta".

   Idm = , A = , B =

   H = , Maxstp =

   The data may be corrected from the keyboard (e.g., H=.01, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. Kutta is a good choice when accuracy requirements are not too high and function evaluations are simple. Kutta requires four function evaluations per integration step, compared with only two using the predictor-corrector method, "Adams".

3. The step size, h, must be chosen with some care. On the one hand, a small h keeps the error due to the Runge-Kutta formulae small. On the other hand, the smaller h is taken, the more integration steps we shall have to perform, and the greater the round off error is likely to be.

4. Since there may be some cumulative round off error in stepping from end points, A to B, i.e., X=A+(N-1)H, B may not be reached exactly. So, the subprogram is exited if X>B-Eps where Eps=1E-6. If X does not satisfy this condition, the subprogram will exit when the maximum number of steps have been performed.

5. A higher order differential equation may be replaced by a system of 1st order equations. Assume the equation is of the form:

$$\frac{d^m y}{dx^m} = F(X, Y, \frac{dy}{dx}, \frac{d^2 y}{dx^2}, \ldots, \frac{d^{m-1} y}{dx^{m-1}}$$

with the given initial conditions $y(x_0)$, $\frac{dy}{dx}(x_0)$,...,

$\frac{d^{m-1} y(x_0)}{dx^{m-1}}$. We may write the system as follows:

$$y_1' = F_1(x, y_1, y_2 \text{---}, y_m)$$
$$y_2' = F_2(x, y_1, y_2 \text{---}, y_m)$$
$$y_m = F_m(x, y_1, y_2 \text{---}, y_m)$$

EXAMPLE 1.

Compute the solution of van der Pol's equation

$y''-(.1)(1-y^2)y'+y = \emptyset$ with initial conditions
$y(\emptyset)=1$, $y'(\emptyset)=\emptyset$.

```
   Let Z=y'
   Then Z'= y''= (.1)(1-y^2)y'-y

In terms of our subprogram Func,
   y = Ysv(1) and y'= Ysv(2).

So, we would code "Func" as follows:
   SUB Func (Ysv(*),X,Idm,F(*))
   F(1)=Ysv(2)
   F(2)=(.1)*(1-Ysv(1)*Ysv(1))*Ysv(2)-Ysv(1)
   SUBEND

EXAMPLE 2.

Compute the solution of

   y''+xy^2=Ø with initial conditions y(Ø)=Ø
                                      y'(Ø)=1.

Let Z=y'
Then Z'=y''= -xy^2
In terms of our subprogram Func,
   y=Ysv(1) and y'=Ysv(2).
So, we would code "Func" as follows:
   SUB Func (Ysv(*),X,Idm,F(*))
   F(1) = Ysv(2)
   F(2) = X*Ysv(1)*Ysv(1)
   SUBEXIT
   SUBEND
```

EXAMPLE 3.

Compute the solution of

y'''+17y''-1Øy'+y=Ø with initial conditions
y(Ø)=y'(Ø)=y''(Ø)=Ø.

Let Z = y'
Then Z' = y''
Let W = Z' = y''
Then W' = Z'' = y''' = -17y''+1Øy' -y.

In terms of our subprogram Func,
y = Ysv(1), y' = Ysv(2) and y'' = Ysv(3).

So, we would code "Func" as follows:

```
SUB Func (Ysv(*),X,Idm,F(*))
F(1) = Ysv(2)
F(2) = Ysv(3)
F(3) = 17*Ysv(3)+1Ø*Ysv(2)-Ysv(1)
SUBEND
```

Methods and Formulae:

This subprogram solves the following system of simultaneous first-order ordinary differential equations:

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_n)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \ldots, y_n)$$

$$\vdots$$

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n)$$

with initial conditions x_0, $y_1(x_0)$, $y_2(x_0), \ldots, y_n(x_0)$ specified by the user.

The program may also be used to solve an equation of the form:

$$\frac{d^m y}{dx^m} = f\left(x, y, \frac{dy}{dx}, \frac{d^2 y}{dx^2}, \ldots, \frac{d^{m-1} y}{dx^{m-1}}\right)$$

with given initial conditions $y(x_0)$, $\frac{dy}{dx}(x_0), \ldots, \frac{d^{m-1} y}{dx^{m-1}}(x_0)$ by rewriting the equation as a system of first-order equations as above. A method for doing this is provided in the Special Considerations and Programming Hints section.

Runge-Kutta methods attempt to obtain greater accuracy, and at the same time avoid the need for higher derivatives, by evaluating the function F at selected points on each subinterval.

For the equation $y'=f(x,y), y(x_0)=y_0$ and step size h, approximations y_n to $y(x_0+nh)$ for $n=\emptyset,1,2,---$ are generated using the recursion formula

$$y_{n+1}=y_n+\frac{1}{6}(k_1+2k_2+2k_3+k_4)$$

where

$$k_1=hf(x_n,y_n)$$
$$k_2=hf(x_n+\frac{h}{2},\ y_n+\frac{1}{2}k_1)$$
$$k_3=hf(x_n+\frac{h}{2},\ y_n+\frac{1}{2}k_2)$$
$$k_4=hf(x_n+h,y_n+k_3)$$

The local truncation error is $O(h^5)$.

This is a single step method. It requires only the value of y at a point $x=x_n$ to find y and y' at $x=x_{n+1}$.

The above formulae may be generalized to any system of 1st order differential equations. Further details may be obtained from the references.

References:

1. Ralston and Wilf, Mathematical Methods for Digital Computers, Vol. 1 (New York: John Wiley and Sons, Inc., 1960), p. 115.

2. Carnahan, Luther, Wilkes, Applied Numerical Methods, (New York: John Wiley and Sons, Inc., 1969), pp. 363-366.

Driver Utilization:

File Name - "KUTTA", cartridge 1

The driver "KUTTA" sets up the necessary input parameters for the subprogram Kutta and prints out the resulting output.

With each question, the user is only required to input the appropriate response.

The user must store the subprogram Func on the mass storage device, using file name "Func".

NOTE: The driver prints every tenth value of the computed vectors. If a different number of values is required, line 770 may be changed, replacing the 10 by an appropriate number.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. The user is required to provide on the mass storage device the file "Func" containing subprogram Func. The driver program will link this subprogram onto subprogram Kutta. (See the Special Considerations section of subprogram Kutta for further details on how to set up subprogram Func.)

3. Get file "KUTTA".
   a. Type: GET "KUTTA"
   b. Press: CONT.

4. Press: RUN.

5. When "HAS Kutta ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 6.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Kutta contained in file "Kutta" will be linked on after the driver.

6. When "HAS Func ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram defining the system of equations to be solved has already been linked on.
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Func contained in file "Func" will be linked on after subprogram Kutta.

7. When "DIMENSION OF SYSTEM OF D.E.?" is displayed:
   a. Enter the dimension of the system of differential equations.
   b. Press: CONT.

8. When "LOWER BOUND?" is displayed:
   a. Enter the lower bound of the interval.
   b. Press: CONT.

9. When "UPPER BOUND?" is displayed:
   a. Enter the upper bound of the interval.
   b. Press: CONT.

10. When "STEP SIZE?" is displayed:
    a. Enter the step size for integrating.
    b. Press: CONT.

11. At this point, a computation is made to estimate the maximum size of the arrays required. A call is then made to Kutta, where the arrays are set up dynamically.

12. When "MAX # OF ITERATION STEPS?" is displayed:
    a. Enter the maximum number of iteration steps permitted. This enables the user to put an additional constraint on the number of integrations performed.
    b. Press: CONT.

13. When "INITIAL VALUES:" is printed and "Y(I)?" (FOR I=1 TO THE NUMBER OF DIMENSIONS OF THE SYSTEM) is displayed:
    a. Enter the Ith initial value.
    b. Press: CONT.
    c. Go to 13 if additional values are required.

14. The program will print the domain values X, as well as the values of the integrated function at X.

EXAMPLE

1.) Solve $y'-xy^{1/3}=\emptyset$ for $1 \leq x \leq 3$, step size = 0.01 subject to the initial condition y(1)=1. Print every tenth value.

```
10    SUB Func(Ysv(*),X,Idm,F(*))
20    F(1)=X*Ysv(1)^(1/3)
30    SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 1
LOWER BOUND= 1
UPPER BOUND= 3
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201

INITIAL VALUES:

   Y( 1)= 1.000000E+00

        X                    Y

    1.000000E+00      1.000000E+00
    1.100000E+00      1.106817E+00
    1.200000E+00      1.227880E+00
    1.300000E+00      1.364136E+00
    1.400000E+00      1.516565E+00
    1.500000E+00      1.686171E+00
    1.600000E+00      1.873982E+00
    1.700000E+00      2.081045E+00
    1.800000E+00      2.308421E+00
    1.900000E+00      2.557187E+00
    2.000000E+00      2.828427E+00
    2.100000E+00      3.123239E+00
    2.200000E+00      3.442725E+00
    2.300000E+00      3.787995E+00
    2.400000E+00      4.160166E+00
    2.500000E+00      4.560359E+00
    2.600000E+00      4.989698E+00
    2.700000E+00      5.449312E+00
    2.800000E+00      5.940333E+00
    2.900000E+00      6.463894E+00
    3.000000E+00      7.021132E+00
```

2.) Solve $y''+2yy'=\emptyset$ for $0 \leq x \leq 2$, step size = .01, subject to the initial conditions $y(\emptyset)=\emptyset, y'(\emptyset)=1$.

The solution may be obtained by first reducing the equation to a set of simultaneous equations using the following substitutions:

Let Z = y'
Then Z' = y''= -2yy'
Where y = Ysv(1) and y' = Ysv(2)

The subprogram Func is set up as follows:

```
10   SUB Func(Ysv(*),X,Idm,F(*))
20   F(1)=Ysv(2)
30   F(2)=-2*Ysv(1)*Ysv(2)
40   SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 2
LOWER BOUND= 0
UPPER BOUND= 2
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201
```

```
INITIAL VALUES:

   Y( 1)= 0.000000E+00
   Y( 2)= 1.000000E+00

        X                       Y

   0.000000E+00        0.000000E+00
   1.000000E-01        9.966799E-02
   2.000000E-01        1.973753E-01
   3.000000E-01        2.913126E-01
   4.000000E-01        3.799490E-01
   5.000000E-01        4.621172E-01
   6.000000E-01        5.370496E-01
   7.000000E-01        6.043678E-01
   8.000000E-01        6.640368E-01
   9.000000E-01        7.162979E-01
   1.000000E+00        7.615942E-01
   1.100000E+00        8.004990E-01
   1.200000E+00        8.336546E-01
   1.300000E+00        8.617232E-01
   1.400000E+00        8.853516E-01
   1.500000E+00        9.051483E-01
   1.600000E+00        9.216686E-01
   1.700000E+00        9.354091E-01
   1.800000E+00        9.468060E-01
   1.900000E+00        9.562375E-01
   2.000000E+00        9.640276E-01
```

3.) Solve $yy''+3(y')^2=\emptyset$ for $0 \leq x \leq .2$, step size = .01, subject to the initial conditions $y(\emptyset)=1$, $y'(\emptyset)=\frac{1}{4}$.

The solution may be obtained by first reducing the equation to a set of simultaneous equations using the following substitutions:

Let Z = y'

Then Z = y'' = $-3(y')^2y$

Where y = Ysv(1) and y' = Ysv(2).

The subprogram Func is set up as follows:

```
10   SUB Func(Ysv(*),X,Idm,F(*))
20   F(1)=Ysv(2)
30   F(2)=-3*Ysv(2)*Ysv(2)/Ysv(1)
40   SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 2
LOWER BOUND= 0
UPPER BOUND= 2
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201
```

INITIAL VALUES:

Y(1)= 1.000000E+00
Y(2)= 2.500000E-01

X	Y
0.000000E+00	1.000000E+00
1.000000E-01	1.024114E+00
2.000000E-01	1.046635E+00
3.000000E-01	1.067790E+00
4.000000E-01	1.087757E+00
5.000000E-01	1.106682E+00
6.000000E-01	1.124683E+00
7.000000E-01	1.141858E+00
8.000000E-01	1.158292E+00
9.000000E-01	1.174055E+00
1.000000E+00	1.189207E+00
1.100000E+00	1.203801E+00
1.200000E+00	1.217883E+00
1.300000E+00	1.231493E+00
1.400000E+00	1.244666E+00
1.500000E+00	1.257433E+00
1.600000E+00	1.269823E+00
1.700000E+00	1.281861E+00
1.800000E+00	1.293569E+00
1.900000E+00	1.304967E+00
2.000000E+00	1.316074E+00

4.) Solve $y''-(.1)(1-y^2)y'+y=\emptyset$ for $0 \leq x \leq .2$, step size = .Ø1, subject to the initial conditions $y(\emptyset)=1$, $y'(\emptyset)=\emptyset$.

The solution may be obtained by first reducing the equation to a set of simultaneous equations using the following substitutions:

Let Z = y'

Then $Z' = y'' = (.1)(1-y^2)y' - y$

Where y = Ysv(1) and y' = Ysv(2).

The subprogram Func is set up as follows:

```
10   SUB Func(Ysv(*),X,Idm,F(*))
20   F(1)=Ysv(2)
21   F(2)=.1*(1-Ysv(1)*Ysv(1))*Ysv(2)-Ysv(1)
30   SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 2
LOWER BOUND= 0
UPPER BOUND= 2
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201
```

```
INITIAL VALUES:

   Y( 1)= 1.000000E+00
   Y( 2)= 0.000000E+00
```

X	Y
0.000000E+00	1.000000E+00
1.000000E-01	9.950041E-01
2.000000E-01	9.800650E-01
3.000000E-01	9.553246E-01
4.000000E-01	9.210119E-01
5.000000E-01	8.774360E-01
6.000000E-01	8.249809E-01
7.000000E-01	7.641003E-01
8.000000E-01	6.953137E-01
9.000000E-01	6.192045E-01
1.000000E+00	5.364177E-01
1.100000E+00	4.476600E-01
1.200000E+00	3.536993E-01
1.300000E+00	2.553641E-01
1.400000E+00	1.535432E-01
1.500000E+00	4.918325E-02
1.600000E+00	-5.671498E-02
1.700000E+00	-1.631025E-01
1.800000E+00	-2.688912E-01
1.900000E+00	-3.729650E-01
2.000000E+00	-4.741948E-01

Subprogram Name: Adams

This subprogram is a locally fifth-order Adams-Bashford procedure for solving systems of first-order ordinary differential equations. Kutta, a Runge-Kutta procedure, is used as a starter. An optional Adams-Moulton corrector is also included.

The user is required to supply the dimension of the system of differential equations, the beginning and end points of integration, the initial value vector and the maximum number of integration steps.

NOTE: Before running this subprogram, the user must supply the subprogram Func containing the user-defined system of equations to be solved. If the included driver is used, subprogram Func must be stored on the default mass storage device in file "Func".

Subprogram Utilization:

File Name - "Adams", cartridge 1

Calling Syntax - CALL Adams (Idm,A,H,B,Maxstp,Crktr,Ynt(*),Y(*),Er(*))

Input Parameters:

Idm	Dimension of the system of differential equations
A	Integration starting point.
H	Integration step size.
B	Integration end point.
Maxstp	Maximum number of integration steps.
Crktr	Corrector flag: Crktr = 1 implies use corrector. Crktr = -1 implies do not use corrector.

Ynt(*)	Initial value vector, subscripted from 1 to Idm.

Subprograms Required:

SUB Kutta (Idm,A,H,B,Maxstp,Ynt(*),Y(*)
Used as a starter for subprogram Adams.

SUB Func (Ysv(*),X,Idm,F(*))
Contains user-defined system of equations; used to generate new function values F(*), from old function values, Ysv(*).

Output Parameters:

Y(*)	Y(I,N) is the solution of Ith component evaluated at X = A+(N-1)*H; Y(*) needs to be dimensioned DIM V(Idm, Nb) where Nb is the number of points of integration.
Er(*)	Error estimate for Y(I,N); Er(*) is subscripted from 1 to Nb where Nb is the number of points of integration.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Adams"
   Idm = , A = , B =
   H = , Crktr = , Maxstp =

   The data may be corrected from the keyboard (e.g., H=.Ø1, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. When functional evaluations are costly, Adams is a good choice since only two evaluations are made per integration step.

3. The step size, h, must be chosen with some care. On the one hand, a small h keeps the error due to the Adams-Bashford formulae small. On the other hand, the smaller h is taken, the more integration steps we shall have to perform, and the greater the rounding error is likely to be.

4. Since there may be some cumulative round off error in stepping from end points A to B, i.e., X = A+(N-1)H,B may not be reached exactly. So, the subprogram is exited if X > B-1E-8.

5. The Moulton corrector term is an available option. In most of the test cases tried, one or two digits of accuracy were gained. The cost is an additional 50-75% in running time.

Methods and Formulae:

This subprogram solves the following system of simultaneous first-order ordinary differential equations:

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_n)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \ldots, y_n)$$

$$\vdots$$

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n)$$

With initial conditions x_0, $y_1(x_0)$, $y_2(x_0),\ldots,y_n(x_0)$ specified by the user.

The program may also be used to solve an equation of the form:

$$\frac{d^m y}{dx^m} = f\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^{m-1}y}{dx^{m-1}}\right)$$

With given initial conditions $y(x_0)$, $\frac{dy}{dx}(x_0),\ldots,\frac{d^{m-1}y}{dx^{m-1}}(x_0)$ by rewriting the equation as a system of first-order equations as above. A method for doing this is provided in the Special Considerations and Programming Hints section.

For the equation $y' = f(x,y)$, $y(x_0)=y_0$ and step size, h, approximations y_n to $y(x_0+nh)$ for $n=0,1,2,\cdots$ are generated using the recursion formula

$$y_{n+1}=y_n+\frac{h}{24}(55f_n-59f_{n-1}+37f_{n-2}-9f_{n-3}).$$

where $f_i=f(x_i,y_i)$.

The local truncation error is $O(h^5)$.

This is a multistep method; thus, it is not self-starting. We must have four successive values of f(x,y) at equally spaced points before this formula can be used. The subprogram Kutta, a Runge-Kutta procedure, is used in Adams to generate these points.

The above formulae may be generalized to any system of first-order differential equations. Further details may be obtained from the references.

Driver Utilization:

File Name - "ADAMS", cartridge 1

The driver "ADAMS" sets up the necessary input parameters for the subprogram Adams as well as prints out the resulting output.

With each question, the user is only required to input the appropriate response.

NOTE: The user is required to provide on the mass storage device the file "Func" containing subprogram Func. The driver will link this subprogram onto subprogram Adams.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. The user is required to provide on the mass storage device the file "Func" containing subprogram Func. The driver program will link this subprogram onto subprogram Adams. (See the Special Considerations section of subprogram Adasm for further details on how to set up subprogram Func.)

3. Get file "ADAMS".
   a. Type: GET "ADAMS"
   b. Press: EXECUTE.

4. Press: RUN.

5. When "HAS Adams ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 6.

or

a. Enter N if the subprogram has not yet been linked on.
b. Press: CONT.
c. At this point, subprogram Adams contained in file "Adams" will be linked on after the device.

6. When "HAS Func ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram defining the system of equations to be solved has already been linked on.
   b. Press: CONT.
   c. Go to 7.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Func contained in file "Func" will be linked on after subprogram Adams.

7. When "DIMENSION OF SYSTEM OF D.E.?" is displayed:
   a. Enter the dimension of the system of differential equations.
   b. Press: CONT.

8. When "LOWER BOUND?" is displayed:
   a. Enter the lower bound of the interval under consideration.
   b. Press: CONT.

9. When "UPPER BOUND?" is displayed:
   a. Enter the upper bound of the interval under consideration.
   b. Press: CONT.

10. When "STEP SIZE?" is displayed:
   a. Enter the step size; i.e., the distance between each point of integration.
   b. Press: CONT.

11. At this point, a computation is made to estimate the maximum size of the arrays required. A call is then made to Adam 1 where the arrays are set up dynamically.

12. When "MAX # OF INTEGRATION STEPS?" is displayed:
    a. Enter the maximum number of integration steps permitted. This enables the user to put an additional constraint on the number of integrations performed.
    b. Press: CONT.

13. When "USE MOULTON CORRECTOR?" is displayed:
    a. Enter Y if the Moulton Corrector is desired. At the cost of additional time, the corrector can be expected to give one or possibly two additional digits of accuracy.
    b. Press: CONT.

or

    a. Enter N if the Moulton Corrector is not desired.
    b. Press: CONT.

14. When "INITIAL VALUES:" is printed and "Y(I)?" (FOR I=1 TO THE NUMBER OF DIMENSIONS OF THE SYSTEM) is displayed:
    a. Enter the Ith initial value.
    b. Press: CONT.
    c. Go to 14 additional values are required.

