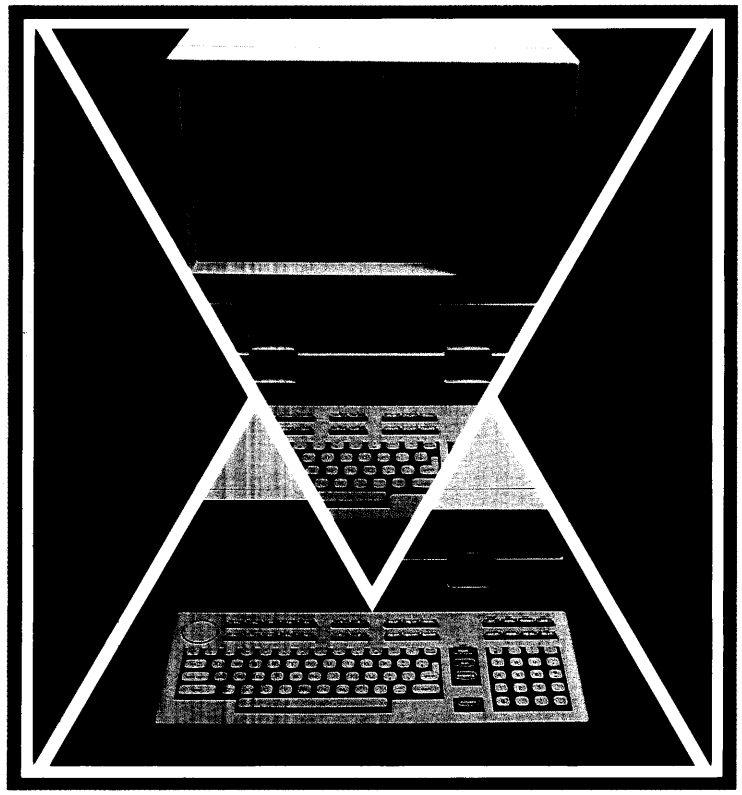
HP 98821A
BASIC Numerical Analysis Library

for the HP 9826 and 9836 Computers



HEWLETT
PACKARD

HEWLETT PACKARD

Warranty Statement

Hewlett-Packard makes no expressed or implied warranty of any kind, including, but not limited to, the implied warranties of merchantibility and fitness for a particular purpose, with regard to the program material contained herein. Hewlett-Packard shall not be liable for incidental or consequential damages in connection with, or arising out of, the furnishing, performance or use of this program material.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warrant that the operation of the CPU, software, or firmware will be uninterrupted or error free.

Use of this manual and tape cartridge (or flexible disc) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form, or with alterations, is expressly prohibited.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Software clause in DAR 7-104.9(a).

HP 98821A
BASIC Numerical Analysis Library

for the HP 9826 and 9836 Computers

Manual Part No. 98821-13111
Flexible Disc No. 98821-13114
98821-13115

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

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

April 1982...First Edition

Table of Contents

Chapter 5: Eigen Analysis

Chapter 6: Interpolation

Chapter 7: Functions

Chapter 8: Fourier Analysis

Introduction

Description

The Numerical Analysis Library provides you with quick access to 57 commonly used numerical analysis routines. These routines are in subprogram form. They can be called up from the library as you need them and appended to your program in memory. Except for error messages, these subprograms contain no input or output operations. Drivers are provided with each subprogram to illustrate some of the techniques for calling the subprogram.

Each section of this manual contains several techniques, giving you the flexibility to choose the most appropriate routine for your specific need.

System Configuration

To use the Numerical Analysis Library, you need:

- a 9826 or 9836 computer with a BASIC language system.
- Numerical Analysis software—part number 98821A. This includes a user's manual and two flexible discs containing the programs.

Using the Numerical Analysis Routines

Drivers and subprograms are contained in mass storage files with the same names. Driver file names have all capital letters and subprogram file names are initially capped followed by lower case letters. For example, "MULLER" contains the driver for subprogram Muller contained in file "Muller".

To get a particular numerical analysis routine, insert the appropriate Numerical Analysis flexible disc into the disc drive with the machine turned on and the BASIC language system loaded. Either select a suitable routine, turn to its user instructions in this manual, and follow them, or write your own "driver" program, including the statement LOADSUB ALL FROM "<filename>" where <filename> is the appropriate numerical analysis subprogram to be used by your program and execute it. If you accidently "loadsub" the wrong file and if that file has any subprograms (either SUB or DEF FN segments), those subprograms will be loaded, right or wrong. If you "loadsub" a file that has been "SAVE'd" (or "RE-SAVE'd") as opposed to "STORE'd" (or "RE-STORE'd") you will get an error 58. If you "loadsub" a file that contains no subprograms, no error will appear and nothing will be loaded.

Using the Numerical Analysis Drivers

The driver programs allow you to use the routines without writing a main program. To use those drivers, type: LOAD "<file name containing driver>", press EXECUTE and RUN. Necessary subprograms will be automatically "loaded", and the program will ask you for your entries.

Redefining User Functions

Some programs (Root Finders, Integration, Differential Equations) need a user function (contained in files "KRYWEN" or "FUNC".) You have to STORE those files before "loading" and "running" the driver. However, if you want to check or redefine your function after you have LOADed the driver and started to RUN it:

1. PAUSE the current program.
2. Type EDIT F and press EXECUTE.
3. EDIT mode displays the program line containing your function.
4. Press ENTER after modifying the program line.
5. Press RUN to restart the program.

Chapter 1
Root Finders

Introduction

Description

This section contains four different algorithms for finding roots of equations. 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, you may find one that will most adequately meet your needs.

Routines

- Bisect - an iterative rootfinder using the bisection method.
- Roota - a general rootfinder using a modified secant method.
- Work - finds the root of a user-defined real-valued continuous function by a modified secant method; contained in file "Roota".
- Chkit - checks if a root has been found or bracketed; contained in file "Roota".
- Monton - checks the three points, (x_1, y_1), (x_2, y_2) and (x_3, y_3), to see that $y = f(x)$ is strictly increasing or decreasing; contained in file "Roota".
- FN Incres - used to check if too many marching steps have been taken in searching for a change of sign; contained in file "Roota".
- Muller - a general rootfinder using a quadratic method.
- Siljak - a polynomial rootfinder with complex coefficients.

Bisection Method

Description

This subprogram will search for solutions of f(x) = 0 over an interval [a,b] where you define the continuous real-valued function f(x). The function may be algebraic of the form ($a_0 + a_1x^{e_1} + a_2x^{e_2} + \text{---} + a_nx^{e_n}$) with a_i real and e_i rational (e.g., $x^3 + 3x^{3/2} + x$) or transcendental (e.g., sin(x) + cos (x)).

You must specify the initial domain value, search increment, error tolerance, and maximum interval-halvings to be used.

Program Usage

Driver Utilization

- File Name - "BISECT", disc 1

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

Subprogram Utilization

- File Name - "Bisect", disc 1
- Calling Syntax

  CALL Biset (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.

- 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.

- Subprograms Required

FN F(X)	Given a domain value x, this function subprogram should return the range value of the function you supply.

Special Considerations and Programming Hints

- 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.
- 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.
- 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 bisection 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 subprogram is to be called a number of times.
- To define the function f(x):

1. PAUSE the current program (if one is running).
2. Type SCRATCH and press EXECUTE.
3. In EDIT mode, type in the following function subprogram pressing ENTER after each line:

```
10   DEF FNF(X)
20 F:   F=<user-defined function of the form (X-3)*(X+4)>
30      RETURN F
40   FNEND
```

4. RE-STORE the function subprogram on file "KRYWEN".
5. LOAD file "BISECT" and press RUN.

(See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the function f(x).)