15. The program will print the domain values x, as well as the values of the integrated function at x.

EXAMPLE

1) See example 1 on p. 113 for a statement of the problem. Subprogram Func is set up as follows:

```
10    SUB Func(Ysv(*),X,Idm,F(*))
20    F(1)=X*Ysv(1)^(1/3)
30    SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 1
LOWER BOUND= 1
UPPER BOUND= 3
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201

INITIAL VALUES:

   Y( 1)= 1.000000E+00

        X                       Y

     1.000000E+00       1.000000E+00
     1.100000E+00       1.106817E+00
     1.200000E+00       1.227880E+00
     1.300000E+00       1.364136E+00
     1.400000E+00       1.516565E+00
     1.500000E+00       1.686171E+00
     1.600000E+00       1.873982E+00
     1.700000E+00       2.081045E+00
     1.800000E+00       2.308421E+00
     1.900000E+00       2.557187E+00
     2.000000E+00       2.828427E+00
     2.100000E+00       3.123239E+00
     2.200000E+00       3.442725E+00
     2.300000E+00       3.787995E+00
     2.400000E+00       4.160166E+00
     2.500000E+00       4.560359E+00
     2.600000E+00       4.989698E+00
     2.700000E+00       5.449312E+00
     2.800000E+00       5.940333E+00
     2.900000E+00       6.463894E+00
     3.000000E+00       7.021132E+00
```

EXAMPLE

2) See example 2 on p. 114 for a statement of the problem. Subprogram Func is set up as follows:

```
10    SUB Func(Ysv(*),X,Idm,F(*))
20    F(1)=Ysv(2)
30    F(2)=-2*Ysv(1)*Ysv(2)
40    SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 2
LOWER BOUND= 0
UPPER BOUND= 2
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201

INITIAL VALUES:

   Y( 1)= 0.000000E+00
   Y( 2)= 1.000000E+00

        X                       Y

 0.000000E+00       0.000000E+00
 1.000000E-01       9.966799E-02
 2.000000E-01       1.973753E-01
 3.000000E-01       2.913126E-01
 4.000000E-01       3.799490E-01
 5.000000E-01       4.621172E-01
 6.000000E-01       5.370496E-01
 7.000000E-01       6.043678E-01
 8.000000E-01       6.640368E-01
 9.000000E-01       7.162979E-01
 1.000000E+00       7.615942E-01
 1.100000E+00       8.004991E-01
 1.200000E+00       8.336547E-01
 1.300000E+00       8.617232E-01
 1.400000E+00       8.853517E-01
 1.500000E+00       9.051483E-01
 1.600000E+00       9.216686E-01
 1.700000E+00       9.354091E-01
 1.800000E+00       9.468061E-01
 1.900000E+00       9.562375E-01
 2.000000E+00       9.640276E-01
```

3) See example 3 on p. 116 for a statement of the problem. Subprogram Func is set up as follows:

```
10   SUB Func(Ysv(*),X,Idm,F(*))
20   F(1)=Ysv(2)
30   F(2)=-3*Ysv(2)*Ysv(2)/Ysv(1)
40   SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 2
LOWER BOUND= 0
UPPER BOUND= 2
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201

INITIAL VALUES:

   Y( 1)= 1.000000E+00
   Y( 2)= 2.500000E-01
```

X	Y
0.000000E+00	1.000000E+00
1.000000E-01	1.024114E+00
2.000000E-01	1.046635E+00
3.000000E-01	1.067790E+00
4.000000E-01	1.087757E+00
5.000000E-01	1.106682E+00
6.000000E-01	1.124683E+00
7.000000E-01	1.141858E+00
8.000000E-01	1.158292E+00
9.000000E-01	1.174055E+00
1.000000E+00	1.189207E+00
1.100000E+00	1.203801E+00
1.200000E+00	1.217883E+00
1.300000E+00	1.231493E+00
1.400000E+00	1.244666E+00
1.500000E+00	1.257433E+00
1.600000E+00	1.269823E+00
1.700000E+00	1.281861E+00
1.800000E+00	1.293569E+00
1.900000E+00	1.304967E+00
2.000000E+00	1.316074E+00

EXAMPLE

4) See example 4 on p. 118 for a statement of the problem. Subprogram Func is set up as follows:

```
10    SUB Func(Ysv(*),X,Idm,F(*))
20    F(1)=Ysv(2)
30    F(2)=.1*(1-Ysv(1)*Ysv(1))*Ysv(2)-Ysv(1)
40    SUBEND
```

```
DIMENSION OF SYSTEM OF D.E.= 2
LOWER BOUND= 0
UPPER BOUND= 2
STEP SIZE= .01
MAX # OF INTEGRATION STEPS= 201

INITIAL VALUES:

   Y( 1)= 1.000000E+00
   Y( 2)= 0.000000E+00

        X                       Y

    0.000000E+00        1.000000E+00
    1.000000E-01        9.950041E-01
    2.000000E-01        9.800650E-01
    3.000000E-01        9.553246E-01
    4.000000E-01        9.210119E-01
    5.000000E-01        8.774360E-01
    6.000000E-01        8.249809E-01
    7.000000E-01        7.641003E-01
    8.000000E-01        6.953137E-01
    9.000000E-01        6.192045E-01
    1.000000E+00        5.364177E-01
    1.100000E+00        4.476600E-01
    1.200000E+00        3.536993E-01
    1.300000E+00        2.553641E-01
    1.400000E+00        1.535432E-01
    1.500000E+00        4.918325E-02
    1.600000E+00       -5.671498E-02
    1.700000E+00       -1.631025E-01
    1.800000E+00       -2.688912E-01
    1.900000E+00       -3.729650E-01
    2.000000E+00       -4.741948E-01
```

Subprogram Name - Ludsht

This subprogram solves the system of equations AX=B using triangular decomposition. The triangular matrices L and U overwrite A and the solution matrix X overwrites B. This saves a significant amount of memory, but in the process, the values contained in matrices A and B are destroyed.

Subprogram Utilization:

File Name - "Ludsht"

Calling Syntax - CALL Ludsht (A(*),B(*),N,M)

Input Parameters:

*A(*)	Array containing the nonsingular matrix A; dimensioned A(1:N, 1:N).
*B(*)	Array containing the coefficient matrix B; dimensioned B(1:N, 1:M).
N	Number of rows in B(*).
M	Number of columns in B(*).

Subprograms Required:

Decomp	Computes the triangular decomposition of A(*) using Gaussian Elimination.
Solve	Given the triangular decomposition of A(*), Solve finds an approximate solution to a single system of equations Ax=b.

* A(*) and B(*) are both input and output parameters.

Output Parameters:

*A(*)	Array containing the Lu demonposition of A(*).
*B(*)	Array containing the solution matrix X to the system AX=B.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Ludsht."

   M = , N =

   The data may be corrected from the keyboard (e.g., N=5, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. One way to solve the system of linear equations Ax = b is to compute the inverse of A and then multiply A^{-1} by b. This method may appear especially attractive if several right-hand sides b_k are involved since the inverse need be computed only once. However, a set of linear equation solving procedures - such as DECOMP and SOLVE - can accomplish this task with fewer operations and with greater accuracy. Once L and U have been computed, the solution of LUx = b requires n^2 - n multiplications and n divisions or a total of n^2 multiplicative operations. Moreover, once A^{-1} has been computed, the evaluation of $A^{-1}b$ requires n^2 multiplicative operations also. Thus both methods require approximately the same number of operations

* A(*) and B(*) are both input and output parameters.

at this point. But the initial calculations of LU and of A^{-1} require about $\frac{1}{3}n^3$ and n^3 multiplicative operations, respectively.

Since the inverse routine is stored in firmware, inversion is probably faster than triangular decomposition. But, since there are fewer operations and hence fewer rounding errors, the use of this subprogram can be expected to give more accurate results in most cases.

3. This subprogram is designed to save as much storage as possible. But care must be taken since both matrix A and matrix B are destroyed in the subprogram.

Methods and Formulae:

Subprogram Ludsht begins by finding the triangular decomposition of A using Subprogram Decomp. The decomposition Lu(*) is stored in A(*).

AX=B is then solved one column at a time using subprogram Solve. The solution vector is then restored in B(*).

References:

1. Forsythe, G. and Moler, C. Computer Solution of Linear Algebraic Systems, (Englewood Cliffs, N.J.: Prentice Hall, Inc.), Ch. 9,11.

Driver Utilization:

File Name - LUDSHT

The driver "LUDSHT" sets up the necessary input parameters for subprogram Ludsht as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "LUDSHT"
   a. Type: GET "LUDSHT"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Ludsht ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Ludsht contained in file "Ludsht" is linked on after the driver.

5. When "DIMENSIONS OF B(*), [ROW,COLUMN]?" is displayed:
   a. Enter the dimensions of the coefficient matrix B, e.g., if B(*) has dimensions 3 by 4, then enter 3,4.
   b. Press: CONT.

6. At this point, subprogram Ludl is called to dynamically set up the arrays A(*) and B(*).

7. When "MATRIX A:" is printed and "A(I,J)?" (FOR I=1 TO # OF ROWS, J=1 TO NUMBER OF COLUMNS) is displayed:
   a. Enter the I,Jth element of matrix A.
   b. Press: CONT.
   c. If more elements are to be entered, go to 7.

8. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in matrix A.
   b. Press: CONT.
   c. Go to 9.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 11.

9. When "COORDINATES OF A(*) [ROW, COLUMN]?" is displayed:
   a. Enter the row and column numbers of the element to be changed, e.g., if element A(3,4) is to be changed, enter 3,4.
   b. Press: CONT.

10. When "A(I,J)?" (where I and J represent the row and column of the element to be changed) is displayed:
   a. Enter the new value of the element.
   b. Press: CONT.
   c. Go to 8.

11. When "MATRIX B:" is printed and "B(I,J)?" (FOR I=1 TO # OF ROWS IN B(*) FOR J=1 TO # OF COLUMNS IN B(*)) is displayed:
    a. Enter the Ith, Jth element of matrix B.
    b. Press: CONT.
    c. If more elements are to be entered, go to 11.

12. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are desired in matrix B.
    b. Press: CONT.
    c. Go to 13.

or

    a. Enter N if no changes are desired.
    b. Press: CONT.
    c. Go to 15.

13. When "COORDINATES OF B(*) [ROW, COLUMN]?" is displayed:
    a. Enter the row and column numbers of the element to be changed, e.g., if element B(3,4) is to be changed, enter 3,4.
    b. Press: CONT.

14. When "B(I,J)?" (WHERE I AND J REPRESENT THE ROW AND COLUMN OF THE ELEMENT TO BE CHANGED) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 12.

15. The program will then print the solution matrix X.

EXAMPLE

```
 EXAMPLE 1


DIMENSIONS OF B(*)= 4 X 2

MATRIX A:
 1.000000E+00 -2.000000E+00  3.000000E+00  1.000000E+00
-2.000000E+00  1.000000E+00 -2.000000E+00 -1.000000E+00
 3.000000E+00 -2.000000E+00  1.000000E+00  5.000000E+00
 1.000000E+00 -1.000000E+00  5.000000E+00  3.000000E+00


MATRIX B:

 3.000000E+00  1.000000E+00
-4.000000E+00  0.000000E+00
 7.000000E+00  0.000000E+00
 8.000000E+00  0.000000E+00


MATRIX X:
 1.000000E+00 -2.884615E-01
 1.000000E+00 -7.307692E-01
 1.000000E+00 -1.923077E-02
 1.000000E+00 -1.153846E-01
```

```
EXAMPLE 2


DIMENSIONS OF B(*)= 3 X 1

MATRIX A:

 3.300000E+01  1.600000E+01  7.200000E+01
-2.400000E+01 -1.000000E+01 -5.700000E+01
-8.000000E+00 -4.000000E+00 -1.700000E+01


MATRIX B:

-3.590000E+02
 2.810000E+02
 8.500000E+01


MATRIX X:

 1.000000E+00
-2.000000E+00
-5.000000E+00
```

Subprogram Name: Decomp

Given a nonsingular matrix A, subprogram Decomp computes the triangular decomposition of A using Gaussian Elimination. The triangular matrices L and U are calculated as well as the permutation matrix P such that LU=PA.

Subprogram Utilization:

File Name - contained in "Ludsht"

Calling Syntax - CALL Decomp (N,A(*),Lu(*),Ips(*))

Input Parameters:

N	Size of matrix A.
A(*)	Array containing nonsingular matrix A; dimensioned A(1:N, 1:N).

Output Parameters:

Lu(*)	Array storing (L-I) and U, the triangular matrices in the triangular decomposition; dimensioned Lu(1:N, 1:N)
Ips(*)	Vector containing the permuted indices, subscripted from 1 to N.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Decomp."
   N =

   The data may be corrected from the keyboard (e.g., N=1Ø, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. If the subprogram detects a row of zeros, the following error message will be printed and the program will pause:

   "ERROR IN SUBPROGRAM Decomp."
   "MATRIX WITH ZERO ROW."

   This test is made at the beginning of the subprogram before any changes are made in any of the matrices. It indicates some problem in the calling program.

3. In the Gaussian Elimination section, there are two tests performed to check that A(*) is not a machine singular matrix. If either test fails, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Decomp"
   "MATRIX IS MACHINE SINGULAR"

Methods and Formulae:

Given a nonsingular matrix A, subprogram Decomp computes the triangular decomposition using Gaussian Elimination. The triangular matrices L and U are calculated as well as the permutation matrix P such that LU=PA.

Temporarily ignoring scaling and pivoting, the central calculation, the elimination, can be expressed by

```
FOR J = K+1 to N
    A(I,J) = A(I,J) - (A(I,K)/A(K,K))*A(K,J).
NEXT J.
```

This operation is carried out by the innermost FOR-NEXT statement. The multipliers, (A(I,K)/A(K,K)), are saved in the lower triangular matrix L.

In Decomp, the element of largest absolute value in each row of the matrix is found and its reciprocal is recorded in vector Scales(*). But no actual scaling is carried out. Instead, these scale factors are used for choosing the pivot element only. This technique has two favorable consequences: exact powers of the machine base are not needed for scaling, and the scale factors do not have to be applied to the right-hand sides.

The same type of consideration is involved in pivoting. The array Ips(*) is initialized so that Ips (I) = I.

The difficulty here is more complex. Certain nonsingular matrices may be made singular as a result of the perturbations introduced by round-off error. If this is the case, a more specific algorithm may be required to solve the system of equations. See the references to Golub and to Golub and Kahan for further details.

More likely, a truly singular input matrix will be perturbed into a neighboring nonsingular matrix by the round-off since normally round-off modifies some element of the pivotal column to a non zero value.

During the elimination, the largest element in the column is chosen as the pivot element, but the rows are not actually interchanged. The corresponding elements of Ips(*) are interchanged instead. We then refer to A(Ips(I),J) instead of A(I,J). This involves no great loss of time as long as all inner loops are on the column subscript J. We gain the time that would be required to carry out the interchange.

Finally, L-I and U are stored in matrix Lu.

Subprogram Name: Solve

Given the triangular decomposition of a nonsingular matrix A, stored in matrix Lu, this subprogram will find an approximate solution to a single system of equations, Ax=b.

Subprogram Utilization:

File Name - contained in file "Ludsht"

Calling Syntax - CALL Solve (N,Lu(*),B(*),X(*),Ips(*))

Input Parameters:

N	Order of matrix Lu(*).
Lu(*)	Array containing the triangular decomposition of the nonsingular matrix A; dimensioned (1:N, 1:N).
B(*)	Vector containing the coefficients b of Ax=b; subscripted from 1 to N.
Ips(*)	Vector containing the permuted indices from subprogram Decomp; subscripted from 1 to N.

Output Parameters:

X(*)	Vector containing the solution to Ax=b; subscripted from 1 to N.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Solve"

   N =

   The data may be corrected from the keyboard (e.g., N=1Ø, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. If the subprogram detects a division by zero, the following error message is printed and the program will pause:

   "ERROR IN SUBPROGRAM Solve"

   "DIVISION BY ZERO DETECTED"

   This indicates that the matrix is machine singular. Certain nonsingular matrices may be made singular as a result of the perturbation introduced by round-off error. If this is the case, a more specific algorithm may be required to solve the system of equations. See the references to Golub and to Golub and Kahan for further details.

Methods and Formulae:

Solve uses the Lu factorization from Decomp to find an approximate solution to a single system of equations, Ax=b.

Solve consists of two steps. The first solves the lower triangular system Ly=b. The second is the back solution, i.e., the solution of the upper triangular system Ux=y. The intermediate vector y is stored in x, and the right-hand side b is not altered.

Subprogram Name: Posdef

Given a symmetric, positive definite matrix, A, stored in symmetric storage mode, Posdef will solve the system of equations AX=B using Cholesky's Method. The value of A is overwritten.

Subprogram Utilization:

File Name - "Posdef", cartridge 1

Calling Syntax - CALL Posdef (S(*),B(*),R,C)

Input Parameters:

*S(*)	Vector containing symmetric, positive-definite matrix, stored in symmetric storage mode, subscripted from 1 to (R+1)R/2.
*B(*)	Array containing coefficient matrix B; dimensioned B(1:R, 1:C).
R	Number of rows of B(*).
C	Number of columns of B(*).

Subprograms Required:

Sub Choles	Performs Cholesky decomposition on S(*).
Sub Solcho	Solves Cholesky system in symmetric storage mode.

Output Parameters:

*S(*)	Array containing Cholesky decomposition, $S=GG^T$ where G is a lower triangular matrix subscripted from 1 to (R+1)R/2.
*B(*)	Array containing solution matrix X to AX=B; dimensioned B(1:R,1:C)

Local Variables:

Baddta	Used in BAD DATA CHECK. Baddta = 1 implies bad data Baddta = Ø implies reasonable data

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Posdef"

   R = , C =

   The data may be corrected from the keyboard (e.g., R=5, EXECUTE.) When CONT is pressed, the program will resume execution at the next line.

*S(*) and B(*) are both input and output parameters.

2. Because of the summetry of the positive definite matrix, it is necessary to store only $\frac{1}{2}$ (n+1), or slightly over half, of its elements, resulting in an important saving of storage for large matrices. Since some additional multiplications and additions are required to manipulate the vector, there is an increase in execution time.

3. Given a summetric matrix A, the symmetric storage mode vector is formed as follows:

A(1,1) A(1,2) A(1;3)- - - A(1,N)

A(2,1) A(2,2) A(2,3)- - - A(2,N)

A(3,1) A(3,2) A(3,3)- - - A(3,N)

A(N,1) A(N,2) A(N,3)- - - A(N,N)

i.e., A(1,1), A(2,1), A(2,2), A(3,1), A(3,2), A(3,3),..., A(N,1), A(N,N). Subprogram Storag may be used to convert a full storage matrix to symmetric storage mode and vice versa.

Methods and Formulae:

Any positive definite matrix A has a unique decomposition in the form $A=GG^T$, where G is a lower triangular matrix with positive diagonal elements. The algorithm is:

FOR J=1 to N

$$G(J,J) = SQR\left(A(J,J) - \sum_{K=1}^{J} (G(J,K))^2\right)$$

FOR I=J+1 TO N

$$G(I,J) = \left(A(I,J) - \sum_{K=1}^{J-1} G(I,K)*G(J,K)\right)/G(J,J)$$

NEXT I

NEXT J

This algorithm is Cholesky's method or the square-root method for factoring a positive definite matrix. It is stored in subprogram Choles.

Given a linear system AX=B, where A is a positive definite matrix, subprogram Posdef first calls Choles to factor A into GG^T. Then subprogram Solcho is called to forward eliminate and back substitute to find the solution matrix X.

Driver Utilization:

File Name - "POSDEF"

The driver "POSDEF" sets up the necessary input parameters for the subprogram Posdef as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "POSDEF"
   a. Type: GET "POSDEF"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Posdef ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Posdef contained in file "Posdef" will be linked on after the driver.

5. When "DIMENSIONS OF B(*) [ROW, COLUMN]?" is displayed:
   a. Enter the dimensions of the coefficient matrix B,i.e., matrix B of AX=B.
   b. Press: CONT.

6. At this point, subprogram Pos 1 is called to dynamically set up the arrays S(*) (the symmetric storage mode vector) and B(*).

7. When "VECTOR S:" is printed and "S(I)?" (FOR I=1 TO # OF ELEMENTS IN SYMMETRIC STORAGE VECTOR) is displayed:
   a. Enter the Ith element of the symmetric storage mode vector.
   b. Press: CONT.
   c. If more elements are to be entered, go to 7.

8. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the vector S.
   b. Press: CONT.
   c. Go to 9.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 11.

9. When "VECTOR ELEMENT?" is displayed:
   a. Enter the coordinate of the element to be changed.
   b. Press: CONT.

10. When "S(I)?" (where I is the coordinate chosen in 9) is displayed:
   a. Enter the new Ith element
   b. Press: CONT.
   c. Go to 8.

11. When "MATRIX B:" is printed and "B(I,J)?" (FOR I=1 TO # OF ROWS; J=1 TO # OF COLUMNS) is displayed:
    a. Enter the I,Jth element of matrix B.
    b. Press: CONT.
    c. If more elements are to be entered, go to 11.

12. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are desired in matrix B.
    b. Press: CONT.
    c. Go to 13.

or

    a. Enter N if no changes are desired.
    b. Press: CONT.
    c. Go to 15.

13. When "COORDINATES OF B(*) [ROW, COLUMN]?" is displayed:
    a. Enter the row and column numbers of the element to be changed, e.g., if element B(4,3) is to be changed, enter 4,3.
    b. Press: CONT.

14. When "B(I,J)?" (where I and J represent the row and column of the element to be changed) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 12.

15. The program will then print the solution matrix X, where columns of X are the solutions to corresponding columns of B.

EXAMPLE

1. Given the positive definite matrix $A = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 10 & 5 \\ -2 & 5 & 6 \end{pmatrix}$

and $B = \begin{pmatrix} -6 \\ 33, \\ 30 \end{pmatrix}$ solve AX=B.

```
DIMENSIONS OF B(*)= 3 X 1

VECTOR S:

 4.000000E+00  2.000000E+00  1.000000E+01 -2.000000E+00  5.000000E+00
 6.000000E+00



MATRIX B:

-6.000000E+00

 3.300000E+01

 3.000000E+01



MATRIX X:

-1.000000E+00

 2.000000E+00

 3.000000E+00
```

2. Given the positive definite matrix $A = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix}$ and

$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$,

Solve AX=B.

```
DIMENSIONS OF B(*)= 3 X 3

VECTOR S:

 1.000000E+00  5.000000E-01  3.333333E-01  3.333333E-01  2.500000E-01
 2.000000E-01



MATRIX B:

 1.000000E+00  0.000000E+00  0.000000E+00

 0.000000E+00  1.000000E+00  0.000000E+00

 0.000000E+00  0.000000E+00  1.000000E+00



MATRIX X:

 9.000000E+00 -3.600000E+01  3.000000E+01

-3.600000E+01  1.920000E+02 -1.800000E+02

 3.000000E+01 -1.800000E+02  1.800000E+02
```

Subprogram Name: Choles

This subprogram performs Cholesky decomposition on a symmetric, positive definite matrix stored in symmetric storage mode.

Subprogram Utilization:

File Name - contained in file "Posdef"

Calling Syntax - CALL Choles (G(*),R)

Input Parameters:

*G(*)	Vector containing the symmetric, positive definite matrix stored in symmetric storage mode; subscripted from 1 to (R+1)R/2.
R	Order of positive definite matrix.

Output Parameters:

*G(*)	Vector containing the lower triangular matrix G where GG^T is the decomposition of the positive definite matrix.

(G(*) is both an input and output parameter.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Choles"

   R =

   The data may be corrected from the keyboard (e.g., R=7, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. If the subprogram detects a nonpositive diagonal element, the following error message will be printed and the program

   "ERROR IN SUBPROGRAM Choles"

   "MATRIX IS NOT MACHINE POSITIVE DEFINITE"

   This is a fatal error. There is no recovery. The PAUSE was inserted to enable the user to check values of the variables in the subprogram.

3. Given a symmetric matrix A, the symmetric storage mode vector is formed as follows:
   A(1,1), A(2,1), A(2,2), A(3,1), A(3,2), A(3,3),---, A(N,1), ---, A(N,N)

Methods and Formulae:

Any positive definite matrix A has a unique decomposition in the form $A=GG^T$, where G is a lower triangular matrix with positive diagonal elements. The algorithm is:

```
FOR J=1 TO N
                                    J
    G(J,J) = SQR(A(J,J) - Σ  G(J,K)^2)
                                   K=1

    FOR I=J+1 TO N
                                  J-1
        G(I,J)=(A(I,J) -  Σ  G(I,K)*G(J,K))/G(J,J)
                                  K=1

    NEXT I
NEXT J.
```

This algorithm is Cholesky's method or the square-root method for factoring a positive definite matrix.

Subprogram Name: Solcho

This subprogram solves a Cholesky system in symmetric storage mode, i.e., given a symmetric, positive definite matrix, $A=GG^T$, stored in symmetric storage mode, solve AX=B.

Subprogram Utilization:

File Name - contained in file "Posdef"

Calling Syntax - CALL Solcho (G(*), B(*), R,C)

Input Parameters:

G(*)	Vector containing the lower triangular matrix G stored in symmetric storage mode, where GG^T is the Cholesky decomposition of the matrix; subscripted from 1 to (R+1) R/2.
*B(*)	Array containing the coefficient matrix B; dimensioned B(1:R,1:C).
R	Number of rows in B(*).
C	Number of columns in B(*).

Output Parameters:

*B(*)	Array containing the solution matrix X of AX=B.

Special Considerations and Programming Hints:

1. Upon entry into the program, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Solcho"

   R = , C =

   The data may be corrected from the keyboard (e.g., R=7, EXECUTE.) When CONT is pressed, the program will resume execution at the next line.

2. The data stored in the coefficient matrix B is destroyed and replaced by the solution matrix X.

3. Given a symmetric matrix A, the symmetric storage mode vector is formed as follows:

   A(1,1), A(2,1), A(2,2), A(3,1), A(3,2), A(3,3),---, A(N,1),---, A(N,N).

Methods and Formulae:

Solcho uses the GG^T factorization from subprogram Choles to find an approximate solution to the system of equations, AX=B. Both A, stored in symmetric storage mode, and B, the coefficient matrix, are destroyed.

Solcho consists of two steps. The first solves the lower triangular system GY=B. The second is the back substitution, i.e., the solution of the upper triangular system $G^TX=B$.

HEWLETT hp PACKARD

MATRIX TRANSPOSITION IN PLACE

Subprogram Name: Xpose

This subprogram transposes a rectangular matrix in place. Transposition of the R by C matrix A amounts to replacing the element at vector position I with the element at position (R*I) (MOD R*C-1). Each subcycle of this permutation is completed in order.

Subprogram Utilization:

File Name - "Xpose"

Calling Syntax - CALL Xpose (A(*), R,C, Nwork)

Input Parameters:

*A(*)	Array containing the rectangular matrix to be transposed; A(*) must be dimensioned A(1:R,1:C) in the calling program.
R	Row dimension of matrix A.
C	Column demension of matrix A.
Nwork	Dimension of a work vector, Moved(*); the optimal size seems to be Nwork = (R+C)/2; if storage constraints limit this, Nwork may be set equal to 1; in any case, Nwork $\geq$ 1.

Subprograms Required:

Sub Factor	Factors a number into its prime powers.

*A(*) is both an input and output parameter.

Output Parameters:

*A(*)	Array containing the transposed matrix; a redimension statement, REDIM A(C,R) must be used in the calling program in order for the elements of A(*) to be actually in their transposed position.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Xpose"

   R = , C = , Nwork =

   The data may be corrected from the keyboard (e.g., R=7, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. Array Moved(*) is a work vector dimensioned dynamically in Xpose by Nwork. References 1 and 3, upon which this subprogram was based, suggest that Nwork=INT((R+C)/2) where R is the row size of A(*) and C is the column size. In any case, Nwork must be at least 1.

3. Arrays Fact(*), Power(*), Nexp(*), Iexp(*) and Moved(*) have all been dimensioned as integer arrays. In the test cases tried, this only caused a slight increase in the execution time of the algorithm at a savings of approximately 128 + (R+C)/2 bytes.

*A(*) is both an input and output parameter.

4. If the calling program has DIM A(R,C), then after Xpose is called there must be a REDIM A(C,R) before any other operations are performed using matrix A.

Methods and Formulae:

This subprogram finds the transposition of a rectangular matrix in place. The transposition of the RxC matrix A amounts to replacing the element at vector position I with the element at position (R*I)(MOD R*C-1). Each sub cycle of this permutation is completed in order. The program breaks into two parts. First determine starting points for the subcycles and then move the data.

For each divisor d of m (m = R*C-1), the subcycles beginning with d and with m-d are done. If the number of data moved is still less than the Euler totient function, Φ(mld), further subcycle starting points of the form sd are tried, for s=2,3,---. The most general test is that sd is acceptable if no element in its subcycle is less than sd or greater than m-sd. Since this test requires much time-consuming computation, it is much faster to look for sd in a table where marks are made to indicate that an element has been moved. Here, a special table of length Nwork is used. Nwork=INT((R+C)/2) was found to be sufficient for most cases. The user has been given the option to make Nwork as small as 1. The trade-off here is storage savings at the price of execution time.

The inner loop of the subprogram computes element by element transposition, moves data, marks in the table, if available, and checks for loop closure.

References:

1. Brenner, N. Algorithm 467: Matrix Transposition in Place. Collected Algorithms From ACM.

2. Knuth, D. The Art of Computer Programming, Vol. 1 Addison-Wesley, Reading, Mass., 1967, p. 180, prob. 12, and p 517, solution to prob. 12.

3. Laflin, S., and Brebner, M.A. Algorithm 380: In-situ Transposition of a Rectangular Matrix. Comm. ACM 13 (May 1970), 324-326.

Driver Utilization:

File Name - "XPOSE", cartridge 1

The driver "XPOSE" sets up the necessary input parameters for the subprogram Xpose as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer. Note that Nwork is set up as the average of the number of rows and number of columns.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "XPOSE"
   a. Type: GET "XPOSE"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Xpose ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Xpose contained in file "Xpose" is linked on after the driver.

5. When "DIMENSIONS OF A(*), [ROW, COLUMN]?" is displayed:
   a. Enter the dimensions of matrix A, e.g., if A(*) has dimensions 3 by 5, then enter 3,5.
   b. Press: CONT.

6. At this point, subprogram Xpo 1 is called to dynamically set up the array A(*).

7. When "MATRIX A:" is printed and "A(I,J)?" (FOR I=1 TO # OF ROWS; J=1 TO # OF COLUMNS) is displayed:
   a. Enter the I,Jth element of matrix A.
   b. Press: CONT.
   c. If more elements are to be entered, go to 7.

8. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the data.
   b. Press: CONT.
   c. Go to 9.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 11.

9. When "COORDINATES OF A(*) [ROW, COLUMN]?" is displayed:
   a. Enter the row and column numbers of the element to be changed, e.g., if element A(4,3) is to be changed, enter 4,3.
   b. Press: CONT.

10. When "A(I,J)?" (where I and J represent the row and column to be changed) is displayed:
   a. Enter the new value of the element.
   b. Press: CONT.
   c. Go to 8.

11. The transpose of A(*) will then be printed. A REDIMensioning statement occurs before this printing, so that matrix A will be in the correct order.

EXAMPLE

```
DIMENSIONS OF A(*)= 6 X 4

MATRIX A:

 1.0000000E+00  1.0000000E+00  1.0000000E+00  1.0000000E+00

 2.0000000E+00  2.0000000E+00  2.0000000E+00  2.0000000E+00

 3.0000000E+00  3.0000000E+00  3.0000000E+00  3.0000000E+00

 4.0000000E+00  4.0000000E+00  4.0000000E+00  4.0000000E+00

 5.0000000E+00  5.0000000E+00  5.0000000E+00  5.0000000E+00

 6.0000000E+00  6.0000000E+00  6.0000000E+00  6.0000000E+00


MATRIX A:

 1.0000000E+00  2.0000000E+00  3.0000000E+00  4.0000000E+00  5.0000000E+00
 6.0000000E+00

 1.0000000E+00  2.0000000E+00  3.0000000E+00  4.0000000E+00  5.0000000E+00
 6.0000000E+00

 1.0000000E+00  2.0000000E+00  3.0000000E+00  4.0000000E+00  5.0000000E+00
 6.0000000E+00

 1.0000000E+00  2.0000000E+00  3.0000000E+00  4.0000000E+00  5.0000000E+00
 6.0000000E+00
```

Subprogram Name: Factor

This subprogram is used in subprogram Xpose, transposing a matrix in place. Factor factors an integer N into its prime factors.

Subprogram Utilization:

File Name - contained in file "Xpose"

Calling Syntax - CALL factor (N,Fact(*),Power(*),Nexp(*),Npower)

Input Parameters:

N	Number to be factored.

Output Parameters:

Fact(*)	Vector containing prime factors of N.
Power (*)	Vector containing factors of N.
Nexp(*)	Vector containing the number of powers of factors of N.
Npower	Number of distinct prime factors of N.

For example, if N = 1960 = 2^3*5*7^2, Npower = 3, Fact(*) = (3,5,7), Power(*) = (8,5,49) and Nexp(*) = (3,1,2).

HEWLETT hp PACKARD

LINEAR ALGEBRAIC SYSTEMS: INVERSE OF POSITIVE DEFINITE MATRIX

Subprogram Name: Pinver

Pinver finds the inverse of a positive definite matrix stored in symmetric storage mode in vector S(*). The inverse overwrites the values contained in S(*).

This subprogram uses Cholesky's method to decompose the positive definite matrix.

The advantage of using this algorithm to find the inverse of a positive definite matrix is twofold:

1. Significant memory is saved;
2. Since pivoting is not required in the Gaussian elemination, there is less roundoff effect on the computed inverse.

Subprogram Utilization:

File Name - "Pinver"

Calling Syntax - CALL Pinver (S(*),R)

Input Parameters:

*S(*)	Vector containing positive definite matrix stored in symmetric storage mode subscripted from 1 to (R+1)R/2.
R	Order of positive definite matrix, i.e., number of rows or number of columns.

*S(*) is both an input and output parameter.

Subprograms Required:

Choles	Performs Cholesky decomposition on S(*).
Invers	Finds the inverse of the lower triangular matrix S(*) stored in symmetric storage mode. The inverse, also a lower triangular matrix, is again stored in S(*).
Triang	Multiplies $S^T S$ where S is a lower triangular matrix stored in symmetric storage mode. The result is restored in S(*).

Output Parameters:

*S(*)	Vector containing the inverse of the initial positive definite matrix stored in symmetric storage mode.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Pinver"

   R =

   The data may be corrected from the keyboard (e.g., R=4, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

*S(*) is both an input and output parameter.

2. This subprogram takes specific advantage of the nature of the matrix. As a result, a minimum number of operations are used. Hence, the user may expect better accuracy than the more general matrix routines. In testing this subprogram and comparing it with the machine inversion routine, a resulting extra two digits of accuracy was not unusual for ill-conditioned matrices.

3. This subprogram also attempts to minimize the storage requirements. The symmetric storage vector S(*), containing the positive definite matrix, is repeatedly overwritten. This takes a little over half the number of bytes as the machine inversion routine.

4. Given a symmetric matrix A, the symmetric storage mode vector is formed as follows:

A(1,1)	A(1,2)	A(1,3)	- - -	A(1,N)
A(2,1)	A(2,2)	A(2,3)	- - -	A(2,N)
A(3,1)	A(3,2)	A(3,3)	- - -	A(3,N)
⋮				
A(N,1)	A(N,2)	A(N,3)	- - -	A(N,N)

i.e., A(1,1), A(2,1), A(2,2), A(3,1), A(3,2), A(3,3), --- , A(A,1). ---, A(N,N).
Subprogram Storage may be used to convert a full storage matrix to symmetric storage mode and vice versa.

Methods and Formulae:

Any positive definite matrix A, has a unique decomposition in the form $A=SS^T$, where S is a lower triangular matrix with positive diagonal elements. This algorithm is Cholesky's method or the square-root method for factoring a positive definite matrix. Subprogram Choles is based on this algorithm.

Subprogram Invers is then called to find the inverse of the lower triangular matrix S(*) stored in symmetric storage mode. (The symmetric storage mode is used even though the matrix is lower triangular and not symmetric.) The inverse, also a lower triangular matrix, is again stored in S(*).

Finally subprogram Triang multiplies S^TS to get the inverse of the original matrix:

$$A + SS^T \text{ implies } A^{-1} = (SS^T)^{-1} = (S^T)^{-1}S^{-1} = (S^{-1})^T S^{-1}$$

Driver Utilization:

File Name - "PINVER", cartridge 1

The driver "PINVER" sets up the necessary input parameters for the subprogram Pinver as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "PINVER"
   a. Type: GET "PINVER"
   b. Press: EXECUTE

3. Press: RUN.

4. When "HAS Pinver ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Pinver contained in file "Pinver" is linked on after the driver.

5. When "ORDER OF A(*), [# OF ROWS OR # OF COLUMNS]?" is displayed:
   a. Enter the order of the matrix.
   b. Press: CONT.

6. At this point, subprogram Pin 1 is called to dynamically set up the vector S(*) which is to contain the positive definite matrix in symmetric storage mode.

7. When "VECTOR S: [IN SYMMETRIC STORAGE MODE]?" is printed and "S(I)?" (FOR I=1 TO (N+1)N/2 where N is the order of matrix A) is displayed:
   a. Enter the Ith element of the vector.
   b. Press: CONT.
   c. If more elements are to be entered, go to 7.

   NOTE: Symmetric Storage Mode is explained in the Special Considerations section of the subprogram algorithm.

8. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the vector.
   b. Press: CONT.
   c. Go to 9.

   or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 11.

9. When "VECTOR ELEMENT?" is displayed:
   a. Enter the coordinate of the element to be changed.
   b. Press: CONT.

10. When "S(I)?" (where I is the coordinate chosen in 9) is displayed:
   a. Enter the new Ith element.
   b. Press: CONT.
   c. Go to 8.

11. The program will then print the inverse stored in symmetric storage mode vector S(*).

EXAMPLE

1. Find the inverse of the symmetric positive definite matrix.

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix}$$

```
ORDER OF A(*)= 3


VECTOR S:  [IN SYMMETRIC STORAGE MODE]

 1.000000E+00  5.000000E-01  3.333333E-01  3.333333E-01  2.500000E-01
 2.000000E-01



INVERSE S:   [IN SYMMETRIC STORAGE MODE]

 9.000000E+00 -3.600000E+01  1.920000E+02  3.000000E+01 -1.800000E+02
 1.800000E+02
```

2. Find the inverse of the symmetric positive definite matrix.

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{pmatrix}$$

```
ORDER OF A(*)= 5


VECTOR S:  [IN SYMMETRIC STORAGE MODE]

 1.000000E+00  5.000000E-01  3.333333E-01  3.333333E-01  2.500000E-01
 2.000000E-01  2.500000E-01  2.000000E-01  1.666667E-01  1.428571E-01
 2.000000E-01  1.666667E-01  1.428571E-01  1.250000E-01  1.111111E-01



INVERSE S:   [IN SYMMETRIC STORAGE MODE]

 2.500000E+01 -3.000000E+02  4.800000E+03  1.050000E+03 -1.890000E+04
 7.938000E+04 -1.400000E+03  2.688000E+04 -1.176000E+05  1.792000E+05
 6.300000E+02 -1.260000E+04  5.670000E+04 -8.820000E+04  4.410000E+04
```

Subprogram Name: Triang

Given a lower triangular matrix stored in symmetric storage mode vector S(*), this subprogram will multiply S^TS. The result, a symmetric matrix, will again be restored in vector S(*).

Subprogram Utilization:

File Name - contained in file "Pinver"

Calling Syntax - CALL Triang (S(*),R)

Input Parameters:

*S(*)	Vector containing lower triangular matrix stored in symmetric storage mode.
R	Order of full storage mode matrix.

Output Parameters:

*S(*)	Product of S^TS stored in symmetric storage mode.

*S(*) is both an input and output parameter.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Triang"

   R =

   The data may be corrected from the keyboard (e.g., R=5, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

Methods and Formulae:

Given a lower triangular matrix stored in symmetric storage mode vector S(*), Triang multiplies $S^T S$. The result is restored in S(*).

```
FOR I=1 TO R
    FOR J=I TO R
    S(J*(J-1)/2+I) = Σ (L=J to R) S(L*(L-1)/2+I)*S(L*(L-1)/2+J)
    NEXT J
NEXT I
```

HEWLETT hp PACKARD

LINEAR ALGEBRAIC SYSTEMS: INVERSE OF LOWER TRIANGULAR MATRIX

Subprogram name: INVERS

Invers finds the inverse of a lower triangular matrix stored in symmetric storage mode vector S(*). The inverse, also a lower triangular matrix, is restored in vector S(*).

Subprogram Utilization:

File Name - contained in file "Pinver"

Calling Syntax - CALL Invers (S(*),R)

Input Paramters:

*S(*)	Vector containing a lower triangular matrix stored in symmetric storage mode; subscripted from 1 to (R+1)R/2.
R	Order of full storage mode matrix.

Output Parameters:

*S(*)	Vector containing the inverse of the input matrix. The inverse is also a lower triangular matrix and is stored in symmetric storage mode.

*S(*) is both an input and output parameter.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Invers"

   R =

   The data may be corrected from the keyboard (e.g., R=5, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. If the matrix is machine singular, the following error message will be rpinted and the program will pause:

   "ERROR IN SUBPROGRAM Invers"

   "MATRIX IS MACHINE SINGULAR"

   The pause allows the user to inspect the values of the variables in the subprogram.

Methods and Formulae:

Invers finds the invers of a lower triangular matrix stored in symmetric storage mode vector S(*). The inverse, also a lower triangular matrix, is restored in S(*). (The lower triangular matrix, of course, is not symmetric. The symmetric storage mode is used solely to save space.)

First, the diagonal elements of the inverse are found:

```
FOR I=1 TO R
   S(I*(I-1)/2) = 1/S(I*(I-1)/2)
NEXT I.
```

Then, the off-diagonal elements are calculated:

```
FOR I=2 TO R
   FOR J=1 TO I-1
      S(I*(I-1)/2+J) = -S(I*(I-1)/2+I)*
```

$$\sum_{L=J}^{I-1} S(I*(I-1)/2+L)*S(L*(L-1)/2+J)$$

```
   NEXT J
NEXT I
```

Subprogram Name: Storag

Given a symmetric matrix, A, this subprogram will store A in a vector, S, in symmetric storage mode. If A is an n x n matrix, S is a vector with n(n+1)/2 elements. Likewise, given S, this subprogram will convert to the symmetric matrix A.

Subprogram Utilization:

File Name - "Storag"

Calling Syntax - CALL Storag (A(*),S(*),N,Flg).

Input Parameters:

A(*)		Array containing symmetric matrix in full storage mode; dimensioned A(1:N,1:N).
	or	
S(*)		Array containing vector in symmetric storage mode; dimensioned S(1:(N+1)N/2).
N		Dimension of symmetric matrix in full storage mode.
Flg		Flg = 1 convert from symmetric to full storage mode.
		Flg = -1 convert from full to symmetric storage mode.

Output Parameters:

A(*)	Array containing symmetric matrix in full storage mode.
or	
S(*)	Array containing vector in symmetric storage mode.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Storag"

   N = , Flg =

   The data may be corrected from the keyboard (e.g., N=6, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. This subprogram saves a significant amount of storage when dealing with large arrays. An n x n matrix A requires storage for n^2 elements while vector S contains only n(n+1)/2 element. A large matrix could be stored on a mass storage device in symmetric storage mode and then accessed in particular programs. The cost for the savings in storage is time - additional operations are required to convert a particular element in the vector S to an element in matrix A.

Methods and Formulae:

Given the symmetric matrix A and the symmetric storage mode vector S, the A(I,J) element of A is stored in S(T) where

T = I*(I-1)/2+J if I>=J

T = J*(J-1)/2+I if I<J

For example,

Let $A=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 8 \\ 3 & 6 & 7 & 9 \\ 4 & 8 & 9 & \emptyset \end{pmatrix}$ Then

S(1) = 1 S(6) = 7
S(2) = 2 S(7) = 4
S(3) = 5 S(8) = 8
S(4) = 3 S(9) = 9
S(5) = 6 S(1Ø) = Ø.

Driver Utilization:

File Name - "STORAG", cartridge 1

The driver "STORAG" sets up the necessary input parameters for the subprogram Storag as well as prints the resulting output.

The user has the option of going from full storage to symmetric storage mode or vice versa.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "STORAG"
   a. Type: GET "STORAG"
   b. Press: EXECUTE

3. Press: RUN.

4. When "HAS Storag ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

a. Enter N if the subprogram has not yet been linked on.
b. Press: CONT.
c. At this point, subprogram Storag contained in file "Storag" is linked on after the driver.