- 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 =
```

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

- 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 BISECTIONS REACHED ON ROOT #<n>".
  "X BETWEEN <left endpoint> AND <right endpoint>"
  "F(X) = <y>"
  "ACCURACY TO (<right endpoint> − <left endpoint>)"
  "AVERAGE VALUE STORED IN Root(*) AS APPROXIMATE X"

  The subprogram will then move to the next search interval.

Methods and Formulae

The subprogram Bisect will search for solutions of f(x) = 0 over an interval [a,b] where you define the real-valued continuous function f(x). The function may be algebraic of the form $a_0 + a_1x^{e_1} + a_2x^{e_2} + \text{---} + a_nx^{e_n}$ with a_i real and e_i rational (e.g., $x^3 + 3x^{3/2} + x$) or transcendental (e.g., sin(x) + cos(x)).

You must specify 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 + Δx]. If the functional values are of opposite sign then the bisection method is used to locate the root. Each subinterval [a + iΔx, a + (i + 1)Δx] 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. You must also specify a maximum number of interval-halvings or bisections (Maxbi), so that an error tolerance that is not satisfied will result in the root localized to an interval of size $2^{-\text{Maxbi}}(b-a)$. The subprogram will examine $N = \text{INT}\ \frac{b-a}{\Delta x}$ intervals.

Reference

1. Stark, Peter A., Introduction to Numerical Methods, London: MacMillan Company, Collier-MacMillan Limited, 1970, pp. 95-96.

User Instructions

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "BISECT" and press RUN.
3. You will be asked to supply entries for the following items:
   - number of roots to be found
   - lower bound of search interval
   - upper bound of search interval
   - maximum number of bisections for searching for any one root in a subinterval
   - error tolerance where $|f(x)|<Tol*(1\ MAX\ x)$ is the tolerance criterion
   - search increment or Δx.

   Press CONTINUE after each entry.
4. 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

User-defined function:

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

User entries:

```
# OF ROOTS TO BE FOUND= 4
LOWER BOUND= 0
UPPER BOUND= 10
MAXIMUM # OF BISECTIONS= 20
ERROR TOLERANCE= 1.E-8
SEARCH INCREMENT= .1
```

Results:

ROOT	FUNCTION VALUE	ACCURACY
6.238996E-01	-9.442352E-09	1.907349E-07
3.228892E+00	1.620545E-08	1.907349E-07
6.307698E+00	-1.632362E-08	1.907349E-07
9.435884E+00	8.958369E-08	3.051758E-06

Modified Secant Method

Description

Given a first guess, this subprogram will search for a solution of f(x) = 0 where you define 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.

You are 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.

Program Usage

Driver Utilization

- File Name - "ROOTA", disc 1

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

Subprogram Utilization

- File Name - "Roota", disc 1
- Calling Syntax

  CALL Roota (Root, Err, Frsges, Step, Istpmx, Itrmx, 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 $\|f(x)\| \leq \text{Tolf}$.

- Output Parameters

Root	Root of f(x)
Err	f(Root)

- Subprograms Required

Chkit	Checks if a root is found or bracketed.
FN F(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 endpoints of the interval be of opposite sign.

Further explanation of these subprograms is given on pages 1-8 and 1-10 through 1-15.

Special Considerations and Programming Hints

- 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 = Istpmx = Itrmx =
  Tola = Tolf =

  The data may be corrected from the keyboard (e.g., Tola = 1E−6, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.
- 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.
- 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. You may want to restart the program and make these changes.

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

- To define the function f(x):

  1. PAUSE the current program (if one is running).
  2. Type SCRATCH and press EXECUTE.
  3. In EDIT mode, type in the following function subprogram pressing ENTER after each line:

```
10    DEF FNF(X)
20 F:    F=<user-defined function of the form (X-3)*(X+4)>
30       RETURN F
40    FNEND
```

  4. RE-STORE the function subprogram on file "KRYWEN".
  5. LOAD file "ROOTA" and press RUN.

(See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the function f(x).)

Methods and Formulae

Subprogram Roota begins by initializing X2 = Frsges. Then subprogram Chkit is called. In Chkit a new x is generated and $|f(x)| \leq$ 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(x) does not change sign, then a test is made to see which way to march:

$(y_2 - y_1) * y_1 < 0$, then march right
$(y_2 - y_1) * y_1 > 0$, 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

$|f(x_2)| \leq$ Tolf

or

$|x_1 - x_2| \leq (1 \text{ MAX } (|x_1| \text{ MAX } |x_2|)) * \text{Tola}$

where x_1, x_2 are consecutive attempts at finding a root.

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

User Instructions

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "ROOTA" and press RUN.
3. You will be asked to supply entries for the following items:
   - first guess for the root
   - step size (a positive value) for marching when looking for an interval in which the function changes sign
   - maximum number of steps allowed in marching to find an interval in which the function changes sign
   - maximum number of iterations allowed after an interval has been found in which the function changes sign
   - tolerance for 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
     $|x_1 - x_2| \leq (1 \text{ MAX } (|x_1| \text{ MAX } |x_2|)) * <\text{tolerance for root}>$
   - tolerance for function evaluation where an approximation x is accepted as a root if $|f(x)| \leq <\text{tolerance for the function}>$.

   Press CONTINUE after each entry.
4. The root x as well as the functional value f(x) will be printed.

Example

User-defined function:

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

User entries:

```
FIRST GUESS= 1
STEP SIZE= .25
MAXIMUM NUMBER OF STEPS= 30
MAXIMUM NUMBER OF ITERATIONS= 30
TOLERANCE FOR ROOT= 1.E-8
TOLERANCE FOR FUNCTION= 1.E-6
```

Results:

```
ROOT= 1.763223E+00    FUNCTION VALUE=-5.031967E-08
```

Subprogram 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", disc 1
- Calling Syntax
  CALL Work (Root, Err, T1, Z1, T3, Z3, Itrmx,Tola, Tolf)
- 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.

- Output Parameters

Root	Root of f(x)
Err	f(Root)

- Subprograms Required

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

Special Considerations and Programming Hints

- Upon entry into the subprogram, Z1 and Z3 must be of opposite sign. If not, an error message is printed and the program pauses.

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

  You may want to restart the program and correct the data.

- 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 you should check both the logic in the calling program and the specific behavior of the function in the interval [T1, T3].

- 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 =

  The maximum number of iterations may be corrected from the keyboard (e.g., Itrmx = <new maximum>, EXECUTE). When CONTINUE is pressed, 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. You 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.

- 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 "Methods and Formulae" section for subprogram Roota on page 1-8.

Subprogram 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", disc 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 a positive value.
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 a root was located. Rtn = − 1 implies a root was not yet located.

- Output Parameters

Root	Root of f(x)
Err	f(Root)

- Subprograms Required

FN F(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.

Special Considerations and Programming Hints
See subprogram Roota (page 1-7 through 1-8).

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 subporgram 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 Monton

This subprogram 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", disc 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

- Upon entering this subprogram, a check is made to see that X1<X2<X3. If not, an error message is printed and the program pauses.

  "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. You may want to redefine the function (see page 1-8).

- Monton's primary purpose is to check that the function is monotonic in the interval under consideration; i.e. $x_1 < x_2 < x_3$ 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 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 =
  X1 = 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. You may want to restart the program and make these changes.

Methods and Formulae

Monton performs two checks:

1. that $x_1 < x_2 < x_3$
   and
2. that $(y_3 - y_2) * (y_2 - y_1) \geq$ 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 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", disc 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 $[x_1, x_3]$
Y1, Y2, Y3	Y1 = f(X1), Y2 = f(X2), Y3 = f(X3)

- Output

FN Incres	I + 1

Special Considerations and Programming Hints

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

  "ERROR IN SUBPROGRAM Incres."
  "NO CHANGE OF SIGN AFTER <i> STEPS"
  X1 = Y1 =
  X2 = Y2 =
  X3 = Y3 =

  If you wish to continue with a larger maximum number of steps, Istpmx may be changed from the keyboard (e.g., Istpmx = <new maximum>, EXECUTE). Press CONTINUE and the program will resume with the new maximum number of steps. If Istpmx = 20 does not work, you may want to attempt Istpmx = 50. If this is not effective, it would not make sense to try Istpmx = 100 or Istpmx = 1000. There is some other difficulty.

Muller's Method

Description

Given a vector of initial guesses, this subprogram will search for solutions of f(x) = 0 where you define the continuous real-valued function f(x). Roots are found by Muller's method, a real, quadratic root solver.

You are 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.

Program Usage

Driver Utilization

- File Name - "MULLER", disc 1

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

Subprogram Utilization

- File Name - "Muller", disc 1
- Calling Syntax

  CALL Muller (Root(*), Nroots, Itmax, Tolf, Eps, Eta, Digits)
- Input Paramters

Root(*)[1]	Vector containing initial guesses of roots.
Nroots	Number of roots to be found.
Itmax[1]	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.

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

- Output Parameters

Root(*)[1]	Vector containing roots of the function.
Itmax[1]	Number of iterations required to find final root.

- Subprograms Required

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

Special Considerations and Programming Hints

- Upon entry into subprogram Muller, 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 Muller."
  Nroots = Itmax =
  Tolf = Digits =

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

- 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>"

  Root(I) is assigned the value 9.999999E99. The subprogram continues execution with a search for the next root.

- To define the function f(x):

  1. PAUSE the current program (if one is running).
  2. Type SCRATCH and press EXECUTE.
  3. In EDIT mode, type in the following function subprogram pressing ENTER after each line:

```
10   DEF FNF(X)
20 F:   F=<user-defined function of the form (X-3)*(X+4)>
30      RETURN F
40   FNEND
```

  4. RE-STORE the function subprogram on file "KRYWEN".
  5. LOAD file "MULLER" and press RUN.

(See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the function f(x).)

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

  However, there are times this method gives misleading results. For example, if the first fifty positive roots of a particular function are desired and you do 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:

Example

User-defined function:

```
F = SIN(X)
```

User entries:

```
# OF ROOTS TO BE FOUND= 6
MAXIMUM # OF ITERATIONS= 20
TOLERANCE FOR FUNCTION= 1.E-6
SPREAD TOLERANCE FOR MULTIPLE ROOTS= 1.E-8
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
```

Results:

```
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:

User-defined function:

```
F = SIN(X)
```

User entries:

```
# OF ROOTS TO BE FOUND= 6
MAXIMUM # OF ITERATIONS= 20
TOLERANCE FOR FUNCTION= 1.E-6
SPREAD TOLERANCE FOR MULTIPLE ROOTS= 1.E-8
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
```

Results:

```
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 $f(x) = 0$ where you define 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 that you supply an initial guess, Root(I),

$P = .9 * \text{Root(I)}$
$P_1 = 1.1 * \text{Root(I)}$
$P_2 = \text{Root(I)}$ if $\text{Root(I)} \neq 0$.

If Root(I) = 0, $P = -1$, $P_1 = 1$, and $P_2 = 0$. From these three starting values, you choose 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 $d_{i+3} = \dfrac{-2 * F(Rt_{i+2}) * (1 + d_{i+2})}{b_{i+2} \pm b^2_{i+2} - 4 * F(Rt_{i+2}) * d_{i+2} * (1 + d_{i+2}) * \{F(Rt_i) * d_{i+2} - F(Rt_{i+1}) * (1 + d_{i+2}) + F(Rt_{i+2})\}^{1/2}}$

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

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

Having found (R − 1) roots, you determine the Rth root by solving the equation $f_R(x) = 0$ where:

$$f_R(x) = \frac{f(x)}{\prod_{i=1}^{R-1} (x - x_i)} \qquad (R = 2,3,\ldots).$$

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

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

or

$|f(Rt)| < \text{Tolf}$ and $|f(Rt)| < \text{Tolf}$ 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), p.602.

User Instructions

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "MULLER" and press RUN.
3. You will be asked to supply entries for the following items:
   - number of roots to be found
   - maximum number of iterations permitted per root
   - tolerance for functional evaluation where x is accepted as a root of $|f(x)| \leq$ <this tolerance>
   - 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>.
   - restart value for multiple roots
   - number of significant digits in root for it to be acceptable
   - initial guesses for the number of roots expected.

   Press CONTINUE after each entry.
4. Values will be printed for all roots found. The value 9.999999E99 is inserted for any roots not found.

Example

User-defined function:

$$F = SIN(X)$$

User entries:

```
# OF ROOTS TO BE FOUND= 4
MAXIMUM # OF ITERATIONS= 20
TOLERANCE FOR FUNCTION= 1.E-8
SPREAD TOLERANCE FOR MULTIPLE ROOTS= 1 E-10
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
```

Results:

```
ROOTS:

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

Siljak's Method

Description

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

$$a_0 + ib_0 + (a_1 + ib_1)Z + (a_2 + ib_2)Z^2 + \text{---} + (a_n + ib_n)Z^n = 0.$$

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

You are required to enter the complex coefficients of the polynomial as well as its degree. Tolerances for the root and for function evaluations and the maximum number of iterations also need to be entered.

Program Usage

Driver Utilization

- File Name - "SILJAK", disc 1

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

Subprogram Utilization

- File Name - "Siljak", disc 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 $a_0 + ib_0 + (a_1 + ib_1)Z + (a_2 + ib_2)Z^2 + \text{---} + (a_n + ib_n)Z^n = 0$, 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 0 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

- 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 = Itmax =

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

- The subprogram has been set to quarter the interval size at most 20 times. If this maximum is exceeded, an error message is printed:

  "ERROR IN SUBPROGRAM Siljak."
  "THE INTERVAL SIZE HAS BEEN QUARTERED 20 TIMES AND THE TOLERANCE FOR FUNCTIONAL EVALUATIONS IS STILL NOT MET."
  Tolf = U = V =

  When CONTINUE is pressed the subprogram is 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.

- If the maximum number of iterations has been exceeded, the following error message will be printed and Siljak will pause:

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

  When CONTINUE is pressed, the subprogram will be exited. 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

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 you are 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.

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

Siljak functions x_k and y_k are defined by $Z^k = x_k + iy_k$ and may be calculated recursively where $x + iy$ are the root approximations.

$$x_0 = 1,\ x_1 = .1,\ y_0 = 0,\ y_1 = 1$$

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

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

$$\mu = \sum_{k=0}^{n} (a_k x_k - b_k y_k)$$

$$\frac{\partial \mu}{\partial x} = \sum_{k=1}^{n} k(a_k x_{k-1} - b_k y_{k-1})$$

$$\frac{\partial v}{\partial x} = \sum_{k=1}^{n} k(a_k y_{k-1} + b_k x_{k-1})$$

Reference

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.

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "SILJAK" and press RUN.
2. You will be asked to supply entries for the following items:
   - degree of polynomial (this is also the number of roots that will be found)
   - maximum number of iterations allowed per root
   - tolerance for roots where a root is accepted if the difference in value of the root approximations of two successive iterations is less than this tolerance
   - tolerance for functional evaluations where the root x is accepted if $|f(x)| \leq$ <this tolerance>
   - the real part and the imaginary part for each coefficient of the polynomial (the array subscript shown corresponds to the exponent of the variable).

   Press CONTINUE after each entry.
3. The real and imaginary parts of the roots will be printed. Any roots not found will contain the value 9.999999E99.

Example

User entries:

```
DEGREE OF POLYNOMIAL= 4
MAX # OF ITERATIONS= 20
TOLERANCE FOR ROOTS= 1.E-8
TOLERANCE FOR FUNCTIONAL EVALUATIONS= 1.E-6


COEFFICIENTS: I(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
```

Results:

```
ROOTS:
        REAL                IMAGINARY


 -2.000000E+00         -1.747497E-24
  2.000000E+00          1.747497E-24
  1.748064E-24         -2.000000E+00
 -1.748065E-24          2.000000E+00
```

Chapter 2
Integration

Introduction

Description

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.

Routines

- Simp - the easiest method to use for well-behaved functions.
- Filon - for functions of the form f(x)sin(x) or f(x)cos(x).
- Cadre - used for a great many functions when other methods fail.
- Inteq - numerical integration with equally-spaced data points.

Simpson's Rule

Description

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

Program Usage

Driver Utilization

- File Name - "SIMP",disc 1

The driver "SIMP" sets up the necessary input parameters for the subprogram Simp and prints out the resulting value of the integral. Intermediate integral values may also be printed if you wish.

Subprogram Utilization

- File Name - "Simp", disc 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.

- Output Parameters

Int	Value of integral.

- Subprograms Required

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

Special Considerations and Programming Hints

- Upon entry into the subprogram, a bad data check is performed. If the program detects "nonsense" data, the program 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 CONTINUE is pressed, the program will resume execution at the next line.

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

  "ERROR IN SUBPROGRAM Simp."
  "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. If the data is corrected from the keyboard, pressing CONTINUE will cause execution to resume.

- To define the function f(x):

  1. PAUSE the current program (if one is running).
  2. Type SCRATCH and press EXECUTE.
  3. In EDIT mode, type in the following function subprogram pressing ENTER after each line.

     ```
     10   DEF FNF(X)
     20 F:   F=<user-defined function of the form (X-3)*(X+4)>
     30      RETURN F
     40   FNEND
     ```

  4. RE-STORE the function subprogram on file "KRYWEN".
  5. LOAD file "SIMP" and press RUN.

  (See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the function f(x).)

Methods and Formulae

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

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,

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

Reference

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

User Instructions

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "SIMP" and press RUN.
3. You will be asked to supply entries for the following items:
   - lower bound of the interval
   - upper bound of the interval
   - maximum number of interval halvings
   - error tolerance where the value of the integral is accepted if the difference in value of two successive approximations is less than this tolerance.

   Press CONTINUE after each entry.
4. The program will print the value of the integral as well as intermediate data points if they were requested.

Example

User-defined function:

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

User entries:

```
LOWER BOUND= 0
UPPER BOUND= 1.04719755512
MAX # OF INTERVAL HALVINGS= 10
ERROR TOLERANCE= .0001
```

Intermediate data points:

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

Results:

```
INTEGRAL= 2.173921E-01
```

Filon's Method

Description

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.

You are required to specify an array of function values, the number of points of evaluation, the number of oscillations (t), and the endpoints of the interval.

Program Usage

Driver Utilization

- File Name - "FILON", disc 1

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

Subprogram Utilization

- File Name - "Filon", disc 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 cosine integral is evaluated. Key = − 1 implies sine integral is evaluated.

- Output Parameter

Int	Value of the integral.

- Subprograms Required

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

Special Considerations and Programming Hints

- 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 Filon."
  T = A = B =
  Nb = Key =

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

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- For a good approximation of the integral, Theta values (Theta = t*h) should be kept less than 1. So, for example, if you want to integrate $\int_0^1 x^2 \sin(10x)dx$, t = 10. Nb should be chosen greater than 11 since:

$$10h < 1,\ h < \frac{1}{10},\ \frac{(b-a)}{Nb-1} < \frac{1}{10},\ \frac{1}{Nb-1} < \frac{1}{10},\ Nb > 11.$$

  Care should also be taken not to choose too large an Nb since this increases the running time of the program.

- 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.
- To define the function f(x) where f(x)sin tx or f(x)cos tx is to be integrated:
  1. PAUSE the current program (if one is running).
  2. Type SCRATCH and press EXECUTE.
  3. In EDIT mode, type in the following function subprogram pressing ENTER after each line.

     ```
     10   DEF FNF(X)
     20 F:   F=<user-defined function of the form (X-3) * (X+4)>
     30      RETURN F
     40   FNEND
     ```

  4. RE-STORE the function subprogram on file "KRYWEN".
  5. LOAD file "FILON" and press RUN.

  (See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the function f(x).)

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

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

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

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

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

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

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

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

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

User Instructions

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "FILON" and press RUN.
3. You will be asked to supply entries for the following items:
   - number of points of integration (odd number required)
   - form of integrand, whether f(x)sin(tx) or f(x)cos(tx)
   - lower bound of the interval
   - upper bound of the interval
   - frequency or number of oscillations of the trigonometric function (must be greater than 0)

   Press CONTINUE after each entry.
4. The program will print the value of the integral.

Examples

User-defined function:

$$F = X*X$$

User entries:

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

Result:

```
INTEGRAL= 4.674011E-01
```

User-defined function:

$$F = LOG(1 + X)$$

User entries:

```
# OF POINTS OF INTEGRATION= 31
Key=-1
LOWER BOUND= 0
UPPER BOUND= 6.28318530 72
FREQUENCY= 10
```

Result:

```
INTEGRAL=-1.976604E-01
```

Using the same function ($F = LOG(1 + X)$):

User entries:

```
# OF POINTS OF INTEGRATION= 71
Key=-1
LOWER BOUND= 0
UPPER BOUND= 6.28318530 72
FREQUENCY= 10
```

Results:

```
INTEGRAL=-1.976264E-01
```

Cautious Adaptive Romberg Extrapolation

Description

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].

You are required to specify the lower and upper bounds, the relative and absolute error tolerances and the user-defined function subprogram.

Program Usage

Driver Utilization

- File Name - "CADRE", disc 1

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

Subprogram Utilization

- File Name - "Cadre", disc 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 (0,0.1); e.g., Rerr = 0.1 indicates that the estimate of the integral is to be correct to one digit, whereas Rerr = 1E − 4 calls for four digits of accuracy.

- 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. Flg = 1, all is well Flg = 2, one or more singularities were successfully handled Flg = 3, in some subintervals, the estimate Vint was accepted merely because Err was small even though no regular behavior could be recognized. Flg = 4, failure, overflow of stack Ts. Flg = 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.