5. When "ORDER OF MATRIX IN FULL STORAGE MODE?" is displayed:
   a. Enter the order of the symmetric matrix in full storage mode, e.g., if A is a 30x30 matrix, the user would enter 30.
   b. Press: CONT.

6. At this point, the dimensions of the required arrays are computed. Storl is then called to dynamically set up the arrays.

7. When "SYMMETRIC TO FULL STORAGE MODE (Y/N)?" is displayed:
   a. Enter Y if the conversion is from symmetric to full storage mode.
   b. Press: CONT.
   c. Go to 8.

or

   a. Enter N if the conversion is from full to symmetric storage mode.
   b. Press: CONT.
   c. Go to 12.

8. When "SYMMETRIC STORAGE MODE:" is printed and "S(I)?" (FOR I=1 TO (N+1)N/2 WHERE N IS THE ORDER OF THE FULL STORAGE MATRIX) is displayed:
   a. Enter the Ith element of the symmetric storage mode vector.
   b. Press: CONT.
   c. If more elements are to be entered, go to 8.

9. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the vector.
   b. Press: CONT.
   c. Go to 1Ø.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 16.

10. When "VECTOR ELEMENT?" is displayed:
   a. Enter the coordinate of the element to be changed.
   b. Press: CONT.

11. When "S(I)?" (where I is the coordinate chosen in 1Ø) is displayed:
   a. Enter the new Ith element.
   b. Press: CONT.
   c. Go to 9.

12. When "FULL STORAGE MODE:" is printed and "A(I,J)?" (FOR I=1 TO # OF ROWS IN A(*), FOR J=1 TO # OF COLUMNS IN A(*)) is displayed:
   a. Enter the Ith, Jth element of matrix A.
   b. Press: CONT.
   c. If more elements are to be entered, go to 12.

13. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in matrix A.
   b. Press: CONT.
   c. Go to 14.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 16.

14. When "COORDINATES OF A(*), [ROW,COLUMN]?" is displayed:
    a. Enter the row and column numbers of the element to be changed, e.g., if element A(5,3) is to be changed, enter 5,3.
    b. Press: CONT.

15. When "A(I,J)?" (WHERE I AND J REPRESENT THE ROW AND COLUMN OF THE ELEMENT TO BE CHANGED) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 13.

16. The program will then print the inverted matrix.

EXAMPLES

```
EXAMPLE 1

ORDER OF MATRIX IN FULL STORAGE MODE= 4

FULL STORAGE MATRIX:

 1.000000E+00  5.000000E-01  3.333333E-01  2.500000E-01

 5.000000E-01  3.333333E-01  2.500000E-01  2.000000E-01

 3.333333E-01  2.500000E-01  2.000000E-01  1.666667E-01

 2.500000E-01  2.000000E-01  1.666667E-01  1.428571E-01


MATRIX IN SYMMETRIC STORAGE MODE:

 1.000000E+00  5.000000E-01  3.333333E-01  3.333333E-01  2.500000E-01
 2.000000E-01  2.500000E-01  2.000000E-01  1.666667E-01  1.428571E-01


EXAMPLE 2

ORDER OF MATRIX IN FULL STORAGE MODE= 4

SYMMETRIC STORAGE MATRIX:

 1.000000E+00  5.000000E-01  3.333333E-01  3.333333E-01  2.500000E-01
 2.000000E-01  2.500000E-01  2.000000E-01  1.666667E-01  1.428571E-01


MATRIX IN FULL STORAGE MODE:

 1.000000E+00  5.000000E-01  3.333333E-01  2.500000E-01

 5.000000E-01  3.333333E-01  2.500000E-01  2.000000E-01

 3.333333E-01  2.500000E-01  2.000000E-01  1.666667E-01

 2.500000E-01  2.000000E-01  1.666667E-01  1.428571E-01
```

HEWLETT hp PACKARD

EIGENANALYSIS: EIGENVALUES AND EIGENVECTORS OF A REAL GENERAL MATRIX

Subprogram Name: Eigen

This subprogram finds all the eigenvalues and eigenvectors of a real general matrix. The eigenvalues are computed by the QR double-step method and the eigenvectors by inverse iteration.

Subprogram Utilization:

File Name - "Eigen"

Calling Syntax - CALL Eigen (N,A(*), Evr(*), Evi(*), Vecr(*), Veci(*), Indic(*))

Input Parameters:

N	Order of matrix A.
A(*)	Array containing matrix for which eigenvalues and eigenvectors are to be found; dimensioned A(1:N, 1:N).

Subprograms Required:

Scale	This subprogram scales matrix A so that the quotient of the absolute sum of the off-diagonal elements of column I and the absolute sum of the off-diagonal elements of row I lies within certain bounds.

Hesqr	This subprogram finds all the eigenvalues of a a real general matrix.
Realve	This subprogram finds the real eigenvector of the real upper-Hessenberg matrix in the array A, corresponding to the real eigenvalue stored in Evr(Ivec). The inverse iteration method is used.
Compve	This subprogram finds the complex eigenvector of the real upper-Hessenberg matrix of order n corresponding to the complex eigenvalue with the real part in Evr(Ivec) and the corresponding imaginary part in Evi(Ivec). The inverse iteration method is used in a modified manner to avoid the use of complex arithmetic.

Output Parameters:

A(*)	The original matrix A is destroyed.
Evr(*)	Vector containing the real parts of the N computed eigenvalues dimensioned Evr(1:N).
Evi(*)	Vector containing the imaginary parts of the N computed eigenvalues dimensioned Evi(1:N).
Vecr(*)	Array containing the real components of the normalized eigenvector I (for I=1 to N), corresponding to the eigenvalue stored in Evr(I) and Evi(I); the Ith eigenvector is stored in column I; dimensioned Vecr(1:N,1:N).
Veci(*)	Array containing the imaginary components of the normalized eigenvector I (for I=1 to N), corresponding to the eigenvalue stored in Evr(I) and Evi(I); the Ith eigenvector is stored in column I; dimensioned Veci(1:N,1:N).
Indic(*)	Array indicating the success of the subprogram as follows:

Value of Indic(I)	Eigenvalue I	Eigenvector I
∅	not found	not found
1	found	not found
2	found	found;

dimensioned Indic (1:N).

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Eigen"

   N =

   The data may be corrected from the keyboard (e.g., N=7, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. The Fortran program from which this subprogram has been adapted (Ref. 1), has been extensively tested (Ref. 2). Here is a quote from their conclusions:

   > "Conclusions. The algorithm is capable of successfully computing eigenvalues and eigenvectors of real general matrices even under conditions considered unstable. It has the advantage of being computationally fast, and has the capability of yielding results with as much precision as the hardware will permit. The algorithm does not break down when presented with a matrix which is not diagonalizable; that is, a set of eigenvectors satisfying the eigenequation is computed regardless of the existence of linearly independent eigenvectors. However, when a matrix is diagonalizable and degenerate, the algorithm does not yield well separated eigenvectors corresponding to non-distinct eigenvalues. Another apparent

disadvantage is the possible indication of completely successful computation (INDIC), even in clearly ill-conditioned situations where computational difficulties are inevitable. This latter property, however, is a common fault of other algorithms as well."

Methods and Formulae:

This subprogram finds all the eigenvalues and the eigenvectors of a real general matrix of order N.

First, in the subprogram Scale the matrix is scaled so that the corresponding rows and columns are approximately balanced and then the matrix is normalized so that the value of the Euclidian norm of the matrix is equal to one.

The eigenvalues are computed by the QR double-step method in the subprogram Hesqr. The eigenvectors are computed by inverse iteration in the subprogram Realve, for the real eigenvalues, or in the subprogram Compve, for the complex eigenvalues.

The elements of the matrix are stored in the two dimensioned array A. The original matrix is destroyed by the subprogram.

Upon output from the subprogram, the real parts of the N computed eigenvalues will be found in the first N places of the array Evr and the imaginary parts in the first N places of the array Evi. The real components of the normalized eigenvector I (for I=1,2,...,N) corresponding to the eigenvalue stored in Evr(I) and Evi(I) will be found in the first N places of the column I of the two-dimensional array Vecr and the imaginary components in the first N places of the column I of the two-dimensional array Veci.

The real eigenvector is normalized so that the sum of the squares of the components is equal to one. The complex eigenvector is normalized so that the component with the largest value in modulus has its real part equal to one and the imaginary part equal to zero.

The array Indic indicates the success of the subprogram Eigen as follows:

Value of Indic(I)	Eigenvalue I	Eigenvector I
Ø	not found	not found
1	found	not found
2	found	found

References:

1. Grad, J., and Brebner, M.A. Algorithm 343, Eigenvalues and Eigenvectors of a real general matrix. Comm ACM, 11 (Dec. 1968), pp. 820-826.

2. Knoble, H.D. Certification of Algorithm 343. Eigenvalues and Eigenvectors of a real general matrix. Comm. ACM, 13 (Feb. 1970), pp. 122-124.

3. Wilkinson, J.H. The Algebraic Eigenvalue Problem. (Oxford: Clarendon Press, 1965), pp. 86-93.

Driver Utilization:

File Name - "EIGEN", cartridge 1

The driver "EIGEN" sets up the necessary input parameters for the subprogram Eigen as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "EIGEN"
   a. Type: GET "EIGEN"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Eigen ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Eigen contained in file "Eigen" is linked on after the driver.

5. When "ORDER OF MATRIX A(*), [# OF ROWS OR # OF COLUMNS]?" is displayed:
   a. Enter the order of the matrix, e.g., if A(*) is a 5 by 5 matrix, Enter 5.
   b. Press: CONT.

6. At this point, subprogram Eigl is called to dynamically set up the arrays A(*), Vecr(*), Veci(*), Evr(*), Evi(*) and Indic(*).

7. When "MATRIX A:" is printed and "A(I,J)?" (for I=1 to # of rows in A(*), for J=1 to # of columns in A(*)) is displayed:
   a. Enter the Ith, Jth element of matrix A.
   b. Press: CONT.
   c. If more elements are to be entered, go to 7.

8. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the matrix.
   b. Press: CONT.
   c. Go to 9.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 11.

9. When "COORDINATES OF A(*), [ROW, COLUMN]?" is displayed:
   a. Enter the row and the column coordinate of the element to be changed, e.g., if A(4,3) is to be changed, enter 4,3.
   b. Press: CONT.

10. When "A(I,J)?" (where I and J are the coordinates of the element to be changed) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 8.

11. The real and imaginary parts of the eigenvalues are then printed as well as the corresponding real and imaginary parts of the eigenvectors in the following manner:
    a. Vector Evr contains the real parts of the N computed eigenvalues.
    b. Vector Evi contains the corresponding imaginary parts of the N computed eigenvalues.
    c. Matrix Vecr contains the real components of the normalized eigenvector I (for I=1 to N), corresponding to the eigenvalue stored in Evr(I) and Evi(I); the Ith eigenvector is stored in column I.
    d. Matrix Veci contains the imaginary components of the normalized eigenvector I (for I=1 to N), corresponding to the eigenvalue stored in Evr(I) and Evi(I); the Ith eigenvector is stored in column I.

EXAMPLES

EXAMPLE 1


```
ORDER OF MATRIX A(*)= 3

MATRIX A:
 3.300000E+01   1.600000E+01   7.200000E+01
-2.400000E+01  -1.000000E+01  -5.700000E+01
-8.000000E+00  -4.000000E+00  -1.700000E+01


REAL COMPONENTS OF EIGENVALUES:
 1.000000E+00   3.000000E+00   2.000000E+00


IMAGINARY COMPONENTS OF THE EIGENVALUES:
 0.000000E+00   0.000000E+00   0.000000E+00


REAL COMPONENTS OF EIGENVECTORS [CONTAINED IN COLUMNS]
-7.644708E-01   7.844645E-01  -7.619048E-01
 6.115766E-01  -5.883484E-01   6.190476E-01
 2.038589E-01  -1.961161E-01   1.904762E-01


IMAGINARY COMPONENTS OF EIGENVECTORS [CONTAINED IN COLUMNS]:
 0.000000E+00   0.000000E+00   0.000000E+00
 0.000000E+00   0.000000E+00   0.000000E+00
 0.000000E+00   0.000000E+00   0.000000E+00
```

```
EXAMPLE 2



ORDER OF MATRIX A(*)= 4

MATRIX A:
-2.000000E+00  2.000000E+00  2.000000E+00  2.000000E+00
-3.000000E+00  3.000000E+00  2.000000E+00  2.000000E+00
-2.000000E+00  0.000000E+00  4.000000E+00  2.000000E+00
-1.000000E+00  0.000000E+00  0.000000E+00  5.000000E+00


REAL COMPONENTS OF EIGENVALUES:
 1.000000E+00  2.000000E+00  3.000000E+00  4.000000E+00


IMAGINARY COMPONENTS OF THE EIGENVALUES:
 0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00


REAL COMPONENTS OF EIGENVECTORS [CONTAINED IN COLUMNS]
-7.302967E-01  6.255432E-01 -5.547002E-01  5.000000E-01
-5.477226E-01  6.255432E-01 -5.547002E-01  5.000000E-01
-3.651484E-01  4.170288E-01 -5.547002E-01  5.000000E-01
-1.825742E-01  2.085144E-01 -2.773501E-01  5.000000E-01


IMAGINARY COMPONENTS OF EIGENVECTORS [CONTAINED IN COLUMNS]:
 0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
 0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
 0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
 0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
```

HEWLETT hp PACKARD

EIGENANALYSIS: COMPLEX EIGENVECTOR OF A REAL UPPER-HESSENBERG MATRIX

Subprogram Name: Compve

This subprogram finds the complex eigenvector of the real upper-Hessenberg matrix of order N corresponding to the complex eigenvalue with real part in Evr(Ivec) and imaginary part in Evi(Ivec). The inverse iteration method is used in a modified manner to avoid the use of complex arithmetic.

Compve is used in subprogram Eigen which finds the eigenvalues and eigenvectors of a real general matrix.

Subprogram Utilization:

File Name - contained in file "Eigen"

Calling Syntax - CALL Compve (N,M,Ivec,A(*),Vecr(*),H(*),Evr(*),
Evi(*),Indic(*),Subdia(*),Work(*),Eps,Ex)

Input Parameters:

N	Order of matrix A.
M	Order of the submatrix obtained by a suitable decomposition of the upper-Hessenberg matrix if some subdiagonal elements are equal to zero; the value of M is chosen so that the last N-M components of the eigenvector are zero.
Ivec	Gives the position of the eigenvalues in the arrays Evr and Evi for which the corresponding eigenvector is computed.
A(*)	Array used for work space.

Vecr(*)	Array containing the real components of the normalized eigenvector I, (FOR I=1 TO N), corresponding to the eigenvalue stored in Evr(I) and Evi(I); the Ith eigenvector is stored in column I.
H(*)	Array containing upper-Hessbenberg matrix from subprogram Eigen.
Evr(*)	Vector containing the real parts of the eigen-values.
Evi(*)	Vector containing the imaginary parts of the eigenvalues.
Subdia(*)	Vector containing part of upper-Hessenberg matrix from subprogram Eigen.
Work(*)	Array used for work space in inverse iteration process.
Eps	Small positive number that numerically represents zero; from subprogram Hesqr.
Ex	2^{-39}

Output Parameters:

A(*)	The contents of array A are destroyed.
Indic(*)	Vector indicating the success of the subprogram as follows:

Value of Indic(I)	Eigenvector I
1	not found
2	found

*H(*)	Array containing computed eigenvectors; the real parts of the first M components of the computed complex eigenvector will be found in the first M places of the column whose element is Vecr(1,Ivec) and the corresponding imaginary parts of the first M components of the complex eigenvector will be found in the first M places of the column whose top element is Vecr(1,Ivec-1).

*H(*) is both an input and output parameter.

Methods and Formulae:

This subprogram finds the complex eigenvector of the real upper-Hessenberg matrix of order N corresponding to the complex eigenvalue with real part in Evr(Ivec) and imaginary part in Evi(Ivec). The inverse iteration method is used in a modified manner to avoid the use of complex arithmetic.

First, a small perturbation of equal eigenvalues is made if necessary, to obtain a full set of eigenvectors. Then Gaussian elimination of the upper-Hessenberg matrix ((H-Fksi*I)*(H-Fksi*I)+(Eta*Eta)*I) in the array A. The row interchanges that occur are indicated in the array Iwork. All the multipliers are stored in the first and second subdiagonal of array A.

The inverse iteration is performed on the matrix until the infinite norm of the right-hand side vector is greater than the bound defined as Ø.Ø1/(N*Ex). Then the residuals are computed and the residuals of the two successive steps of the inverse iteration are compared. If the infinite norm of the residual vector is greater than the infinite norm of the previous residual vector, then the computed eigenvector of the previous step is taken as the final eigenvector.

HEWLETT hp PACKARD

EIGENANALYSIS: REAL EIGENVECTOR OF A REAL UPPER-HESSENBERG MATRIX

Subprogram Name: Realve

This subprogram finds the real eigenvector of the real upper-Hessenberg matrix in the array A, corresponding to the real eigen value stored in Evr(Ivec). The inverse iteration method is used.

Realve is used in subprogram Eigen which finds the eigenvalues and eigenvectors of a real general matrix.

Subprogram Utilization:

File Name - contained in file "Eigen"

Calling Syntax - CALL Realve(N,M,Ivec,A(*),Vecr(*),Evr(*),
Evi(*),Iwork(*),Work(*),Indic(*),Eps,Ex)

Input Parameters:

N	Order of matrix A.
M	Order of the submatrix obtained by a suitable decomposition of the upper-Hessenberg matrix if some subdiagonal elements are equal to zero; the value of M is chosen so that the last N-M components of the eigenvector are zero.
Ivec	Gives the position of the eigenvalue in the array Evr for which the corresponding eigenvector is computed.
Evr(*)	Vector containing the real parts of the eigenvalues.
Evi(*)	Vector containing the imaginary parts of the eigenvalues.

Eps	Small positive number that numerically represents zero; from subprogram Hesqr.
Ex	2^{-39}

Output Parameters:

Vecr(*)	Array containing the real components of the normalized eigenvector I, (FOR I=1 TO N), corresponding to the eigenvalue stored in Evr(I) and Evi(I); the Ith eigenvector is stored in column I.
Indic(*)	Vector indicating the success of the subprogram as follows:

Value of Indic(I)	Eigenvector I
1	not found
2	found

Methods and Formulae:

This subprogram finds the real eigenvector of the real upper-Hessenberg matrix in the array A, corresponding to the real eigenvalue stored in Evr(Ivec). The inverse iteration method is used.

First, a small perturbation of equal eigenvalues is made if necessary, to obtain a full set of eigenvectors. Then Gaussian elimination of the upper-Hessenberg matrix A is employed. All row interchanges are indicated in the array Iwork. All the multipliers are stored as the subdiagonal elements of A.

The inverse iteration is performed on the matrix until the infinite norm of the right-hand side vector is greater than the bound defined as 0.01/(N*Ex). Then the residuals are computed and the residuals of the two successive steps of the inverse iteration are compared. If the infinite norm of the residual vector is greater than the infinite norm of the previous residual vector, then the computed eigenvector of the previous step is taken as the final eigenvector.

Subprogram Name: Hesqr

This subprogram finds the eigenvalues of a real general matrix. The original matrix A of order N is reduced to upper-Hessenberg form H by means of similarity transformations (Householder's Method).

Hesqr is used in subprogram Eigen which finds the eigenvalues and eigenvectors of a real general matrix.

Subprogram Utilization:

File Name - contained in file "Eigen"

Calling Syntax - CALL Hesqr(N,A(*),H(*),Evr(*),Evi(*),Subdia(*), Indic(*),Eps,Ex)

Input Parameters

N	Order of matrix A
*A(*)	Array containing general matrix A; if used as part of subprogram Eigen, A(*) contains the scaled and normalized matrix A outputted from subprogram Scale; dimensioned A(1:N,1:N).
*H(*)	Array containing original matrix A; H(1:N,1:N)
Ex	2^{-39}

*A(*) and H(*) are both input and output parameters.

Output Parameters:

*A(*)	The input array is destroyed.
*H(*)	Array containing original upper half of matrix H; the special vectors used in the definition of the Householder transformation matrices are stored in the lower part of array H.
Evr(*)	Vector containing the real parts of the N eigenvalues to be found; dimensioned Evr(1:N).
Evi(*)	Vector containing the imaginary parts of the N eigenvalues to be found; dimensioned Evi(1:N).
Subdia(*)	Vector containing parts of input matrix H; dimensioned Subdia (1:N).
Indic(*)	Vector indicating the success of the subprogram as follows:

Value of Indic(I)	Eigenvalue I
Ø	not found
1	found; dimensioned Indic(1:N).

Eps	Small positive number that numerically represents zero in the subprogram; Eps = (Euclidian norm of H)*Ex

Methods and Formulae:

This subprogram finds all the eigenvalues of a real general matrix. The original matrix A of order N is reduced to the upper-Hessenberg form H by means of similarity transformations (Householder Method). The matrix H is preserved in the upper half of the array H and in the array subdia. The special vectors used in the definition of the Householder transformation matrices are stored in the lower part of the array H.

*A(*) and H(*) are both input and output parameters.

The real parts of the N eigenvalues will be dound in the first N places of the array Evr, and the imaginary parts in the first N places of the array Evi. The array Indic indicates the success of the routine as follows

Value of Indic(I)	Eigenvalue I
0	not found
1	found

Eps is a small positive number that numerically represents zero in the program. Eps = (Euclidian norm of H)*Ex.

HEWLETT hp PACKARD

EIGENANALYSIS: SCALES GENERAL MATRIX

Subprogram Name: Scale

This subprogram scales a general matrix so that the quotient of the absolute sum of the off-diagonal elements of column I and the absolute sum of the off-diagonal elements of Row I lies within certain values.