- Subprograms Required

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

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- Cadre can, in many cases, handle jump discontinuities and certain algebraic discontinuities. See reference 1 for full details.
- The relative error parameter Rerr must be in the interval [0,0.1]. For example, Rerr = 0.1 indicates that the estimate of the integral is to be correct to one digit, whereas Rerr = 1E − 4 calls for four digits of accuracy.
- The absolute error parameter, Aerr, should be nonnegative. In order to give a reasonable value for Aerr, you must know the approximate magnitude of the integral being computed. In many cases, it is satisfactory to use Aerr = 0. In this case, only the relative error requirement is satisfied in the computation.
- 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.
- 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."

- To define the function f(x):

  1. PAUSE the current program (if one is running).
  2. Type SCRATCH and press EXECUTE.
  3. In EDIT mode, type in the following function subprogram pressing ENTER after each line:

     ```
     10   DEF FNF(X)
     20 F:   F=<user-defined function of the form (X-3)*(X+4)>
     30      RETURN F
     40   FNEND
     ```

  4. RE-STORE the function on subprogram on file"KRYWEN".
  5. LOAD file "CADRE" and press RUN.

  (See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the function f(x).)

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

$$| \int_a^b f(x)dx - \text{Int} | \leq = \text{MAX}(\text{Aerr}, \text{Rerr} * | \int_a^b f(x)dx |).$$

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 cautious 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.

Reference

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

User Instruction

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "CADRE" and press RUN.
3. You will be asked to supply entries for the following items:
   - lower bound of the interval of integration
   - upper bound of the interval of integration
   - absolute error tolerance
   - relative error tolerance.

   Press CONTINUE after each entry.
4. The program will print the integral value, the estimated error term, and the suggested reliability of the results.

Examples

User-defined function:

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

User entries:

```
LOWER BOUND= 0
UPPER BOUND= 1.0471975512
ABSOLUTE ERROR= 1.E-10
RELATIVE ERROR= 1.E-8
```

Results:

```
INTEGRAL= 2.173928E-01
ESTIMATED ERROR= 4.557823E-11
FLG=1
```

User-defined function:

```
F = LOG(EXP(1)/X)
```

User entries:

```
LOWER BOUND= 1.E-11
UPPER BOUND= 1
ABSOLUTE ERROR= 1.E-8
RELATIVE ERROR= 1.E-6
```

Results:

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

Numerical Integration with Equally-Spaced Data Points

Description

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.

You are 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.

Program Usage

Driver Utilization

- File Name - "INTEQ", disc 1

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

Subprogram Utilization

- File Name - "Inteq", disc 1
- Calling Syntax
  CALL Inteq (N, Inc, F(*), Int)
- Input Paramters

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

- 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 Inteq."
  "ODD # OF DATA POINTS IS REQUIRED."
  N =

  The number of data points can be corrected from the keyboard (e.g., N = <new number of data points>, EXECUTE). When CONTINUE is pressed, the program will continue execution at the next line.

- 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.

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

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

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

Reference

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

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "INTEQ" and press RUN.
2. You will be asked to supply entries to the following items:
   - number of data points (odd number required)
   - increment between data points
   - appropriate function value of each data point.

   Press CONTINUE after each entry.
3. The program will print the value of the integral.

Examples

User entries:

```
# 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
```

Result:

```
INTEGRAL= 5.000000E-01
```

User entries:

```
# 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
```

Result:

```
INTEGRAL= 1.541667E+01
```

Numerical Integration with Unequally-Spaced Data Points

Description

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]. You are required to input the data points and the tolerance desired.

Program Usage

Driver Utilization

- File Name - "INTUN", disc 1

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

Subprogram Utilization

- File Name - "Intun", disc 1
- 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 of the data points subscripted from 1 to N.

- Output Parameters

Int	Value of the integral.

Special Considerations and Programming Hints

- 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 = Esp =
  X_(1) = X_(N) =

  The data may be corrected from the keyboard (e.g., N = 21, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.
- 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.
- 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.
- The number of data points must be greater than 2, otherwise an error will occur in the dimension statement in line XXX 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 $s''(x_1) = s''(x_n) = 0$.

The procedure to determine s(x) involves the iterative solution of a set of simultaneous linear equations by the method of successive overrelaxation. You can specify accuracy to which these equations are solved. 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., Ed., "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.

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "INTUN" and press RUN.
2. You will be asked to supply entries to the following items:
   - number of data points to be used in the computation
   - error tolerance desired in solving the system of equations
   - x (in increasing order) and y coordinate of each data point.

   Press CONTINUE after each entry.
3. The program will then print the endpoints of the interval of integration and the value of the integral.

Examples

User entries:

```
# OF DATA POINTS= 5
ERROR TOLERANCE= 1.E-6

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
```

Result:

```
INTEGRAL FROM  0.000 TO  4.000 IS  8.000000E+00
```

User entries:

```
# OF DATA POINTS= 8
ERROR TOLERANCE= 1.E-6

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
```

Result:

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

Chapter 3
Ordinary Differential Equations

Introduction

Description

A differential equation is a relation among an independent variable and a dependent variable and its derivatives. For example, $y^1 + xy^2 = 0$. The solution is a function y of x satisfying this equation and passing through a given point, (x_0,y_0).

This section contains two methods for solving systems of first order differential equations. (Higher order equations may be solved by reducing to a system of first order equations.)

Routines

- Kutta - a good, stable method when accuracy requirements are not high or when the equation does not involve a large number of calculations.
- Adams - a method which is more accurate and requires fewer computations but may not be as stable in some cases.

Runge-Kutta Method

Description

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

You 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, you must also 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".

Program Usage

Driver Utilization

- File Name - "KUTTA", disc 1

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

Note

The driver prints every tenth value of the computed vectors. If a different number of values is required, line 420 may be changed, replacing the 10 by an appropriate number.

Subprogram Utilization

- File Name - "Kutta", disc 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.

- 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.

- Subprograms Required

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.

Special Considerations and Programming Hints

- 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."
A =          B =
H =          Maxstp =
```

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

- 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 Adams predictor-corrector method.
- 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.
- 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.
- To define the system of equations to be solved:

1. PAUSE the current program (if one is running).
2. Type SCRATCH and press EXECUTE.
3. In EDIT mode, type in the subprogram Func, pressing ENTER after each line. The Subprogram Utilization section explains the parameters used and the examples following explain how to set up a system of equations in the subprogram.
4. RE-STORE the subprogram on file "Func".
5. LOAD file "KUTTA" and press RUN.

(See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the system of equations.)

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

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

with the given initial conditions $y(x_0), \frac{dy}{dx}(x_0), \ldots, \frac{d^{m-1}y}{dx^{m-1}}(x_0)$

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 $y'' - (.1)(1 - y^2)y' + y = 0$ with initial conditions $y(0) = 1$, $y'(0) = 0$.
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: F(1) = Ysv(2)
F(2) = (.1)*(1 - Ysv(1)*Ysv(1))*Ysv(2) - Ysv(1)
SUBEND
```

Example 2.

Compute the solution of $y'' + xy^2 = 0$ with initial conditions $y(0) = 0$, $y'(0) = 1$.

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: F(1) = Ysv(2)
F(2) = X*Ysv(1)*Ysv(1)
SUBEND
```

Example 3.
Compute the solution of

$y''' + 17y'' - 10y' + y = 0$ with initial conditions $y(0) = y'(0) = y''(0) = 0$.

Let $Z = y'$, then $Z' = y''$.
Let $W = Z' = y''$, then $W' = Z'' = y''' = -17y'' + 10y' - y$.

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: F(1) = Ysv(2)
F(2) = Ysv(3)
F(3) = 17*Ysv(3) + 10*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 you specifying the initial conditions, x_0, $y_1(x_0)$, $y_2(x_0)$, ..., $y_n(x_0)$.

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

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

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 $n = 0, 1, 2, \ldots$ 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.

User Instructions

1. If you have not already defined the system of equations you wish to solve, you should do so at this time and save on file "Func".
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "KUTTA" and press RUN.
3. You will be asked to supply entries for the following items:
   - dimension of the system of differential equations
   - lower bound of the integral
   - upper bound of the integral
   - step size for integrating
   - maximum number of iteration steps permitted
   - initial values for y for each dimension of the system.
4. The program will print the domain values x as well as the values of the integrated function at x.

Examples

- Solve $y' - xy^{1/3} = 0$ for $1 \leq x \leq 3$, step size = 0.01 subject to the initial condition $y(1) = 1$. Print every tenth value. With $y = y_{sv}(1)$, subprogram Func is set up as

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

User entries:

```
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
```

Results:

X	Y
1.000000E+00	1.000000E+00
1.100000E+00	1.106817E+00
1.200000E+00	1.227380E+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	2.309421E+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

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

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

  Let $Z = y'$, then $Z' = y'' = -2yy'$.
  With $y = Ysv(1)$ and $y' = Ysv(2)$, the subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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

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

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

  Let $Z = y'$, then $Z = y'' = -3(y')^2 y$.
  With $y = Ysv(1)$ and $y' = Ysv(2)$, the subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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

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

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

  Let $Z = y'$, then $Z' = y'' = (.1)(1 - y^2)y' - y$. With $y = Ysv(1)$ and $y' = Ysv(2)$, the subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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	4.916325E-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

Adams-Bashford-Moulton Method

Description

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.

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

Note

Before running this subprogram, you must also 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".

Program Usage

Driver Utilization

- File Name - "ADAMS", disc 1

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

Subprogram Utilization

- File Name - "Adams", disc 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.

- 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.
Er(*)	Error estimate for Y(I,N); Er(*) is subscripted from 1 to Nb where Nb is the number of points.

- Subprograms Required

Kutta (Idm, A, H, B, Maxstp (Ynt(*), Y(*))	Used as a starter for subprogram Adams.
Func (Ysv(*), X, Idm, F(*))	Contains user-defined system of equations; used to generate new function values F(*), from old function values, Ysv(*).

Special Considerations and Programming Hints

- 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."
  A = B =
  H = Crktr = Maxstp =

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

- When functional evaluations are costly, Adams is a good choice since only two evaluations are made per integration step.
- 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.
- 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.
- 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.
- To define the system of equations to be solved:

  1. PAUSE the current program (if one is running).
  2. Type SCRATCH and press EXECUTE.
  3. In EDIT mode, type in the subprogram Func, pressing ENTER after each line. The Subprogram Utilization section explains the parameters used and the examples on pages 3-8 through 3-11 explain how to set up a system of equations in the subprogram.
  4. RE-STORE the subprogram on file "Func".
  5. LOAD file "ADAMS" and press RUN.

  (See "Redefining User Functions" in the Introduction, following the Table of Contents, for another way to modify the system of equations.)

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 you specifying the initial conditions, x_0, $y_1(x_0)$, $y_2(x_0)$, ..., $y_n(x_0)$.

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

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

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 $n = 0, 1, 2, \ldots$ 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 given on page 3-7.

User Instructions

1. If you have not already defined the function you wish to solve, you should do so at this time.
2. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "ADAMS" and press RUN.
3. You will be asked to supply entries for the following items:
   - dimension of the system of differential equations
   - lower bound of the interval under consideration
   - upper bound of the interval under consideration
   - step size or distance between each point of integration
   - maximum number of integration steps permitted
   - initial values for y for each dimension of the system.

   You may also choose to use the Moulton corrector.
   Press CONTINUE after each entry.
4. The program will print the domain values x, as well as the values of the integrated function at x.

Examples

- See example on p. 3-8 for a statement of the problem. Subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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

- See example on p. 3-9 for a statement of the problem. Subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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

- See example on p. 3-10 for a statement of the problem. Subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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

- See example on p. 3-11 for a statement of the problem. Subprogram Func is set up as follows:

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

User entries:

```
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
```

Results:

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

Chapter 4
Linear Algebraic Systems

Introduction

Description

This section contains a wide variety of routines dealing with matrices and systems of simultaneous linear equations. Significant savings in storage are obtained in the matrix inversion routines by taking advantage of special types of matrices, such as positive definite. An additional digit or two of accuracy may also be obtained in many cases. There are also routines for storing symmetric matrices using minimal memory and for transposing a matrix in place, that is, replacing a matrix by its transpose.

Routines

- Ludsht - solves the system of equations Ax = b using triangular decomposition.
- Decomp - computes the triangular decomposition of nonsingular matrix A; contained in file "Ludsht.".
- Solve - finds an approximate solution to a single system of equations, Ax = b; contained in file "Ludsht".
- Posdef - given a symmetric, positive definite matrix, A, stored in symmetric storage mode, will solve the system of equations Ax = b using Cholesky's Method.
- Choles - performs Cholesky decomposition on a symmetric, positive definite matrix stored in symmetric storage mode; contained in file "Posdef".
- Solcho - solves a Cholesky system in symmetric storage mode; contained in file "Posdef".
- Pinver - finds the inverse of a positive definite matrix stored in symmetric storage mode in vector S(*).
- Triang - given a lower triangular matrix stored in symmetric storage mode vector S(*), this routine will multiply S^TS; contained in file "Pinver".
- Invers - finds the inverse of a lower triangular matrix stored in symmetric storage mode vector S(*); contained in file "Pinver".
- Storag - will store a symmetric matrix, A, in a vector, S, in symmetric storage mode, or a vector, S, into a symmetric matrix, A.

Linear Equation Solver-Minimum Storage

Description

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.

Program Usage

Driver Utilization

- File Name - "LUDSHT", disc 1

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

Subprogram Utilization

- File Name - "Ludsht", disc 1
- Calling Syntax
  Call Ludsht (A(*), B(*), N, M)
- Input Parameters

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

- Output Parameters

A(*)[1]	Array containing the Lu decomposition of A(*).
B(*)[1]	Array containing the solution matrix X to the system AX = 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.

Further explanation of these subprograms is given on pages 4-6 through 4-10.

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

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

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

Reference

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

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "LUDSHT" and press RUN.
2. You will be asked to supply entries for the following items:
   - dimensions of the coefficient matrix B (row and column)
   - elements of nonsingular matrix A
   - elements of coefficient matrix B

   Press CONTINUE after each entry.
3. The program will then print the solution matrix X.

Examples

User entries:

```
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  5.000000E+00
-2.000000E+00  3.000000E+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
```

Results:

```
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
```

User entries:

```
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
```

Results:

```
MATRIX X:

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

Subprogram 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

- FileName - contained in file "Ludsht", disc 1
- 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

- 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 = 10, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.

- 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. You may want to restart the program to correct this.

- 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."

  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.

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.

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.