Scale is used in subprogram Eigen which finds the eigenvalues and eigenvectors of a real general matrix.

Subprogram Utilization:

File Name - contained in file "Eigen"

Calling Syntax - CALL Scale (N,A(*),H(*) , Prfact(*),Enorm)

Input Parameters:

N	Order of matrix A.
*A(*)	Array containing matrix to be scaled and normalized so that the Euclidian norm is equal to one; dimensioned A(1:N).

Output Parameters:

*A(*)	Array containing the scaled matrix.
H(*)	Array containing temporary storage for original A(*).

*A(*) is both an input and output parameter.

Prfact(*) Vector containing the scaling factor; the component I of the eigenvector obtained by using the scaled matrix must be divided by the value found in Ith position of Prfact(*). In this way, the eigenvector of the non-scaled matrix is obtained.

Enorm Scaling factor; the eigenvalues of the normalized matrix must be multiplied by Enorm in order that they become the eigenvalues of the non-normalized matrix.

Methods and Formulae:

This subprogram stores the matrix of the order N from the array A into the array H. Afterward the matrix in the array A is scaled so that the quotient of the absolute sum of the off-diagonal elements of column I and the absolute sum of the off-diagonal elements of Row I lies within the values of Boundl and Bound2. The component I of the eigenvector obtained by using the scaled matrix must be divided by the value found in Prfact(I) of the array Prfact. In this way the eigenvector of the non-scaled matrix is obtained.

After the matrix is scaled it is normalized so that the value of the Euclidian norm is equal to one. If the process of scaling was not successful the original matrix from the array H would be stored back into A and the eigenproblem would be solved by using this matrix. The eigenvalues of the normalized matrix must be multiplied by the scalar Enorm in order that they become the eigenvalues of the non-normalized matrix.

For more general information, see subprogram Eigen.

HEWLETT hp PACKARD

EIGENANALYSIS: EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC MATRIX

Subprogram Name: Symqr

This subprogram finds the eigenvalues and, at the user's option, the eigenvectors of a real symmetric matrix. If the matrix is not initially tridiagonal, it is reduced to tridiagonal form by Householder's method. The eigenvalues of the tridiagonal matrix are then calculated by a variant of the QR algorithm with origin shifts.

Subprogram Utilization:

File Name - "Symqr"

Calling Syntax - CALL Symqr (A(*),D(*),E(*),KØ,R,Eps,Abscnv, Vec,Trd)

Input Parameters:

A(*)	Array containing the following: if the matrix is not initially tridiagonal, it is contained in the lower triangle of A(*); if the matrix is initially tridiagonal, input from A(*) is not used; dimensioned A(1:N,1:N).
D(*)	Vector containing the diagonal elements if the matrix is initially tridiagonal; subscripted from 1 to N.

E(*)	Vector containing the off-diagonal elements if the matrix is initially tridiagonal; subscripted from 1 to N-1.
KØ	An initial origin shift to be used until the computed shifts settle down.
N	Order of the matrix, i.e., number of rows or number of columns in the matrix.
Eps	Convergence tolerance.
Abscnv	Abscnv=1 if absolute convergence criterion is to be used. Abscnv=Ø if relative convergence criterion is to be used. NOTE: See Special Considerations and Programming Hints for further details.
Vec	Vec = 1 if eigenvectors are to be computed. Vec = Ø if eigenvectors are not to be computed.
Trd	Trd = 1 if matrix is tridiagonal. Trd = Ø if matrix is not tridiagonal.

Output Parameters:

A(*)	If eigenvectors are not requested, the lower triangle of A(*) is destroyed while the elements above the diagonal are left undisturbed; if eigenvectors are requested, they are returned in the columns of A(*).
D(*)	Vector containing the eigenvalues of the matrix.
E(*)	E(I) contains the number of iterations required to compute the approximate eigenvalue D(I).

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Symqr"

   N = , Eps =

   The user may correct the data from the keyboard (e.g., N = 10, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. The maximum number of iterations allowed per eigenvalue is set at 50. If this is exceeded, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Symqr"

   "MAX # OF ITERATIONS EXCEEDED ON EIGENVALUE" #

   This is a fatal error. (The program may not be continued from this point, but must be restarted.) The program pause allows the user to query other variables in the sub-program environment.

3. To avoid an excessive number of QR steps, an important consideration when eigenvectors are computed, the following guidelines should be followed. The convergence tolerance should not be smaller than the data warrants (reference 1, p. 102). The relative convergence criterion should be used only when there are eigenvalues, small compared to the elements of the matrix, that are nonetheless determined to high relative accuracy.

4. For best results when there is a wide disparity in the sizes of the elements of the matrix, the matrix should be arranged so that the smaller elements appear in the lower right-hand corner.

Methods and Formulae:

Symqr finds the eigenvalues and, at the user's option, the eigenvectors of a real symmetric matrix. If the matrix is not initially tridiagonal, it is reduced to tridiagonal form by Householder's method. The eigenvalues of the tridiagonal matrix are calculated by a variant of the QR algorithm with origin shifts (see reference 2).

Eigenvectors are calculated by accumulating the products of the transformations used in the Householder transformations and the QR steps, a procedure which guarantees a nearly orthonormal set of approximate eigenvectors.

At each QR step, the eigenvalues of the 2x2 submatrix in the lower right-hand corner are computed, and the one nearest the last diagonal element is distinguished. When these numbers settle down, they are used as origin shifts.

The user may choose between absolute and relative convergence criteria. The former accepts the last diagonal element as an approximate eigenvalue when the last off-diagonal element is a small multiple, Eps, of the infinity norm of the matrix. The latter requires that the last off-diagonal element be small compared to the last two diagonal elements.

References:

1. Stewart, G.W. "Eigenvalues and Eigenvectors of a Real Symmetric Matrix, Comm. ACM (June 1970), pp. 384-86.

2. Stewart, G.W. "Incorporating Origin Shifts into the Symmetric QR Algorithm for Symmetric Tridiagonal Matrices", Comm. ACM (June 1970), pp. 365-367.

3. Wilkinson, J.H. The Algebraic Eigenvalue Problem. Clarendon Press, Oxford, 1965.

Driver Utilization:

File Name - "SYMQR", cartridge 1

The driver "SYMQR" sets up the necessary input parameters for the subprogram Symqr as well as prints the results.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 1 with the machine turned on.

2. Get file "SYMQR"
   a. Type: GET "SYMQR"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Symqr ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Symqr contained in file "Symqr" is linked on after the driver.

5. When "ORDER OF A(*), [# OF ROWS OR # OF COLUMNS]?" is displayed:
   a. Enter the order of the matrix, e.g., if A is a 10x10 matrix, enter 1Ø.
   b. Press: CONT.

6. At this point, subprogram Temp is called to dynamically set up the array A(*) and the vectors D(*) and E(*).

7. When "IS MATRIX INITIALLY TRIDIAGONAL (Y/N)?" is displayed:
   a. Enter Y if the matrix is initially tridiagonal.
   b. Press: CONT.
   c. Go to 12.

or

   a. Enter N if the matrix is not initially tridiagonal.
   b. Press: CONT.

8. When "MATRIX A:" is printed and "A(I,J)?" (FOR I=1 TO # OF ROWS IN A(*), FOR J=1 TO I) is displayed:
   a. Enter the Ith, Jth element of matrix A.
   b. Press: CONT.
   c. If more elements are to be entered, go to 8.

9. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the matrix.
   b. Press: CONT.
   c. Go to 10.

or

   a. Enter N if no changes are desired.
   b. Press: CONT.
   c. Go to 20.

10. When "COORDINATES OF A(*), [ROW, COLUMN]?" is displayed:
   a. Enter the row and column numbers of the element to be changed, e.g., if element A(3,4) is to be changed, enter 3,4.
   b. Press: CONT.

11. When "A(I,J)?" (where I and J represent the row and column of the element to be changed) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 9.

12. When "VECTOR D:" is printed and "D(I)?" (FOR I=1 TO # OF ROWS IN A(*)) is displayed:
    a. Enter the Ith element of D(*), i.e., the A(I,I) diagonal element of the matrix.
    b. Press: CONT.
    c. If more elements are to be entered, go to 12.

13. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are desired in the vector.
    b. Press: CONT.
    c. Go to 14.

or

    a. Enter N if no changes are desired.
    b. Press: CONT.
    c. Go to 16.

14. When "VECTOR ELEMENT?" is displayed:
    a. Enter the subscript of the element to be changed.
    b. Press: CONT.

15. When "D(I)?" (where I is the subscript of the element to be changed) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 13.

16. When "VECTOR E:" is printed and "E(I)?" (FOR I=1 TO # OF ROWS IN A(*)-1) is displayed:
    a. Enter the Ith element of E(*), i.e., the A(I+1,I) off-diagonal element of the matrix.
    b. Press: CONT.
    c. If more elements are to be entered, go to 16.

17. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are desired in the vector.
    b. Press: CONT.
    c. Go to 18.

or

    a. Enter N if no changes are desired.
    b. Press: CONT.
    c. Go to 20.

18. When "VECTOR ELEMENT?" is displayed:
    a. Enter the subscript of the element to be changed.
    b. Press: CONT.

19. When "E(I)?" (WHERE I IS THE SUBSCRIPT OF THE ELEMENT TO BE CHANGED) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 17.

20. When "INITIAL ORIGIN SHIFT?" is displayed:
    a. Enter the value for the initial origin shift. Reference 2 has more details on choosing this value. If the user is unsure of a choice, enter Ø.
    b. Press: CONT.

21. When "SHOULD ABSOLUTE CONVERGENCE CRITERION BE USED (Y/N)?" is displayed:

    a. Enter Y if an absolute convergence criterion should be used. This criterion accepts the last diagonal element as an approximate eigenvalue when the last off-diagonal element is a small multiple (Eps asked for in #22) of the infinity norm of the matrix. This criterion should be used except when there are eigenvalues, small compared to the elements of the matrix, that are determined to high relative accuracy.

    b. Press: CONT.

or

    a. Enter N if a relative convergence criterion should be used. This requires that the last off-diagonal be small compared to the last two diagonal elements. This relative convergence criterion should be used only when there are eigenvalues, small compared to the elements of the matrix, that are nonetheless determined to high relative accuracy.

    b. Press: CONT.

22. When "CONVERGENCE TOLERANCE?" is displayed:

    a. Enter the convergence tolerance to be used with #21 above.

    b. To avoid an excessive number of QR steps, an important consideration when eigenvectors are computed, the convergence tolerance should not be smaller than the data warrants.

    c. Press: CONT.

23. When "SHOULD EIGENVECTORS BE COMPUTED (Y/N)?" is displayed:

    a. Enter Y if the eigenvectors of the matrix are to be computed.

    b. Press: CONT.

    c. Go to 24.

or

a. Enter N if no eigenvectors are to be computed.
b. Press: CONT.

24. The eigenvalues, as well as the eigenvectors, if desired, will then be printed. Eigenvector I corresponding to eigenvalue I is contained in column I of the eigenvector matrix.

EXAMPLE

```
ORDER OF A(*)= 4

MATRIX A:
 5.000000E+00  4.000000E+00  1.000000E+00  1.000000E+00
 4.000000E+00  1.000000E+00  4.000000E+00  1.000000E+00
 1.000000E+00  4.000000E+00  5.000000E+00  1.000000E+00
 1.000000E+00  1.000000E+00  1.000000E+00  1.000000E+00


ORDER OF A(*)= 4

MATRIX A:
 5.000000E+00  4.000000E+00  1.000000E+00  1.000000E+00
 4.000000E+00  5.000000E+00  1.000000E+00  1.000000E+00
 1.000000E+00  1.000000E+00  4.000000E+00  2.000000E+00
 1.000000E+00  1.000000E+00  2.000000E+00  4.000000E+00


INITIAL ORIGIN SHIFT= 0.000000E+00
CONVERGENCE TOLERANCE= 1.000000E-05

EIGENVALUES:
 1.000000E+01  5.000000E+00  1.000000E+00  2.000000E+00


EIGENVECTORS:  [CONTAINED IN COLUMNS]
 6.324555E-01  3.162278E-01 -7.071068E-01  0.000000E+00
 6.324555E-01  3.162278E-01  7.071068E-01  1.885618E-12
 3.162278E-01 -6.324555E-01  5.012600E-12 -7.071068E-01
 3.162278E-01 -6.324555E-01  6.012600E-12  7.071068E-01
```

HEWLETT hp PACKARD

INTERPOLATION: CONFLUENT DIVIDED DIFFERENCES

Subprogram Name: Dvdfc

Given a set of real numbers, $\{X_1, X_2, \ldots, X_k\}$, and a corresponding set of function values, $\{V_1, V_2, \ldots, V_k\}$, the forward divided differences are defined as follows:

Zero order differences are

$$\Delta f(X_n) = V_n \quad , n=1, \ldots, k$$

First order differences are

$$\Delta f(X_n, X_{n+1}) = (V_{n+1} - V_n)/(X_{n+1} - X_n), \quad N = 1, \ldots, K-1.$$

Higher order differences are defined in terms of lower order differences:

$$\Delta f(X_I, X_{I+1}, \ldots, X_N) = (\Delta f(X_{I+1}, \ldots, X_N) - \Delta f(X_I, \ldots, X_{N-1}))/(X_N - X_I),$$

$$N = I+2, \ldots, K.$$

The differences may be displayed in the form of a table:

X_1	V_1			
		$\Delta f(X_1, X_2)$		
X_2	V_2		$\Delta f(X_1, X_2, X_3)$	
		$\Delta f(X_2, X_3)$		$\Delta f(X_1, X_2, X_3, X_4)$
X_3	V_3		$\Delta f(X_2, X_3, X_4)$	
		$\Delta f(X_3, X_4)$		
X_4	V_4			

For example:

X	f(X)			
2.0	1.11			
		0.15		
2.2	1.14		3.33E-2	
		0.17		-8.08E-2
2.5	1.19		-5.56E-2	
		0.12		
3.1	1.26			

This program calculates $f(X_1, X_2, \ldots, X_n)$ for any integral value of n in the interval [2,K].

Subprogram Utilization:

File Name: "Dvdfc"

Calling Syntax - FN Dvdfc (N,X(*),V(*),B(*))

Input Parameters:

N	Takes the values N = 2,3,... in turn for calculating the forward divided difference $\Delta f(X_1, X_2, \ldots, X_n)$.
X(*)	Vector containing the initial x-values; the values X(I) need not be distinct or in any special order, but once the vector X is chosen it will determine the interpretation of B(*) and V(*); subscripted from 1 to at least N.

V(*)	Vector containing the values of the functions f(X); subscripted from 1 to at least N.
B(*)	Vector containing backward differences. When N=1, the state of B(*) is irrelevant. When N is greater than 1, B(I) must contain $\Delta f(X_I,\ldots,X_{N-1})$ for I = 1,2,...,N-1 before Dvdfc is called.

Output Parameters:

B(*)	Vector containing backward divided differences. After Dvdfc is called, one will find B(I) = $\Delta f(X_I,X_{I+1},\ldots,X_{N-1},X_N)$ for I = 1,2...,N-1,N.
FNDvdfc	Value of the forward divided difference $\Delta f(X_1,X_2,\ldots,X_N)$.

Special Considerations and Programming Hints:

1. The values X(I) need not be distinct or in any special order, but once the array X is chosen, it will fix the interpretation of B(*) and V(*). If X(1), X(2),...,X(N) are in monotonic order, then the effect of roundoff upon any nth divided difference is no more than would be caused by perturbing each F(X(I)) by n units at most in its last significant place. But if the X's are not in monotonic order, the error can be catastrophic if some of the divided differences are relatively large.

2. The following program segment is an example of how Dvdfc can be used to construct a table of forward or backward differences:

```
FOR I=1 TO N
X(I) =
V(I) =
F(I) = FNDvdfc(I,X(*),V(*),B(*))
NEXT I
```

The array F can be used in subprogram Fnewt or the array B in Bnewt which are also contained in the NUMERICAL ANALYSIS package.

Methods and Formulae:

A full explanation of the algorithm employed may be found in reference 1.

References:

1. Kahan, W. and Farras, I., "Algorithm 167: Calculation of Confluent Divided Differences", Comm. ACM, April 1963.
2. Thacher, H., "Certification of Algorithm 167: Calculation of Confluent Divided Differences", Collected Algorithms from ACM, p. 167-168.

Driver Utilization:

File Name - "DVDFC", cartridge 2

The driver "DVDFC" sets up the necessary input parameters for the subprogram Dvdfc as well as prints the results. At the user's option, subprograms Bnewt, a Newton interpolator with backward divided differences, or Fnewt, a Newton interpolator with forward divided differences, may be called to locate points of interpolation.

With each question, the user is only required to respond with an appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANLAYSIS cartridge 2, with the machine turned on.

2. Get file "DVDFC"
   a. Type: GET "DVDFC"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Dvdfc ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Dvdfc contained in file "Dvdfc" is linked on after the driver.

5. When "NUMBER OF DATA POINTS?" is displayed:
   a. Enter the number of data points to be used in the calculation.
   b. Press: CONT.

6. At this point, subprogram Dvdfctemp is called to dynamically set up arrays X(*), V(*), F(*) and B(*).

7. When "X(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the Ith X-value.
   b. Press: CONT.

8. When "V(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the value of f(X(I)).
   b. Press: CONT.

9. a. Dvdfc is called at this point and the value of the forward divided difference $F(I) = \Delta f(X_1, X_2, \ldots, X_I)$ is calculated as well as the value of B(I). The values X(I), V(I), B(I) and F(I) are then printed. F(*) may be used later in subprogram Fnewt to perform Newton interpolation with forward divided differences and B(*) may be used later in subprogram Bnewt to perform Newton interpolation with backward divided differences.
   b. If more points are to be entered, go to 7.

10. When "NEWTON INTERPOLATION WITH BACKWARD DIVIDED DIFFERENCES (Y/N)?" is displayed:
   a. Enter Y if Newton interpolation is to be performed with backward divided differences.
   b. Press: CONT.
   c. Go to 11.

or

   a. Enter N if Newton interpolation with backward divided differences is not desired.
   b. Press: CONT.
   c. Go to 15.

11. When "HAS Bnewt ALREADY BEEN LINKED ON (Y/N)?" is displayed:
    a. Enter Y if the subprogram has already been linked on.
    b. Press: CONT.
    c. Go to 12.

or

    a. Enter N if the subprogram has not yet been linked on.
    b. Press: CONT.
    c. At this point, subprogram Bnewt contained in file "Bnewt" is linked on. This subprogram is used to perform Newton interpolation with backward divided differences.

12. When "Z[DOMAIN VALUE OF PT. OF INTERPOLATION]?" is displayed:
    a. Enter the domain value of the point to be interpolated.
    b. Press: CONT.

13. The value of the interpolated point and its derivative, as well as an error estimate is then printed.

14. When "ANOTHER ARGUMENT (Y/N)?" is displayed:
    a. Enter Y if the user desires to interpolate another point.
    b. Press: CONT.
    c. Go to 12.

or

    a. Enter N if no more points are to be interpolated using backward divided differences.

15. When "NEWTON INTERPOLATION WITH FORWARD DIVIDED DIFFERENCES (Y/N)?" is displayed:
    a. Enter Y if Newton interpolation is to be performed with forward divided differences.
    b. Press: CONT.
    c. Go to 16.

or

a. Enter N if Newton interpolation with forward divided differences is not desired.
b. Press: CONT.
c. Program terminates.

16. When "HAS Fnewt ALREADY BEEN LINKED ON (Y/N)?" is displayed:
a. Enter Y if the subprogram has already been linked on.
b. Press: CONT.
c. Go to 17.

or

a. Enter N if the subprogram has not yet been linked on.
b. Press: CONT.
c. At this point, subprogram Fnewt contained in file "Fnewt" is linked on. This subprogram is used to perform Newton interpolation with forward divided differences.

17. When "Z [DOMAIN VALUE OF PT. OF INTERPOLATION]?" is displayed:
a. Enter the domain value of the point to be interpolated.
b. Press: CONT.

18. The value of the interpolated point and its derivative, as well as an error estimate is then printed.

HEWLETT hp PACKARD

INTERPOLATION: NEWTON INTERPOLATION WITH BACKWARD DIVIDED DIFFERENCES

Subprogram Name: Bnewt

This subprogram performs Newton interpolation with backward divided differences. Bnewt may be used in conjunction with subprogram Dvdfc, a routine for calculating confluent divided differences.

Subprogram Utilization:

File Name - "Bnewt", cartridge 2

Calling Syntax - CALL Bnewt (Z,N,X(*),B(*),P,D,E)

Input Parameters:

Z	Domain value of the point to be interpolated.
N	Number of data points used.
X(*)	Vector containing the initial x-values; the values X(I) need not be distinct nor in any special order, but the components must correspond to the values in B(*); subscripted from 1 to at least N.
B(*)	Vector containing the backward divided differences B(I) = $\Delta f(X_I, X_{I+1}, \ldots, X_N)$ for I=1 to N; subscripted from 1 to at least N.

Output Parameters:

P	Value of the following polynomial in Z of degree N-1 at most: B(N)+(Z-X(N))*{B(N-1)+(Z-X(N-1){B(N-2)+... +(Z-X(2))B(1)}...}}. This polynomial is an interpolation polynomial which would, but for rounding errors, match the values of the function f(X) and any of its derivatives that subprogram Dvdfc might have been given.
D	Value of the derivative of P.
E	Estimated maximum error in P caused by round-off during the execution of Bnewt.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Bnewt"

   N =

   The user may correct the data from the keyboard (e.g., N=20, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

Methods and Formulae:

A full explanation of the algorithm employed may be found in reference 1.

HEWLETT hp PACKARD

INTERPOLATION: NEWTON INTERPOLATION WITH FORWARD DIVIDED DIFFERENCES

Subprogram Name: Fnewt

This subprogram performs a Newton interpolation with forward divided differences. Fnewt may be used in conjunction with subprogram Dvdfc, a routine for calculating confluent divided differences.