References

1. Golub, G., "Numerical Methods for Solving Linear Least Squares Problems", Numer. Math., Vol. 7 (1965) pp. 206-216.
2. Golub, G. and Kahan, W., "Calculating the Singular Values and Pseudo-inverse of a Matrix", J. SIAM Numerical Analysis Series B, Vol. 2 (1965) pp. 205-224.

Subprogram 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", disc 1
- 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 Lu(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

- 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 = 10, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.

- 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 (page 4-8) 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.

Positive Definite Matrices

Description

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.

Program Usage

Driver Utilization

- File Name - "POSDEF", disc 1

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

Subprogram Utilization

- File Name - "Posdef", disc 1
- Calling Syntax
  CALL Posdef (S(*), B(*), R, C)
- Input Parameters

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

- Output Parameters

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

- Subprograms Required

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

Further explanation of these subprograms is given on pages 4-15 through 4-17.

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

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- Because of the symmetry 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.

- 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(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

The 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.

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "POSDEF" and press RUN.
2. You will be asked to supply entries for the following items:
   - dimensions of coefficient matrix B (row and column)
   - elements of the symmetric storage vector S
   - elements of coefficient matrix B.

   Press CONTINUE after each entry.
3. The program will then print the solution matrix X where columns of X are the solutions to corresponding columns of B.

Examples

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 $AX = B$.

User entries:

```
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
```

Results:

```
MATRIX X:

-1.000000E+00   2.000000E+00   3.000000E+00
```

- 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 $AX = B$.

User entries:

```
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
```

Results:

```
MATRIX X:

 9.000061E+00  -3.600032E+01   3.000030E+01
-3.600032E+01   1.920017E+02  -1.800015E+02
 3.000030E+01  -1.800015E+02   1.800014E+02
```

Subprogram 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", disc 1
- Calling Syntax

  CALL Choles (G(*), R)
- Input Parameters

G(*)[1]	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(*)[1]	Vector containing the lower triangular matrix G where GG^T is the decomposition of the positive definite matrix.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- If the subprogram detects a nonpositive diagonal element, the following error message will be printed and the program pauses:

  "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 you to check values of the variables in the subprogram.

- 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)

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

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

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

FOR I = J + 1 TO N

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

NEXT I

NEXT J

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

Subprogram 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", disc 1
- 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(*)[1]	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(*)[1]	Array containing the solution matrix X of $AX = B$.

Special Considerations and Programming Hints

- 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 Solcho."
  R = C =

  The data may be corrected from the keyboard (e.g., R = 7, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.
- The data stored in the coefficient matrix B is destroyed and replaced by the solution matrix X.
- 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 $G^TX = Y$.

1 B(*) is both an input and output parameter.

Positive Definite Matrix

Description

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 two-fold:

1. Significant memory is saved.
2. Since pivoting is not required in the Gaussian elimination, there is less roundoff effect on the computed inverse.

Program Usage

Driver Utilization

- File Name - "PINVER", disc 1

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

Subprogram Utilization

- File Name - "Pinver", disc 1
- Calling Syntax

  CALL Pinver (S(*), R)
- Input Parameters

S(*)[1]	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.

- Output Parameters

S(*)[1]	Vector containing the inverse of the initial positive definite matrix stored in symmetric storage mode.

1 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^TS where S is a lower triangular matrix stored in symmetric storage mode. The result is restored in S(*).

Further explanation of these subprograms is given on pages 4-15 through 4-16 and 4-22 through 4-23.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- This subprogram takes specific advantage of the nature of the matrix. As a result, a minimum number of operations are used. Hence, you may expect better accuracy than the more general matrix routines.
- This subprogram also attempts to minimize the storage requirements. The symmetric storage vector S(*), containing the positive definite matrix, is repeatedly overwritten.
- 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(n, 1), ..., A(n, n). Subprogram Storag (p. 4-24) may be used to convert a full storage matrix to symmetric storage mode and vice versa.

1 S(*) is both an input and output parameter.

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:

$$A + SS^T \text{ implies } A^{-1} = (SS^T)^{-1} = (S^T)^{-1}S^{-1} = (S^{-1})^T S^{-1}$$

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "PINVER" and press RUN.
2. You will be asked to supply entries for the following items:
   - order of positive definite matrix A (number of rows or number of columns)
   - elements of vector S in symmetric storage mode.

   Press CONTINUE after each entry.
3. The program will print the inverse stored in symmetric storage mode vector S.

Examples

- Find the inverse of the following symmetric positive definite matrix:

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix}$$

User entries:

```
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
```

Results:

```
INVERSE S:    [IN SYMMETRIC STORAGE MODE]

 9.000000E+00 -3.600000E+01   1.920000E+02
 3.000000E+01 -1.800000E+02   1.800000E+02
```

- Find the inverse of the following symmetric positive definite matrix:

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{pmatrix}$$

User entries:

```
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
```

Results:

```
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 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", disc 1
- Calling Syntax

  CALL Triang (S(*), R)
- Input Parameters

S(*)[1]	Vector containing lower triangular matrix stored in symmetric storage mode.
R	Order of full storage mode matrix.

- Output Parameters

S(*)[1]	Product of S^TS stored in symmetric storage mode.

Special Considerations and Programming Hints

- 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 CONTINUE 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^TS. The result is restored in$S(_Z)$.

```
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
```

1 S(*) is both an input and output parameter.

Subprogram 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(*).

- File Name - contained in file "Pinver", disc 1
- Calling Syntax

  CALL Invers (S(*), R)
- Input Parameters

S(*)[1]	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(*)[1]	Vector containing the inverse of the input matrix. The inverse is also a lower triangular matrix and is stored in symmetric storage mode.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

Methods and Formulae

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 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 + I) = 1/S(I*(I - 1)/2 + I)
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 to I-1) S(I*(I - 1)/2 + L)*S(L - 1)/2 + J)
    NEXT J
NEXT I
```

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

Symmetric Storage Mode

Description

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.

Program Usage

Driver Utilization

- File Name - "STORAG", disc 1

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

Subprogram Utilization

- File Name - "Storag", disc 1
- 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

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- 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 $n(n + 1)/2$ elements. 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 $I \geq J$
$T = J*(J - 1)/2 + I$ if $I < J$

For example,

Let $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 8 \\ 3 & 6 & 7 & 9 \\ 4 & 8 & 9 & 0 \end{pmatrix}$, 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(10) = 0.

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "STORAG" and press RUN.
2. You will be asked to supply entries for the following items:
   - order of the symmetric matrix in full storage mode
   - elements of the symmetric storage mode vector if conversion is from symmetric to full storage mode
   - elements of the full storage matrix if the conversion is from full to symmetric storage mode.

   Press CONTINUE after each entry.
3. The program will print the inverted matrix.

Examples

- Full to symmetric storage mode

User entries:

```
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
```

Results:

```
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
```

- Symmetric to full storage mode

User entries:

```
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
```

Results:

```
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
```

Chapter 5
Eigen Analysis

Introduction

Description

This section contains several routines for computing the eigenvalues and eigenvectors of real variables. Suppose A is a square matrix, and consider the equation $Ax = \lambda x$, where x is a vector and λ is a constant. A number λ for which this equation has a non-zero solution vector x is called an eigenvalue of the matrix A. The solution x is called an eigenvector of the matrix A.

Routines

- Eigen - finds all the eigenvalues and eigenvectors of a real general matrix.
- Compve - finds the complex eigenvector of the real upper-Hessenberg matrix of order n; contained in file "Eigen".
- Realve - finds the real eigenvector of the real upper-Hessenberg matrix in the array A; contained in file "Eigen".
- Scale - 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; contained in file "Eigen".
- Symqr - finds the eigenvalues and eigenvectors of a real symmetric matrix.