Subprogram Utilization:

File Name - "Fnewt", cartridge 2

Calling Syntax - CALL Fnewt (Z,N,X(*),F(*),R,D,E)

Input Parameters:

Z	Domain value of the point to be interpolated.
N	Number of data points used.
X(*)	Vector containing the initial X-values; the values X(I) need not be distinct nor in any special order, but the components must correspond to the values in B(*); subscripted from 1 to at least N.
F(*)	Vector containing the forward divided differences F(I) = $\Delta f(X_1, X_2, \ldots, X(I))$ for I=1 to N; subscripted from 1 to at least N.

Output Parameters:

R	Value of the following polynomial in Z of degree N-1 at most: F(1)+(Z-X(1))·{F(2)+(Z-X(2)){F(3)+...+ (Z-X(N-1))F(N)}...}}. This polynomial is an interpolation polynomial which would, except for rounding errors, match the values of the function f(X) and any of its derivatives that subprogram Dvdfc might have been given.
D	Value of the derivative of R.
E	Estimated maximum error in R caused by round-off during the execution of Fnewt.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Fnewt"

   N =

   The user may correct the data from the keyboard (e.g., N=5, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

Methods and Formulae:

A full explanation of the algorithm employed may be found in reference 1.

EXAMPLE

```
NUMBER OF DATA POINTS= 11

 I         X(*)              V(*)              B(*)              F(*)

 1    -5.000000E+00    -5.000000E+00    -5.000000E+00    -5.000000E+00
 2    -3.000000E+00    -3.000000E+00    -3.000000E+00     1.000000E+00
 3    -1.000000E+00    -1.000000E+00    -1.000000E+00     0.000000E+00
 4     1.000000E+00     1.000000E+00     1.000000E+00     0.000000E+00
 5     3.000000E+00     3.000000E+00     3.000000E+00     0.000000E+00
 6     5.000000E+00     5.000000E+00     5.000000E+00     0.000000E+00
 7     7.000000E+00     7.000000E+00     7.000000E+00     0.000000E+00
 8     9.000000E+00     9.000000E+00     9.000000E+00     0.000000E+00
 9     1.100000E+01     1.100000E+01     1.100000E+01     0.000000E+00
10     1.300000E+01     1.300000E+01     1.300000E+01     0.000000E+00
11     1.500000E+01     1.500000E+01     1.500000E+01     0.000000E+00


NEWTON INTERPOLATION WITH BACKWARD DIVIDED DIFFERENCES:

Z= 2
INTERPOLATED PT.= 2.000000E+00
DERIVATIVE= 1.000000E+00
ERROR= 2.050000E-06

Z=-2
INTERPOLATED PT.=-2.000000E+00
DERIVATIVE= 1.000000E+00
ERROR= 2.650000E-06

Z= 12
INTERPOLATED PT.= 1.200000E+01
DERIVATIVE= 1.000000E+00
ERROR= 1.050000E-06


NEWTON INTERPOLATION WITH FORWARD DIVIDED DIFFERENCES:


Z= 2
INTERPOLATED PT.= 2.000000E+00
DERIVATIVE= 1.000000E+00
ERROR= 3.450000E-07

Z=-2
INTERPOLATED PT.=-2.000000E+00
DERIVATIVE= 1.000000E+00
ERROR= 1.650000E-07

Z= 12
INTERPOLATED PT.= 1.200000E+01
DERIVATIVE= 1.000000E+00
ERROR= 9.450000E-07
```

```
NUMBER OF DATA POINTS= 13

 I         X(*)              V(*)              B(*)              F(*)

 1     0.000000E+00      0.000000E+00      0.000000E+00      0.000000E+00
 2     1.000000E+00      2.588190E-01      2.588190E-01      2.588190E-01
 3     2.000000E+00      5.000000E-01      5.000000E-01     -8.819000E-03
 4     3.000000E+00      7.071068E-01      7.071068E-01     -2.739367E-03
 5     4.000000E+00      8.660254E-01      8.660254E-01      9.675833E-05
 6     5.000000E+00      9.659258E-01      9.659258E-01      8.015000E-06
 7     6.000000E+00      1.000000E+00      1.000000E+00     -3.108333E-07
 8     7.000000E+00      9.659258E-01      9.659258E-01     -1.000000E-08
 9     8.000000E+00      8.660254E-01      8.660254E-01      4.811508E-10
10     9.000000E+00      7.071068E-01      7.071068E-01     -1.102292E-12
11     1.000000E+01      5.000000E-01      5.000000E-01      1.488095E-12
12     1.100000E+01      2.588190E-01      2.588190E-01     -3.858024E-13
13     1.200000E+01      0.000000E+00      0.000000E+00      6.430040E-14

NEWTON INTERPOLATION WITH BACKWARD DIVIDED DIFFERENCES:

Z= 1.5
INTERPOLATED PT.= 3.826835E-01
DERIVATIVE= 2.418710E-01
ERROR= 8.264025E-07

Z= 8.25
INTERPOLATED PT.= 8.314696E-01
DERIVATIVE=-1.454479E-01
ERROR= 1.945792E-07


NEWTON INTERPOLATION WITH FORWARD DIVIDED DIFFERENCES:


Z= 1.5
INTERPOLATED PT.= 3.826835E-01
DERIVATIVE= 2.418710E-01
ERROR= 2.326118E-03

Z= 11.25
INTERPOLATED PT.= 1.950901E-01
DERIVATIVE=-2.567699E-01
ERROR= 2.893984E-07
```

Subprogram Name: Spline

This subprogram computes a curve s(x) that passes through the N data points (x_i, y_i) supplied by the user.

The integral $\int_{x_1}^{x_n} s(t)\,dt$ is calculated, as well as s(x) and and s´(x) for any point x in the interval $[x_1, x_n]$.

Subprogram Utilization:

File Name - "Spline", cartridge 2

Calling Syntax - CALL Spline (N, Narg, X(*),Y(*), Domain(*), Func(*), Deriv(*), Int, Eps)

Input Parameters:

N	Number of data points.
Narg	Number of arguments for which the derivative or functional value is to be computed; Narg=Ø means that only the integral is to be calculated.
X(*)	Vector containing domain values of the data points subscripted from 1 to N; values should be entered in increasing order.
Y(*)	Vector containing range values of the data points subscripted from 1 to N.

Domain(*)	Vector containing the domain values for which the derivative or functional value is to be computed; subscripted from 1 to Narg; arguments may be entered in any order, but these values must be contained in the interval [x(1),x(N)].
Eps	Error tolerance in iterative solution of simultaneous equations.

Output Parameters:

Func(*)	Vector containing the interpolated function values for output arguments in Domain(*); subscripted from 1 to Narg.
Deriv(*)	Vector containing the derivative values for output arguments in Domain(*); subscripted from 1 to Narg.
Int	Integral $\int_{x_1}^{x_n} s(x)\,dx$.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Spline"

   N =

   The data may be corrected from the keyboard (e.g., Eps=1E-6, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. The arguments for which the derivative or functional values are to be computed must lie in the interval [X(1), X(N)]. If an argument is found outside this range, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Spline"

   "ARGUMENT OUT OF BOUNDS"

   X(1) = , X(N) = , Domain(I) =

   The program may be saved in the following way:

   a. Type: Domain(I) = a permissible value.

   b. Press: EXECUTE.

   c. Type: CONT Corrector. (Type each letter. Do not use the CONT key.).

   d. Press: EXECUTE.

3. The data points (x_i, y_i), for i=1 to n should be entered in increasing order of the x_i, where the x_i are discrete and $x_i < x_{i+1}$ for i=1, n-1. The output arguments t_γ, γ=1, Narg may be entered in any order.

4. The error factor of the solutions is approximately equal to h^4 for the integral, h^3 for the functional values and h^2 for the derivative, where h is the average interval size.

5. X(*), Y(*), Domain(*), Deriv(*) and Func(*) must be dimensioned in the calling program.

Methods and Formulae:

This subprogram computes a curve s(x) that passes through the n data points (x_i, y_i) supplied by the user and computes certain information at any point t_j on the curve as long as t_j is in the interval $[x_1, x_n]$. The information that can be computed is the integral over the interval and the derivative or functional value at any point on the interval.

The method implemented fits a curve through the points and integrates, differentiates and interpolates that curve. The curve used is the cubic natural spline which derives its name from a draftsman's mechanical spline. If the spline is considered as a function represented by s(x), then the second derivative s"(x) approximates the curvature. For the curve through data points (x_1,y_1), (x_2,y_2), ..., (x_n,y_n) we want $\int_{x_1}^{x_n} (s''(x))^2 dx$ to be minimized in order to achieve the "smoothest" curve.

The spline function with minimum curvature has cubic polynomials between adjacent data points. Adjacent polynomials are joined continuously with continuous first and second derivatives as well as $s''(x_1) = s''(x_n) = 0$.

The procedure to determine s(x) involves the iterative solution of a set of simultaneous linear equations by Young's method of successive overrelaxation. The accuracy to which these equations are solved is specified by the user.

The formulae employed are:

FORMULAE:

$$\int_{x_i}^{x_{i+1}} s(x)\,dx \approx \{\frac{1}{2}(x_{i+1} - x_i)(y_i + y_{i+1}) - \frac{1}{24}(x_{i+1} - x_i)^3 [s''(x_i) + s''(x_{i+1})]\}$$

$$s(t_j) = y_i + (t_j - x_i)\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) + (t_j - x_i)(t_j - x_{i+1})\frac{1}{6}[s''(x_i) + s''(x_{i+1}) + s''(t_j)]$$

$$s'(t_j) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{1}{6}[(t_j - x_i) + (t_j - x_{i+1})](s''(x_i) + s''(t_j)) + \frac{1}{6}(t_j - x_i)(t_j - x_{i+1})\left(\frac{s''(x_{i+1}) - s''(x_i)}{x_{i+1} - x_i}\right)$$

REFERENCES:

1. Ralston and Wilf, Mathematical Methods for Digital Computers, Vol. II (New York: John Wiley and Sons, 1967) pp. 156 - 158.

2. Greville, T.N.E., Editor, "Proceedings of An Advanced Seminar Conducted by the Mathematics Research Center", U.S. Army, University of Wisconsin, Madison. October 7 - 9, 1968. Theory and Applications of Spline Functions (New York, London: Academic Press, 1969), pp. 156 - 167.

Driver Utilization:

File Name: "SPLINE", cartridge 2

The driver "SPLINE" sets up the necessary input parameters for the subprogram Spline as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 2, with the machine turned on.

2. Get file "SPLINE"
   a. Type: GET "SPLINE"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Spline ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Spline contained in file "Spline" is linked on after the driver.

5. When "# OF DATA POINTS?" is displayed:
   a. Enter the number of data points to be used in the calculation.
   b. Press: CONT.

6. When "# OF DOMAIN POINTS?" is displayed:
   a. Enter the number of arguments for which the derivative or interpolated functional value is to be computed. If only the value of the integral is desired, enter Ø.
   b. Press: CONT.

7. At this point, subprogram Spline 1 is called to dynamically set up the arrays X(*), Y(*) , Domain(*), Deriv(*) and Func(*).

8. When "ERROR TOLERANCE?" is displayed:
   a. Enter the error tolerance for the iterative solution of the simultaneous equations used in the subprogram.
   b. Press: CONT.

9. When "DATA POINTS:" is printed and "X(I)?" (FOR I=1 TO NUMBER OF DATA POINTS) is displayed:
   a. Enter the appropriate x-value. The data points should be entered so that the X(I) are discrete and X(I)<X(I+1) FOR I=1 TO N-1. The points need not be equally spaced.
   b. Press: CONT.

10. When "Y(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the appropriate y-value.
   b. Press: CONT.
   c. Go to 9 if there are more data values to be entered.

11. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are to be made in the data.
    b. Press: CONT.
    c. Go to 12.

or

    a. Enter N if the data is correct.
    b. Press: CONT.
    c. Go to 15.

12. When "DATA POINT #?" is displayed:
    a. Enter the number of the data point to be changed.
    b. Press: CONT.

13. When "X(I)?" (WHERE I IS THE NUMBER OF THE DATA POINTS TO BE CHANGED) is displayed:
    a. Enter the correct X-value.
    b. Press: CONT.

14. When "Y(I)?" (WHERE I IS THE NUMBER OF THE DATA POINT TO BE CHANGED) is displayed:
    a. Enter the correct Y-value.
    b. Press: CONT.
    c. Go to 11.

15. When "DOMAIN VALUES:" is printed and "DOMAIN (I)?" (FOR I=1 TO NUMBER OF DOMAIN POINTS) is displayed:
    a. Enter the appropriate X-value to be evaluated in the cubic spline. Domain values must be contained in the interval [X(1), X(N)].
    b. Press: CONT.
    c. Go to 15 if there are more domain values to be entered.

16. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are to be made in the data.
    b. Press: CONT.
    c. Go to 17.

or

    a. Enter N if the data is correct.
    b. Press: CONT.
    c. Go to 19.

17. When "DOMAIN VALUE #?" is displayed:
    a. Enter the number of the domain value to be changed.
    b. Press: CONT.

18. When "DOMAIN (I)?" (WHERE I IS THE NUMBER OF THE DOMAIN VALUE TO BE CHANGED) is displayed:
    a. Enter the correct domain value.
    b. Press: CONT.
    c. Go to 16.

19. The program will print the resulting interpolated functional values, the values of the derivative and the value of the integral over the interval [X(1),X(N)].

EXAMPLE.

```
# OF DATA POINTS= 5
# OF DOMAIN POINTS= 4
ERROR TOLERANCE= .000001


DATA POINTS:

   X( 1)=-3.000000E+00      Y( 1)= 3.000000E+00
   X( 2)=-1.000000E+00      Y( 2)= 6.000000E+00
   X( 3)= 2.000000E+00      Y( 3)= 8.000000E+00
   X( 4)= 4.000000E+00      Y( 4)= 2.000000E+00
   X( 5)= 7.000000E+00      Y( 5)= 5.000000E+00


DOMAIN VALUES:

   Domain( 1)=-2.500000E+00
   Domain( 2)= 0.000000E+00
   Domain( 3)= 2.500000E+00
   Domain( 4)= 5.000000E+00


INTEGRAL FROM -3.000000E+00 TO  7.000000E+00 =  4.984962E+01


DOMAIN VALUES    FUNCTION VALUES   DERIVATIVE VALUES

-2.500000E+00      3.691810E+00      1.399138E+00
 0.000000E+00      7.752235E+00      1.573946E+00
 2.500000E+00      6.670259E+00     -3.141092E+00
 5.000000E+00      1.343550E+00      5.030651E-01
```

HEWLETT hp PACKARD

INTERPOLATION: CHEBYSCHEV POLYNOMIAL

Subprogram Name: Cheby

This subprogram fits the tabular function Y(X) (given as m points (X,Y)) by a polynomial

$$P = \sum_{i=\emptyset}^{n} A_i X^i .$$

This polynomial is the best polynomial approximation of Y(X) in the Chebyschev sense.

Subprogram Utilization:

File Name - "Cheby", cartridge 2

Calling Syntax - CALL Cheby (M,N,X(*),Y(*),A(*))

Input Parameters:

M	Number of data points to be used in the computation.
N	Degree of the approximating polynomial; for valid results, M > (N+1).
X(*)	Vector containing the X-values of the data points subscripted from 1 to M; the X(I) must be entered in increasing order.
Y(*)	Vector containing the Y-values of the data points subscripted from 1 to M.

Output Parameters:

A(*)	Vector containing the coefficients of the polynomial approximation subscripted from Ø to N; P(X) = A(Ø)+A(1)*X+A(2)*X↑2+...+A(N)*X↑N.

Special Considerations and Programming Hints:

1. Although this procedure is an implementation of a finite algorithm, roundoff errors may give rise to cyclic changes of the reference set causing the procedure to fail to terminate.

2. X(*),Y(*) and A(*) must be dimensioned in the calling program: DIM X(1:M), Y(1:M), A(Ø,N)

Methods and Formulae:

See the two references for a complete description of the method and formulae employed in subprogram Cheby.

References:

1. Newhouse, Albert, "Chebyschev Curve Fit", Comm. ACM 5 (May 1962), p. 281.

2. Stiefel, E. Numerical Methods of Tchebysheff Approximation, U. of Wisconsin Press (1959), p. 217-232.

Driver Utilization:

File Name - "CHEBY

The driver "CHEBY" sets up the necessary input parameters for the subprogram Cheby as well as prints the coefficients of the resulting polynomial.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 2 with the machine turned on.

2. Get file "CHEBY"
   a. Type: GET "CHEBY"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Cheby ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Cheby contained in file "Cheby" is linked on after the driver.

5. When "# OF DATA POINTS?" is displayed:
   a. Enter the number of data points to be used in the calculation.
   b. Press: CONT.

6. When "DEGREE OF POLYNOMIAL APPROXIMATION?" is displayed:
   a. Enter the degree of the polynomial desired for the approximation. For reasonable results, the degree of the polynomial should be less than the number of data points used minus one, i.e., (degree of polynomial) < (number of data pts. - 1).
   b. Press: CONT.

7. At this point, subprogram Chebytemp is called to dynamically set up the arrays X(*),Y(*) and A(*).

8. When "DATA POINTS:" is printed and "X(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the Ith element of X(*).
   b. Press: CONT.

9. When "Y(I)?" (FOR I=1 TO # OF DATA POINTS) is displayed:
   a. Enter the Ith element of Y(*).
   b. Press: CONT.
   c. If more points are to be entered, go to 8.

10. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are desired in the data.
   b. Press: CONT.
   c. Go to 11.

or

   a. Enter N if no changes are desired in the data.
   b. Press: CONT.
   c. Go to 14.

11. When "DATA POINT #?" is displayed:
    a. Enter the subscript of the element to be changed.
    b. Press: CONT.

12. When "X(I)?" (WHERE I IS THE SUBSCRIPT OF THE ELEMENT TO BE CHANGED) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.

13. When "Y(I)?" (WHERE I IS THE SUBSCRIPT OF THE ELEMENT TO BE CHANGED) is displayed:
    a. Enter the new value of the element.
    b. Press: CONT.
    c. Go to 1Ø.

14. The coefficients of the Chebyschev Polynomial will then be printed where P(X) = A(Ø)+A(1)*X+A(2)*X↑2+...+A(N)*X↑N.

```
# OF DATA POINTS= 8

DEGREE OF POLYNOMIAL APPROXIMATION= 3

DATA POINTS:

   X( 1)=-5.000000E+00      Y( 1)= 2.500000E+01
   X( 2)= 5.000000E+00      Y( 2)= 2.500000E+01
   X( 3)=-4.000000E+00      Y( 3)= 1.600000E+01
   X( 4)= 4.000000E+00      Y( 4)= 1.600000E+01
   X( 5)=-3.000000E+00      Y( 5)= 9.000000E+00
   X( 6)= 3.000000E+00      Y( 6)= 9.000000E+00
   X( 7)=-2.000000E+00      Y( 7)= 4.000000E+00
   X( 8)= 2.000000E+00      Y( 8)= 4.000000E+00

COEFFICIENTS OF CHEBYSCHEV POLYNOMIAL:

   A( 0)= 0.000000E+00
   A( 1)= 0.000000E+00
   A( 2)= 1.000000E+00
   A( 3)= 0.000000E+00


# OF DATA POINTS= 6

DEGREE OF POLYNOMIAL APPROXIMATION= 3

DATA POINTS:

   X( 1)=-4.000000E+00      Y( 1)=-6.000000E+01
   X( 2)=-2.000000E+00      Y( 2)=-8.000000E+00
   X( 3)=-1.000000E+00      Y( 3)= 1.235600E+00
   X( 4)= 1.000000E+00      Y( 4)= 1.000000E+00
   X( 5)= 3.000000E+00      Y( 5)= 2.500000E+01
   X( 6)= 5.000000E+00      Y( 6)= 1.200000E+02

COEFFICIENTS OF CHEBYSCHEV POLYNOMIAL:

   A( 0)=-3.071429E-01
   A( 1)=-1.357143E-01
   A( 2)= 3.571429E-02
   A( 3)= 9.571429E-01
```

HEWLETT hp PACKARD

MATHEMATICAL FUNCTIONS: HYPERBOLIC COSINE

Subprogram Name: FN Cosh

This subprogram calculates the value of the hyperbolic cosine function.

Subprogram Utilization:

File Name - "Cosh", cartridge 2

Calling Syntax - FN Cosh(x)

Input Parameters:

X	Domain value of the function.

Output Parameters:

FN Cosh(x)	Value of the hyperbolic cosine at x.

Methods and Formulae:

The standard formula is used:

$$\text{Cosh}(x) = (e^{x}+e^{-x})/2.$$

References:

1. Hart, J., et.al. Computer Approximations, John Wiley and Sons, Inc., New York, 1968.

Subprogram Name: FN Sinh

This subprogram calculates the value of the hyperbolic sine function.

Subprogram Utilization:

File Name - "Sinh", cartridge 2

Calling Syntax - FN Sinh(X)

Input Parameters:

X Domain value of the function

Output Parameters:

FN Sinh(X) Value of hyperbolic sine at X.

Methods and Formulae:

The hyperbolic sine function may be defined as follows:

$\text{Sinh}(x) = (e^{x}-e^{-x})/2.$

But this formula is not appropriate for a computer approximation. Instead, the positive domain has been segmented into four sections:

1) $x \leq 0.5$: here significance would be lost in the subtraction if the above standard formula was used. Instead, the following polynomial approximation is employed:

$\text{Sinh} \simeq .008420538263*x^{5} + .166656747807*x^{3}+1.00000034157.$

2) $.5 < x \leq 1$:

$\text{Sinh} \approx \frac{1}{2}[D(x)+D(x)/(D(x)+1)]$

where $D(x) = e^{x}-1.$

3) $1<x\leq 10$: Here, the standard formula is used.

4) $x>10$: Here, e^{-x} has lost all significance compared with e^{x}, so we use:

$$\sinh \approx e^{x}/2.$$

The identity sinh(-x) = sinh(x) is used for negative domain values.

Subprogram Name: FN Tanh

This subprogram calculates the value of the hyperbolic tangent function.

Subprogram Utilization:

File Name - "Tanh", cartridge 2

Calling Syntax - FN Tanh(X)

Input Parameters:

X	Domain value of the function.

Output Parameters:

FN Tanh(X)	Value of the hyperbolic tangent function.

Methods and Formulae:

The hyperbolic tangent function may be defined as follows:

$$\text{Tanh}(x) = (e^{x}-e^{-x})/(e^{x}+e^{-x}).$$

But this formula, by itself, is not appropriate for a computer approximation.

Instead, the positive domain has been segmented into three sections:

1) $|x|<1$: here significance would be lost in the subtraction if the above standard formula was used. $\text{Tanh} \approx D(2x)/(2+D(2x))$ where $D(x)=e^x-1$.

2) $1 \leq |x|<10$: here, the standard formula is used.

3) $|x|>10$: here, e^{-x} has lost all significance compared with e^x, so $\text{Tanh} \approx 1$.

The identity $\text{Tanh}(x) = \text{Tan}(-x)$ allows for the absolute values used above.

Subprogram Name: FN Gamma

For a given argument X, (X≠0,-1,-2,...) function subprogram FN Gamma computes the value of the gamma function. For an argument X>70.75, the log gamma function, FNL gamma, should be used to avoid machine overflow.

Subprogram Utilization:

File Name - "Gamma", cartridge 2

Calling Syntax - FN Gamma(X)

Input Paramters:

X	Function argument; X≠Ø,-1,-2,-3,...; arguments must be in the range [-69, 70.957] to avoid machine overflow.

Output Parameters:

FN Gamma(X)	Value of the gamma function evaluated at X.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects an argument equal to zero or a negative integer, the following error message is printed and the program pauses:
   "ERROR IN FUNCTION Gamma"
   "ARGUMENT VALUE IS EITHER Ø OR A NEGATIVE INTEGER"
   Unless the user is willing to change his argument value,

there is no recovery from this error since the function is not able to handle zero or negative integer arguments.

2. If the subprogram detects an argument which will cause machine overflow, the following error message is printed and the program pauses:

   "ERROR IN FUNCTION Gamma"

   "ARGUMENT VALUE OUT OF RANGE"

   ARGUMENT =

   Again, unless the user is willing to change his argument value, there is no recovery from this error. The argument must be between -69 and 70.957.

Methods and Formula:

For a given argument x, ($x \neq \emptyset, -1, -2, \ldots$) this function subprogram computes the value of the gamma function of x, $\Gamma(x)$. The recurrence $\Gamma(Z+n) = \prod_{k=\emptyset}^{n-1} (Z+k)\Gamma(Z)$ allows the computation of $\Gamma(x)$ to be reduced to the computation of $\Gamma(2+x)$, with $0 \leq x \leq 1$. 2 is chosen because of the poles at zero and the negative integers.

Then

$$\Gamma(x) = \prod_{k=\emptyset}^{n-1} (x+2+k)\Gamma(x+2) \quad \text{for } N>\emptyset$$

$$\frac{\Gamma(x+2)}{\prod_{k=1}^{|N|} (x+2-k)} \quad \text{for } N>\emptyset.$$

For large x (in absolute value), the above computation is time consuming, and it is more economical to use an approximation to LOG(T(X)). We use the Stirling form:

$$\text{LOG}(\Gamma(x)) = (x-\tfrac{1}{2})\text{LOG}(x) - x + \text{LOG}(\sqrt{2\pi}) + \Phi(x)$$

where $\Phi(x)$ is a rational approximation of the form:

$$\Phi(x) = \frac{1}{x} R_{n,m}(1/x^2).$$

Subprogram Name: FN Lgamma

For a given real, non negative argument X, function subprogram FN Lgamma computes the value of the natural logarithm of the absolute value of the gamma function.

Subprogram Utilization:

File Name - "Lgamma", cartridge 2

Calling Syntax - FN Lgamma(X).

Input Parameters:

X	Domain value of the function; X must be a real, non negative number.

Output Parameters:

FN Lgamma(X)	Value of the log gamma function evaluated at X.

Methods and Formulae:

For a given real, non negative X, this function subprogram computes the value of the log gamma function of X.

We use the Stirling form:

$$LOG(\Gamma(x))=(x-\tfrac{1}{2})LOG(x)-x+LOG(\sqrt{2\pi})+\Phi(x)$$

where $\Phi(x)$ is a rational approximation of the form:

$$\Phi(x) = \frac{1}{x}R_{n,m}(1/x^2).$$

Subprogram Name: Cadd

Subprogram Cadd adds a set of complex numbers.

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Cadd (N,A(*),B(*),R,I)

Input Parameters:

N	Number of complex numbers to be added.
A(*)	Vector containing the real components of the complex numbers to be added; subscripted from 1 to N.
B(*)	Vector containing the imaginary components of the complex numbers to be added; subscripted from 1 to N.

Output Parameters:

R	Real component of the sum of the complex numbers.
I	Imaginary component of the sum of the complex numbers.

Special Considerations and Programming Hints:

1. If there is a wide range in the magnitude of the numbers to be added, smaller quantities should be stored first in the arrays A(*) and B(*) to avoid roundoff error.

HEWLETT hp PACKARD

COMPLEX FUNCTIONS: MULTIPLICATION WITH COMPLEX NUMBERS

Subprogram Name: Cmult

Subprogram Cmult multiplies two complex numbers.

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Cmult (A1,B1,B2,R,I)

Input Parameters:

A1,A2	Real components of the two complex numbers to be multiplied.
B1,B2	Imaginary components of the two complex numbers to be multiplied.

Output Parameters:

R	Real component of the product.
I	Imaginary component of the product.

COMPLEX FUNCTIONS: DIVISION WITH COMPLEX NUMBERS

Subprogram Name: Cdivid

Subprogram Cdivid divides one complex number by another.

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Cdivid (A1, A2, B1, B2, R,I)

Input Parameters:

A1	Real component of the dividend.
B1	Imaginary component of the dividend.
A2	Real component of the divisor.
B2	Imaginary component of the divisor.

Output Parameters:

R	Real component of the quotient.
I	Imaginary component of the quotient.

HEWLETT hp PACKARD

COMPLEX FUNCTIONS: SQUARE ROOT OF A COMPLEX NUMBER.

Subprogram Name: Csqr

Subprogram Csqr finds the square root of a complex number.

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Csqr (A,B,R,I)

Input Parameters:

A	Real component of the complex number.
B	Imaginary component of the complex number.

Output Parameters:

R	Real component of the square root.
I	Imaginary component of the square root.

Methods and Formulae:

$$R = \frac{A^2+B^2+A}{2} \; ; \; I = \frac{A^2+B^2-A}{2}$$

Special Considerations:

1. To keep the radical real, the positive sign will always be used with the ABS(A+IB).

HEWLETT hp PACKARD

COMPLEX FUNCTIONS: EXPONENTIAL VALUE OF A COMPLEX NUMBER

Subprogram Name: Cexp

Subprogram Cexp finds the exponential value of a complex number.

Subprogram Utilization:

File Name - contained in "Complx"

Calling Syntax - CALL Cexp(A,B,R,I)

Input Parameters:

A	Real component of the complex number.
B	Imaginary component of the complex number.

Output Parameters:

R	Real component of the exponential.
I	Imaginary component of the exponential.

Methods and Formulae:

$R = e^{A}*\cos(B)$; $I=e^{A}*\sin(B)$.

Special Considerations:

1. All of the trigonometric functions require that the argument of the function be given in radians, and that the machine is in the RAD mode.

Subprogram Name: Clog

Subprogram Clog finds the natural logarithm of a complex number.

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Clog (A,B,R,I)

Input Parameters:

A	Real component of the complex number.
B	Imaginary component of the complex number.

Output Parameters:

R	Real component of the natural logarithm.
I	Imaginary component of the natural logarithm.

Methods and Formulae:

$R = LOG(\sqrt{A^2+B^2})$; $I = ATN(B/A)$.

Special Considerations:

1. All the trigonometric functions require that the argument of the function be given in radians, and that the machine is in the RAD mode.

2. The logarithm function is a multivalued function. When this function is used, the principal value of the function is the

calculated result. Thus, the function LOG(EXP(Z)) will not necessarily return the value Z.

COMPLEX FUNCTIONS: ABSOLUTE VALUE OF A COMPLEX NUMBER

Subprogram Name: Cabs

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Cabs (X,Y,Cabs)

Input Parameters:

X	Real component of the complex number.
Y	Imaginary component of the complex number.

Output Parameters:

Cabs	Absolute value of the complex number X+IY.

Subprogram Name: Polyev

Subprogram Polyev evaluates the complex polynomial $(a_0+b_0 i) + (a_1+b_1 i)\ Z + \ldots + (a_n + b_n i)Z^n$ at a complex point.

Subprogram Utilization:

File Name - contained in file "Complx"

Calling Syntax - CALL Polyev (N,A,B,Rcoef(*),Icoef(*),Rval, Ival)

Input Parameters:

N	Degree of the complex polynomial.
A	Real component of the complex number at which the polynomial is to be evaluated.
B	Imaginary component of the complex number at which the polynomial is to be evaluated.
Rcoef(*)	Vector containing the real components of the coefficients of the polynomial (Rcoef (Ø)+Icoef (Ø)*I)+(Rcoef(1)+Icoef(1)*I)*Z+...+(Rcoef(N)+Icoef (N)*I)*Z↑N; subscripted from Ø to N.
Icoef(*)	Vector containing the imaginary components of the coefficients of the polynomial (Rcoef(Ø)+Icoef(Ø) *I)+(Rcoef(1)+Icoef(1)*I)*Z+...+(Rcoef(N)+Icoef(N)*I) *Z↑N; subscripted from Ø to N.

Output Parameters:

Rval	Real component of the evaluated polynomial.
Ival	Imaginary component of the evaluated polynomial.

HEWLETT hp PACKARD

MISCELLANEOUS: FOURIER SERIES COEFFICIENTS FOR EQUALLY-SPACED DATA POINTS

Subprogram Name: Foureq

This subprogram calculates the Fourier Series Coefficients A(I) and B(I) of the Fourier Series corresponding to a function F(X) which is specified by N discrete equally-spaced data points (X(I),Y(I)), I=1,---,N.

Subprogram Utilization:

File Name - "Foureq"

Calling Syntax - CALL Foureq (N,Npts,Init,Incre,Y(*),A(*),B(*))

Input Parameters:

N	Highest numbered Fourier Series coefficient.
Npts	Number of data points (must be odd).
Init	Initial domain value X(1)
Incre	Increment between domain values; $\Delta x=x(I+1)-X(I)$
Y(*)	Vector containing range values for data points subscripted from 1 to Npts.

Output Parameters:

A(*)	Vector containing the A_R Fourier Series coefficients subscripted from $\emptyset$ to N.
B(*)	Vector containing the B_R Fouries Series coefficients subscripted from 1 to N.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Foureq"

   N = , Npts = , Incre =

   The data may be corrected from the keyboard (e.g., N=5, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. Since a Simpson's Method is used in the computation of the coefficients, an odd number of data points must be provided. If this is not the case, the following error message will be printed:

   "ERROR IN SUBPROGRAM Foureq"

   "ODD NUMBER OF DATA POINTS REQUIRED"

   Npts =

3. For valid results, the user should provide at least N data points to compute N coefficients.

4. Y(*),A(*) and B(*) must all be dimensioned in the calling program.

Methods and Formulae:

If: $g(x) = f(x) \cos\left(\frac{2\pi i x}{T}\right)$

and $h(x) = f(x) \sin\left(\frac{2\pi i x}{T}\right)$ then:

$$a_i \approx \frac{2\Delta x}{3T}\{g(x_1) + 4g(x_2) + 2g(x_3) + 4g(x_4) + \ldots + 4g(x_{n-1}) + g(x_n)\}$$

$$b_i \approx \frac{2\Delta x}{3T}\{h(x_1) + 4h(x_2) + 2h(x_3) + 4h(x_4) + \ldots + 4h(x_{n-1}) + h(x_n)\}$$

Sine and cosine functional values are computed recursively with the following formula:

$$\sin\left(\frac{2\pi x_i}{\Delta x}(J+1)\right) = \sin\left(\frac{2\pi x_i}{\Delta x}\right)\cos\left(\frac{2\pi x_i}{\Delta x}J\right) + \cos\left(\frac{2\pi x_i}{\Delta x}\right)\sin\left(\frac{2\pi x_i}{\Delta x}J\right)$$

$$\cos\left(\frac{2\pi x_i}{\Delta x}(J+1)\right) = \cos\left(\frac{2\pi x_i}{\Delta x}\right)\cos\left(\frac{2\pi x_i}{\Delta x}J\right) - \sin\left(\frac{2\pi x_i}{\Delta x}\right)\sin\left(\frac{2\pi x_i}{\Delta x}J\right)$$

REFERENCES:

1. Hamming, R.W., Numerical Methods for Scientists and Engineers (McGraw-Hill, 1962), pp. 67-80.

2. Acton, Forman S., Numerical Methods that Work (Harper and Row, 1970), pp. 221-257.

Driver Utilization:

File Name - "FOUREQ", cartridge 2

The driver "FOUREQ" sets up the necessary input parameters for the subprogram Foureq as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 2 with the machine turned on.

2. Get file "FOUREQ"
   a. Type: Get "FOUREQ"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Foureq ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CTON.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Foureq contained in file "Foureq" is linked on after the driver.

5. When "# OF DATA POINTS (ODD # REQUIRED) " is displayed:
   a. Enter the number of data points to be used in the calculation. There must be an odd number of points.
   b. Press: CONT.

6. When "HIGHEST FOURIER SERIES COEFFICIENT?" is displayed:
   a. Enter the highest coefficient desired in the Fourier Series. This should be less than or equal to the number of data points for valid results.
   b. Press: CONT.

7. At this point, subprogram Four1 is called to dynamically set up the vectors A(*),B(*) and Y(*).

8. When "INITIAL DOMAIN VALUE?" is displayed:
   a. Enter the initial domain value, i.e., the first x value.
   b. Press: CONT.

9. When "INCREMENT?" is displayed:
   a. Enter the increment between x values.
   b. Press: CONT.

10. When "RANGE VALUES:" is printed and "Y(I)?" (FOR I=1 TO NPTS) is displayed:
   a. Enter the appropriate y-value.
   b. Press: CONT.
   c. Go to 10 if there are more data values to be entered.

11. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are to be made in the data.
   b. Press: CONT.
   c. Go to 12.

or

a. Enter N if the data is correct.
b. Press: CONT.
c. Go to 14.

12. When "DATA POINT #?" is displayed:
a. Enter the number of the data point to be changed.
b. Press: CONT.

13. When "Y(I)?" is displayed:
a. Enter the correct y-value.
b. Press: CONT.
c. Go to 11.

EXAMPLE

```
EXAMPLE FOR FOURE0 IN FOURIER SERIES SECTION.



# OF DATA POINTS= 31
HIGHEST FOURIER SERIES COEFFICIENT= 5
INITIAL DOMAIN VALUE= 0
INCREMENT= .1


RANGE VALUES:

   Y( 1)= 1.500000E+00
   Y( 2)= 3.000000E+00
   Y( 3)= 3.000000E+00
   Y( 4)= 3.000000E+00
   Y( 5)= 3.000000E+00
   Y( 6)= 3.000000E+00
   Y( 7)= 3.000000E+00
   Y( 8)= 3.000000E+00
   Y( 9)= 3.000000E+00
   Y(10)= 3.000000E+00
   Y(11)= 1.500000E+00
   Y(12)= 0.000000E+00
   Y(13)= 0.000000E+00
   Y(14)= 0.000000E+00
   Y(15)= 0.000000E+00
   Y(16)= 0.000000E+00
   Y(17)= 0.000000E+00
   Y(18)= 0.000000E+00
   Y(19)= 0.000000E+00
   Y(20)= 0.000000E+00
   Y(21)= 1.000000E+00
   Y(22)= 2.000000E+00
   Y(23)= 2.000000E+00
   Y(24)= 2.000000E+00
   Y(25)= 2.000000E+00
   Y(26)= 2.000000E+00
   Y(27)= 2.000000E+00
   Y(28)= 2.000000E+00
   Y(29)= 2.000000E+00
   Y(30)= 2.000000E+00
   Y(31)= 1.000000E+00


COEFFICIENTS:

   A( 0)= 1.638889E+00
   A( 1)= 1.322781E+00     B( 1)= 4.774700E-01
   A( 2)=-7.448371E-01     B( 2)= 2.387741E-01
   A( 3)=-5.555556E-02     B( 3)=-2.222222E-13
   A( 4)= 2.900530E-01     B( 4)= 1.197223E-01
   A( 5)=-3.333333E-01     B( 5)= 9.622504E-02
```

HEWLETT hp PACKARD

MISCELLANEOUS: FOURIER SERIES COEFFICIENTS FOR UNEQUALLY--SPACED DATA POINTS

Subprogram Name: Fourun

This subprogram calculates the Fourier Series Coefficients for a function defined by discrete data points (X(I),Y(I)), I=1,---,N. The data pairs must be entered such that the X(I) are discrete, but not necessarily equally-spaced, and X(I)< X(I+1) for I=1,---, N-1.

Subprogram Utilization:

File Name - "Fourun"

Calling Syntax - CALL Fourun (N,Npts,X(*),Y(*),A(*),B(*))

Input Parameters:

N	Highest numbered Fourier Series coefficient.
Npts	Number of data points.
X(*)	Vector containing domain values for data points subscripted from 1 to Npts.
Y(*)	Vector containing range values for data points subscripted from 1 to Npts.

Output Parameters:

A(*)	Vector containing the A_k Fourier Series coefficients subscripted from Ø to N.
B(*)	Vector containing the B_k Fourier Series coefficients subscripted from 1 to N.

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Fourun"

   N = , Npts =

   X(1) = , X(Npts) =

   The data may be corrected from the keyboard (e.g., N=5, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. For valid results, the user should provide at least N data points to compute N coefficients.

3. X(*),Y(*),A(*),B(*) must all be dimensioned in the calling subprogram.

4. The data points (x_i,y_i) should be entered so that the x_i are discrete and $x_i<x_{i+1}$ for i=1 to N-1. The points need not be equally spaced.

Methods and Formulae:

This subprogram calculates the Fourier Series coefficients a_i and b_i of the Fourier Series corresponding to a function f(x) which is specified by N discrete data points (x_i,y_i), i=1,N.

The finite Fourier Series is given by the formula:

$\frac{a_0}{2} + \sum_{i=1}^{N} \left(a_i \cos \frac{i\pi x}{T} + b_i \sin \frac{i\pi x}{T}\right)$ where the Fourier coefficients a_i and b_i are:

$$a_i = \frac{2}{T} \int_{x_1}^{x_1+T} f(x)\cos \frac{2\pi i x}{T}\, dx \qquad \text{for } i=\emptyset, N.$$

$$b_i = \frac{2}{T} \int_{x_1}^{x_1+T} f(x)\sin \frac{2\pi i x}{T}\, dx \qquad \text{for } i=1, N.$$

T specifies the period equivalent to $(x_n - x_1)$ and N indicates the number of coefficients desired. The coefficients are evaluated by numerically integrating a parabola passing through three successive points. Execution time depends on the number of coefficients calculated.

FORMULAE:

$$A_j = \sum_{i=2}^{k-1} S_i \qquad \text{for } j = 1, n$$

$$B_j = \sum_{i=2}^{k-1} T_i \qquad \text{for } j = 1, n$$

Where:

$$Q_i = \frac{(x_{i+1}-x_{i+2})\, y_i - (x_i-x_{i+2})\, y_{i+1} + (x_i-x_{i+1})\, y_{i+2}}{(x_{i+1}-x_{i+2})\,(x_i-x_{i+2})\,(x_i-x_{i+1})}$$

$$A = \frac{1}{2}\,(Q_i+Q_{i-1})$$

$$R_i = \frac{-(x_{i+1}^2 - x_{i+2}^2)\, y_i - (x_i^2-x_{i+2}^2)\, y_{i+1} + (x_i^2-x_{i+1}^2)\, y_{i+2}}{(x_{i+1}-x_{i+2})\,(x_i-x_{i+2})\,(x_i-x_{i+1})}$$

$$B = \frac{1}{2}\,(R_i+R_{i-1})$$

$$S_i = \frac{(2Ax_{i+1}+B)\cos\left[2\pi\left(\frac{x_{i+1}J}{x_k-x_1}\right)\right] - (2Ax_i+B)\cos\left[2\pi\left(\frac{x_iJ}{x_k-x_1}\right)\right]}{\left(\frac{J}{x_k-x_1}\right)^2} +$$

$$\frac{y_{i+1}\left(\frac{J}{x_k-x_1}\right)^2 - 2A}{\left(\frac{2\pi J}{x_k-x_1}\right)^3} \sin\left[2\pi\left(\frac{x_{i+1}J}{x_k-x_1}\right)\right] -$$

$$\frac{y_i\left(\frac{J}{x_k-x_1}\right)^2 - 2A}{\left(\frac{2\pi J}{x_k-x_1}\right)^3} \sin\left[2\pi\left(\frac{x_iJ}{x_k-x_1}\right)\right]$$

$$T_i = \frac{(2Ax_{i+1}+B)\ \sin[2\pi frc(\frac{x_{i+1}J}{x_k-x_1})] - (2Ax_i+B)\ \sin[2\pi frc(\frac{x_iJ}{x_k-x_1})]}{(\frac{J}{x_k-x_1})^2} -$$

$$\frac{y_{i+1}(\frac{J}{x_k-x_1})^2 - 2A}{(\frac{2\pi J}{x_k-x_1})^3} \cos[2\pi(\frac{x_{i+1}J}{x_k-x_1})] +$$

$$\frac{y_i(\frac{J}{x_k-x_1})^2 - 2A}{(\frac{2\pi J}{x_k-x_1})^3} \cos[2\pi(\frac{x_iJ}{x_k-x_1})]$$

Where J is the number of coefficient being computed and frc ( ) indicates fractional part.

$$A_0 = \sum_{i=2}^{k=1} U_i$$

$$U_i = \frac{A(x_{i+1}^3-x_i^3)}{3} + \frac{B(x_{i+1}^2-x_i^2)}{2} + C(x_{i+1}-x_i)$$

Where A, B are defined above and:

$$C = \frac{1}{2}(P_i+P_{i-1})$$

$$P_i = \frac{(x_{i+1}-x_{i+2})x_{i+1}x_{i+2}y_i-(x_i-x_{i+2})x_ix_{i+2}y_{i+1} + (x_i-x_{i+1})x_ix_{i+1}y_{i+2}}{(x_{i+1}-x_{i+2})\ (x_i-x_{i+2})\ (x_i-x_{i+1})}$$

Sine and cosine functional values are computed recursively with the

following formulae:

$$\sin\left(\frac{2\pi x_i}{x_k - x_1}(J+1)\right) = \sin\left(\frac{2\pi x_i}{x_k - x_1}\right)\cos\left(\frac{2\pi x_i}{x_k - x_1}J\right) + \cos\left(\frac{2\pi x_i}{x_k - x_1}\right)\sin\left(\frac{2\pi x_i}{x_k - x_i}J\right)$$

$$\cos\left(\frac{2\pi x_i}{x_k - x_1}(J+1)\right) = \cos\left(\frac{2\pi x_i}{x_k - x_1}\right)\cos\left(\frac{2\pi x_i}{x_k - x_1}J\right) - \sin\left(\frac{2\pi x_i}{x_k - x_1}\right)\sin\left(\frac{2\pi x_i}{x_k - x_1}J\right)$$

Where i is the data point number and J is the coefficient number

REFERENCES:

1. Hewlett Packard 9820A Math Pac, pp. 43 - 50.

2. Hamming, R.W. Numerical Methods for Scientists and Engineers (M^{C}Graw-Hill, 1962), pp. 67 - 80.

3. Acton, Forman S., Numerical Methods that Work (Harper and Row, 1970), pp. 221 - 257.

Driver Utilization:

File Name - "FOURUN", cartridge 2

The driver "FOURUN" sets up the necessary input parameters for the subprogram Fourun as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 2 with the machine turned on.

2. Get file "FOURUN"
   a. Type: GET "FOURUN"
   b. Press: EXECUTE.

3. Press: RUN.

4. When "HAS Fourun ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Fourun contained in file "Fourun" will be linked on after the driver.

5. When "# OF DATA POINTS?" is displayed:
   a. Enter the number of data points to be used in the calculation.
   b. Press: CONT.

6. When "HIGHEST FOURIER SERIES COEFFICIENT?" is displayed:
   a. Enter the highest Fourier Series Coefficient desired. For valid results, this should be less than or equal to the number of data points provided.
   b. Press: CONT.

7. At this point, subprogram Four1 is called to dynamically set up the arrays A(*),B(*),X(*), and Y(*).

8. When "DATA VALUES:(X VALUES MUST BE DISTINCT AND IN INCREASING ORDER)" is printed and "X(I)?" (FOR I=1 TO Npts) is displayed:
   a. Enter the appropriate x-value. The data points should be entered so that the X(I) are discrete and X(I)<X(I+1) for I=1 to Npts-1. The points need not be equally spaced.
   b. Press: CONT.

9. When "Y(I)?" (FOR I=1 TO Npts) is displayed:
   a. Enter the appropriate y-value.
   b. Press: CONT.
   c. Go to 8 if there are more data values to be entered.

10. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are to be made in the data.
   b. Press: CONT.
   c. Go to 11.

or

a. Enter N if the data is correct.
b. Press: CONT.
c. Go to 14.

11. When "DATA POINT #?" is displayed:
a. Enter the number of the data point to be changed.
b. Press: CONT.

12. When "X(I)?" (WHERE I IS THE SUBSCRIPT OF THE DATA POINT TO BE CHANGED) is displayed:
a. Enter the correct x-value.
b. Press: CONT.

13. When "Y(I)?" (WHERE I IS THE SUBSCRIPT OF THE DATA POINT TO BE CHANGED) is displayed:
a. Enter the correct y-value.
b. Press: CONT.
c. Go to 10.

14. The program will print the resulting A_k and B_k Fourier Series coefficients.

EXAMPLE

```
# OF DATA POINTS= 37
HIGHEST FOURIER SERIES COEFFICIENT= 10


DATA VALUES:

   X( 1)=0.000000E+00     Y( 1)=1.500000E+00
   X( 2)=1.000000E-01     Y( 2)=3.000000E+00
   X( 3)=1.500000E-01     Y( 3)=3.000000E+00
   X( 4)=2.000000E-01     Y( 4)=3.000000E+00
   X( 5)=2.500000E-01     Y( 5)=3.000000E+00
   X( 6)=3.000000E-01     Y( 6)=3.000000E+00
   X( 7)=4.000000E-01     Y( 7)=3.000000E+00
   X( 8)=5.000000E-01     Y( 8)=3.000000E+00
   X( 9)=6.000000E-01     Y( 9)=3.000000E+00
   X(10)=7.000000E-01     Y(10)=3.000000E+00
   X(11)=8.000000E-01     Y(11)=3.000000E+00
   X(12)=9.000000E-01     Y(12)=3.000000E+00
   X(13)=1.000000E+00     Y(13)=1.500000E+00
   X(14)=1.100000E+00     Y(14)=0.000000E+00
   X(15)=1.200000E+00     Y(15)=0.000000E+00
   X(16)=1.300000E+00     Y(16)=0.000000E+00
   X(17)=1.500000E+00     Y(17)=0.000000E+00
   X(18)=1.700000E+00     Y(18)=0.000000E+00
   X(19)=1.900000E+00     Y(19)=0.000000E+00
   X(20)=2.000000E+00     Y(20)=0.000000E+00
   X(21)=2.500000E+00     Y(21)=0.000000E+00
   X(22)=2.750000E+00     Y(22)=0.000000E+00
   X(23)=3.000000E+00     Y(23)=0.000000E+00
   X(24)=3.500000E+00     Y(24)=0.000000E+00
   X(25)=3.700000E+00     Y(25)=0.000000E+00
   X(26)=3.900000E+00     Y(26)=0.000000E+00
   X(27)=4.000000E+00     Y(27)=0.000000E+00
   X(28)=4.250000E+00     Y(28)=0.000000E+00
   X(29)=4.500000E+00     Y(29)=0.000000E+00
   X(30)=4.800000E+00     Y(30)=0.000000E+00
   X(31)=5.000000E+00     Y(31)=0.000000E+00
   X(32)=5.200000E+00     Y(32)=0.000000E+00
   X(33)=5.400000E+00     Y(33)=0.000000E+00
   X(34)=5.800000E+00     Y(34)=0.000000E+00
   X(35)=5.850000E+00     Y(35)=0.000000E+00
   X(36)=5.900000E+00     Y(36)=0.000000E+00
   X(37)=6.000000E+00     Y(37)=1.500000E+00
```

A COEFFICIENTS	B COEFFICIENTS
5.000000E-01	
8.262358E-01	4.768298E-01
4.119709E-01	7.131732E-01
5.566667E-12	6.307360E-01
-2.036138E-01	3.520154E-01
-1.614057E-01	9.246861E-02
1.000000E-12	-7.230697E-04
1.122901E-01	6.417231E-02
9.651357E-02	1.666468E-01
-1.700000E-12	1.936027E-01
-7.365751E-02	1.275692E-01

Subprogram Name: Fft

This subprogram calculates a Fast Fourier Transform from a set of time domain points to a set of frequency domain points.

At the user's option, the inverse Fast Fourier Transform, calculating the set of time domain points from a set of frequency domain points, may also be calculated.

Subprogram Utilization:

File Name - "Fft", cartridge 2

Calling Syntax - CALL Fft(N,Power,Flg,R(*),I(*))

Input Parameters:

N	(Number of time domain points)/2; or (number of frequency coefficient pairs)/2+1.
Power	$N = 2^{(power\ -1)}$
Flg	Flg=1 calculate Fast Fourier Transform from set of time domain points to set of frequency domain points. Flg=-1 calculate Inverse Fast Fourier Transform from set of frequency domain points to set of time domain points.
R(*)	Vector subscripted from 1 to N; as time domain data, R(*) contains the odd numbered data points, i.e., R(J) = time domain in data for I=1 to N; as frequency domain data, R(1) contains the DC term, and R(J) = real component of data (J-1) for J=2 to N.

*I(*) Vector subscripted from 1 to N; as time domain data, I(*) contains the even-numbered data points, i.e., I(J) = time domain data (2*J) for J=1 to N; as frequency domain data, I(1) contains the maximum frequency term, and

I(J) = Imaginary component of data (J-1) for J=2 to N.

Output Parameters:

*R(*) Vector subscripted from 1 to N (see R(*) as an input parameter).

*I(*) Vector subscripted from 1 to N (see I(*) as an input parameter).

Special Considerations and Programming Hints:

1. Upon entry into the subprogram, there is a BAD DATA CHECK. If the subprogram detects "nonsense" data, the following error message is printed and the program pauses:

   "ERROR IN SUBPROGRAM Fft"

   "N = ", "Flg = ", "Power = "

   The data may be corrected from the keyboard (e.g., Flg=1, EXECUTE). When CONT is pressed, the program will resume execution at the next line.

2. The following tables may be used as guidelines for memory and time requirements:

* R(*) and I(*) are both input and output parameters.

Memory Size

Memory Size	Full Precision Time Data	Full Precision Frequency Data	Short Precision Time Data	Short Precision Frequency Data	Integer Precision Time Data	Integer Precision Frequency Data
Standard 56426 Bytes	4096	2048	8192	4096	16384	8192
Option 204 187306 Bytes	16384	8192	32768	16384	32768*	16384*
Option 205 318186 Bytes	32768	16384	32768*	16384*	32768*	16384*
Option 206 449 066 Bytes	32768*	16384*	32768*	16384*	32768*	16384*

# points	FFT Model 1xx	IFT Model 1xx	FFT Model 2xx	FFT Model 2xx
256	17 sec	15.5 sec	2 sec	2 sec
512	35 sec	32.5 sec	4.5 sec	4 sec
1024	73 sec	69 sec	10 sec	9 sec
2048	155 sec	147 sec	21 sec	19.5 sec
4096	330 sec	314 sec	45 sec	42 sec

Methods and Formulae:

This method uses a modification of the basic FFT algorithm. The modified algorithm takes advantage of the fact that series data will be real and the space normally reserved for the imaginary part of the complex sequence can be used to calculate a double-length real transform. This is represented for two "N" length transforms as:

$Z(n)=X(n)+i\ Y(n)\quad \emptyset \leq n < N$ data points

The transform is:

$Z(m)=X(m)+i\ Y(m)$

where

$$X(m)=\frac{Z(m)+Z^*(N-m)}{2}$$

$$Y(m)=\frac{Z(m)-Z^*(N-m)}{2i}$$

Z* is the complex conjugate of Z.

The time series F(n) is given by:

$F(n)=X(2n)+Y(2n+1)$.

The advantage gained from this adaptation of the general FFT algorithm for time series data are:

A. A transform of twice the length can be handled with no increase in storage for input data.
B. Since the calculation of the transform is treated as an interactive process intermediate and final results are stored in the same locations used for input.

References:

1. Brigham, E.O., The Fast Fourier Transform, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1974, Chapter 10.

2. Cooley, J.W., and Tukey, J.W., "An Algorithm for Machine Calculation of Complex Fourier Series," Math. Computation. Vol. 19, pp. 297-301, April 1965.

The transform of this is:

$$F(m)= \sum_{n=\emptyset}^{N-1} X(2n)w^{mn} + \sum_{n=\emptyset}^{N-1} Y(2n+1)w^{mn}$$

$$= \sum_{p=\emptyset}^{N-1} X(p)w^{2mp} + \sum_{p=\emptyset}^{N-1} Y(p)w^{2mp}w^{m}$$

and:

$$F(m)=X(m) + w^{m}Y(m) \qquad (1)$$

$$F(N-m) = X^*(m) - [w^{m} Y(m)]^* \qquad (2)$$

Similarly the inverse transform may be obtained from

(1) and (2):

$$Z(m) = \frac{F(m) + F^*(N-m)}{2} + i\, w^{-m} \frac{F(m) - F^*(N-m)}{2}$$

$$Z(N-m) = \left[\frac{F(m) + F^*(N-m)}{2}\right]^* - iw^{-m} \left[\frac{F(m) - F^*(N-m)}{2}\right]^*$$

This is simply an interchange of Z(m) and F(m) in (1) and (2), and substitution of $(-w^{-m})$ for w^{m}.

Note:

A. Since F(∅) and F(N) are real only, F(N) can be stored in the imaginary location of F(∅), i.e., F(1).
B. $w^{m} = c^{-2\pi im/2N}$. This is half the minimum value of rotation normally used in an N point transform.
C. * = complex conjugate.

Driver Utilization:

File Name - "FFT", cartridge 2

The driver "FFT" sets up the necessary input parameters for the subprogram FFT as well as prints the resulting outputs.

With each question, the user is only required to respond with the appropriate answer.

User Instructions:

1. Insert the NUMERICAL ANALYSIS cartridge 2, with the machine turned on.

2. Get file "FFT"
   a. Type: GET "FFT"
   b. Press: CONT.

3. Press: RUN.

4. When "HAS Fft ALREADY BEEN LINKED ON (Y/N)?" is displayed:
   a. Enter Y if the subprogram has already been linked on.
   b. Press: CONT.
   c. Go to 5.

or

   a. Enter N if the subprogram has not yet been linked on.
   b. Press: CONT.
   c. At this point, subprogram Fft contained in file "Fft" is linked on after the driver.

5. When "INPUT TIME OR FREQUENCY DATA (T/F)?" is displayed:
   a. Enter T if time data is to be entered.
   b. Press: CONT.
   c. Go to 6.

or

   a. Enter F if frequency data is to be entered.
   b. Press: CONT.
   c. Go to 13.

6. When "# OF DATA POINTS [POWER OF 2 FROM 4 to 1024]?" is displayed:
   a. Enter the number of time domain points to be used in the calculations. This number must be a power of 2 between 4 and 1024.
   b. Press: CONT.

7. At this point, a check is made on the number of data points. If an error is detected, the following message will be printed:

   "# OF POINTS OUT OF RANGE (4,1024)."
   "N = "

   When "ENTER NUMBER OF POINTS AGAIN" is displayed:
   a. Enter the number of time domain points.
   b. Press: CONT.

8. When "TIME DOMAIN DATA:" is printed and "DATA POINT (J)" (FOR J=1 TO # OF TIME DOMAIN POINTS) is displayed:
   a. Enter the appropriate time domain point.
   b. Press: CONT.
   c. Go to 8 if there are more points to be entered.

9. When "CHANGES (Y/N)?" is displayed:
   a. Enter Y if changes are to be made in the data.
   b. Press: CONT.
   c. Go to 1Ø.

or

a. Enter N if no changes are desired in the data.
b. Press: CONT.
c. Go to 12.

10. When "DATA POINT #?" is displayed:
a. Enter the number of the data point to be changed.
b. Press: CONT.

11. When "DATA POINT (I)?" (WHERE I IS THE COORDINATE OF THE POINT TO BE CHANGED) is displayed:
a. Enter the correct data point value.
b. Press: CONT.
c. Go to 9.

12. The program will print the "DC TERM", the "MAX FREQUENCY" and the real and imaginary components of the resulting frequency data. And the program ends.

13. When "(# OF COEFFICIENT PAIRS *2)+2 [POWER OF 2 BETWEEN 4 and 1024]?" is displayed:
a. Enter the value of the number of coefficient pairs times 2 plus 2. So, for example, if there are 15 coefficient pairs, we would enter 32 = 15*2+2. The number entered must be a power of 2 between 4 and 1024.
b. Press: CONT.

14. At this point, a check is made on the number of data points. If an error is detected, the following message will be printed:
"# OF POINTS OUT OF RANGE (4,1024)"
"N = "
When "ENTER NUMBER OF POINTS AGAIN." is displayed:
a. Enter the number of time domain points.
b. Press: CONT.

15. When "DC TERM?" is displayed:
    a. Enter the DC term.
    b. Press: CONT.

16. When "MAX FREQUENCY TERM?" is displayed:
    a. Enter the maximum frequency term.
    b. Press: CONT.

17. When "FREQUENCY DOMAIN DATA:" is printed and "REAL (J)" (FOR J=1 TO # OF POINTS) is displayed:
    a. Enter the appropriate real coefficient.
    b. Press: CONT.

18. When "IMAG(J)" (FOR J=1 TO # OF POINTS) is displayed:
    a. Enter the appropriate imaginary coefficient.
    b. Press: CONT.
    c. Go to 17 if there are more points to be entered.

19. When "CHANGES (Y/N)?" is displayed:
    a. Enter Y if changes are to be made in the data.
    b. Press: CONT.
    c. Go to 20.

or

    a. Enter N if no changes are desired in the data.
    b. Press: CONT.
    c. Go to 23.

20. When "DATA POINT #?" is displayed:
    a. Enter the number of the data point to be changed.
    b. Press: CONT.

21. When "REAL (J)?" (WHERE J IS THE COORDINATE OF THE REAL POINT TO BE CHANGED) is displayed:
    a. Enter the correct real coefficient.
    b. Press: CONT.

22. When "IMAG (J)?" (WHERE J IS THE COORDINATE OF THE IMAGINARY POINT TO BE CHANGED) is displayed:
    a. Enter the coefficient.
    b. Press: CONT.

23. The program will then print the time domain data points obtained from the inverse Fast Fourier Transform.

EXAMPLE

```
EXAMPLE 1


# OF DATA POINTS= 16


TIME DOMAIN DATA:

   POINT(  1)= 0.000000E+00     POINT(  2)= 3.826834E-01
   POINT(  3)= 7.071068E-01     POINT(  4)= 9.238795E-01
   POINT(  5)= 1.000000E+00     POINT(  6)= 9.238795E-01
   POINT(  7)= 7.071068E-01     POINT(  8)= 3.826834E-01
   POINT(  9)= 0.000000E+00     POINT( 10)=-3.826834E-01
   POINT( 11)=-7.071068E-01     POINT( 12)=-9.238795E-01
   POINT( 13)=-1.000000E+00     POINT( 14)=-9.238795E-01
   POINT( 15)=-7.071068E-01     POINT( 16)=-3.826834E-01



DC TERM= 0
MAX FREQUENCY= 0


FREQUENCY DOMAIN:

              REAL            IMAGINARY

  1       1.875000E-12      -1.000000E+00
  2       0.000000E+00       0.000000E+00
  3       4.783543E-13      -2.404849E-12
  4       0.000000E+00       0.000000E+00
  5      -4.783543E-13       9.515058E-14
  6       0.000000E+00       0.000000E+00
  7      -1.875000E-12      -1.750000E-11
```

EXAMPLE 2

OF DATA POINTS= 16

TIME DOMAIN DATA:

```
   POINT(  1)= 1.000000E+00     POINT(  2)= 1.306600E+00
   POINT(  3)= 1.414200E+00     POINT(  4)= 1.306600E+00
   POINT(  5)= 1.000000E+00     POINT(  6)= 5.412000E-01
   POINT(  7)= 0.000000E+00     POINT(  8)=-5.412000E-01
   POINT(  9)=-1.000000E+00     POINT( 10)=-1.306600E+00
   POINT( 11)=-1.414200E+00     POINT( 12)=-1.306600E+00
   POINT( 13)=-1.000000E+00     POINT( 14)=-5.412000E-01
   POINT( 15)= 0.000000E+00     POINT( 16)= 5.412000E-01
```

DC TERM= 0
MAX FREQUENCY= 0

FREQUENCY DOMAIN:

	REAL	IMAGINARY
1	1.000010E+00	-1.000010E+00
2	0.000000E+00	0.000000E+00
3	-1.339454E-06	-1.339453E-06
4	0.000000E+00	0.000000E+00
5	6.134476E-06	-6.134478E-06
6	0.000000E+00	0.000000E+00
7	-1.502235E-05	-1.502235E-05

```
(# OF COEFFICIENT PAIRS+2)+2= 16


DC TERM= 0
MAX FREQUENCY TERM= 0


FREQUENCY DOMAIN DATA:

    REAL(  1)= 1.000000E+00      IMAG(  1)=-1.000000E+00
    REAL(  2)= 0.000000E+00      IMAG(  2)= 0.000000E+00
    REAL(  3)=-1.339500E-06      IMAG(  3)=-1.339500E-06
    REAL(  4)= 0.000000E+00      IMAG(  4)= 0.000000E+00
    REAL(  5)= 6.134500E-06      IMAG(  5)=-6.134500E-06
    REAL(  6)= 0.000000E+00      IMAG(  6)= 0.000000E+00
    REAL(  7)=-1.502200E-05      IMAG(  7)=-1.502200E-05



TIME DOMAIN:

    DATA POINT(  1)= 9.999898E-01     DATA POINT(  2)= 1.306587E+00
    DATA POINT(  3)= 1.414186E+00     DATA POINT(  4)= 1.306587E+00
    DATA POINT(  5)= 9.999898E-01     DATA POINT(  6)= 5.411945E-01
    DATA POINT(  7)= 2.000000E-12     DATA POINT(  8)=-5.411945E-01
    DATA POINT(  9)=-9.999898E-01     DATA POINT( 10)=-1.306587E+00
    DATA POINT( 11)=-1.414186E+00     DATA POINT( 12)=-1.306587E+00
    DATA POINT( 13)=-9.999898E-01     DATA POINT( 14)=-5.411945E-01
    DATA POINT( 15)=-2.000000E-12     DATA POINT( 16)= 5.411945E-01
```