Eigenvalues and Eigenvectors of a Real General Matrix

Description

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.

Program Usage

Driver Utilization

- File Name - "EIGEN", disc 1

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

Subprogram Utilization

- File Name - "Eigen", disc 1
- 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).

- 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
0	not found	not found
1	found	not found
2	found	found;

dimensioned Indic(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 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.

Further explanation of these subprograms is given on pages 5-8 through 5-14.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- 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-dimensional 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
0	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.

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "EIGEN" and press RUN.
2. You will be asked to supply entries for the following items:
   - order of real general matrix A
   - elements of matrix A.

   Press CONTINUE after each entry.
3. 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

User entries:

```
ORDER OF MATRIX A(*)= 3


MATRIX A:

 8.000000E+00 -1.000000E+00 -5.000000E+00
-4.000000E+00  4.000000E+00 -2.000000E+00
 1.800000E+01 -5.000000E+00 -7.000000E+00
```

Results:

```
REAL COMPONENTS OF EIGENVALUES:

 2.000000E+00  2.000000E+00  1.000000E+00


IMAGINARY COMPONENTS OF THE EIGENVALUES:

 4.000000E+00 -4.000000E+00  0.000000E+00


REAL COMPONENTS OF EIGENVECTORS
     [CONTAINED IN COLUMNS]:

 5.000000E-01  5.000000E-01 -4.082483E-01
 3.053113E-16  3.053113E-16 -8.164966E-01
 1.000000E+00  1.000000E+00 -4.082483E-01


IMAGINARY COMPONENTS OF EIGENVECTORS
     [CONTAINED IN COLUMNS]:

 5.000000E-01 -5.000000E-01  0.000000E+00
 1.000000E+00 -1.000000E+00  0.000000E+00
 0.000000E+00  0.000000E+00  0.000000E+00
```

User entries:

```
ORDER OF MATRIX A(*)= 4


MATRIX A:

 4.000000E+00 -5.000000E+00  0.000000E+00
 3.000000E+00  0.000000E+00  4.000000E+00
-3.000000E+00 -5.000000E+00  5.000000E+00
-3.000000E+00  4.000000E+00  0.000000E+00
 3.000000E+00  0.000000E+00  5.000000E+00
 4.000000E+00
```

Results:

```
REAL COMPONENTS OF EIGENVALUES:

 1.200000E+01  1.000000E+00  1.000000E+00
 2.000000E+00


IMAGINARY COMPONENTS OF THE EIGENVALUES:

 0.000000E+00  5.000000E+00 -5.000000E+00
 0.000000E+00


REAL COMPONENTS OF EIGENVECTORS
      [CONTAINED IN COLUMNS]:

-5.000000E-01 -1.000000E+00 -1.000000E+00
 5.000000E-01  5.000000E-01  5.777183E-16
 5.777183E-16  5.000000E-01 -5.000000E-01
-2.475935E-16 -2.475935E-16 -5.000000E-01
-5.000000E-01  1.000000E+00  1.000000E+00
 5.000000E-01


IMAGINARY COMPONENTS OF EIGENVECTORS
      [CONTAINED IN COLUMNS]:

 0.000000E+00  8.253118E-17 -8.253118E-17
 0.000000E+00  0.000000E+00  1.000000E+00
-1.000000E+00  0.000000E+00  0.000000E+00
 1.000000E+00 -1.000000E+00  0.000000E+00
 0.000000E+00  0.000000E+00  0.000000E+00
 0.000000E+00
```

Subprogram 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", disc 1
- 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(*)[1]	Array containing upper-Hessenberg matrix from subprogram Eigen.
Evr(*)	Vector containing the real parts of the eigenvalues.
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(*)[1] — 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).

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 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.

1 H(*) is both an input and output parameter.

Subprogram 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.

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", disc 1
- Calling Syntax
  CALL Realve(N, M, Ivec, A(*), Vecr(*), Evr(*), Evi(*), 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.
A(*)	Array containing values for which real eigenvalues of the real upper-Hessenberg matrix are to be found.
Evr(*)	Vector containing the real parts of the eigenvalues.
Evi(*)	Vector containing the imaginary parts of the eigenvalues.
Work(*)	Array used for work space.
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 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", disc 1
- Calling Syntax
  CALL Hesqr(N, A(*), H(*), Evr(*), Evi(*), Subdia(*), Indic(*), Eps, Ex)
- Input Parameters

N	Order of matrix A.
A(*)[1]	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(*)[1]	Array containing original matrix A; dimensioned H(1:N, 1:N).
Ex	2^{-39}

- Output Parameters

A(*)[1]	The input array is destroyed.
H(*)[1]	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**
0	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.

[1] A(*) and H (*) are both input and output parameters.

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.

The real parts of the n 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 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.

1 A(*) and H(*) are both input and output parameters.

Subprogram 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", disc 1
- Calling Syntax
  CALL Scale (N, A(*), H(*), Prfact(*), Enorm)
- Input Parameters

N	Order of matrix A.
A(*)[1]	Array containing matrix to be scaled and normalized so that the Euclidian norm is equal to one; dimensioned A(1:N).

- Output Parameters

A(*)[1]	Array containing the scaled matrix.
H(*)	Array containing temporary storage for original A(*).
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 Bound1 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.

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

Eigenvalues and Eigenvectors of a Real Symmetric Matrix

Description

This subprogram finds the eigenvalues and, upon your request, 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.

Program Usage

Driver Utilization

- File Name - "SYMQR", disc 1

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

Subprogram Utilization

- File Name - "Symqr", disc 1
- Calling Syntax
  CALL Symqr (A(*), D(*), E(*), K0, N, Eps, Abscnv, Vec, Trd)
- Input Parameters

A(*)[1]	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(*)[1]	Vector containing the diagonal elements if the matrix is initially tridiagonal; subscripted from 1 to N.
E(*)[1]	Vector containing the off-diagonal elements if the matrix is initially tridiagonal; subscripted from 1 to N − 1.
K0	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 = 0 if relative convergence criterion is to be used. See Special Considerations and Programming Hints for further details.
Vec	Vec = 1 if eigenvectors are to be computed. Vec = 0 if eigenvectors are not to be computed.
Trd	Trd = 1 if matrix is tridiagonal. Trd = 0 if matrix is not tridiagonal.

1 A(*), D(*), and E(*) are both input and output parameters.

- Output Parameters

A(*)[1]	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(*)[1]	Vector containing the eigenvalues of the matrix.
E(*)[1]	E(I) contains the number of iterations required to compute the approximate eigenvalue D(I).

Special Considerations and Programming Hints

- 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 =

  You may correct the data from the keyboard (e.g., N = 10, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.

- 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 <n>."

  This is a fatal error. (The program may not be continued from this point, but must be restarted.) The program pause allows you to query other variables in the subprogram environment.

- 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.xxx). 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.
- 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.

1 A(*), D(*), and E(*) are both input and output parameters.

Methods and Formulae

Symqr finds the eigenvalues and, if desired, 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.

You 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 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. 384-386.
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, Oxford: Clarendon Press, 1965.

User Instructions

1. Make sure Numerical Analysis flexible disc 1 is inserted correctly into the disc drive. Load file "SYMQR" and press RUN.
2. You will be asked to supply entries for the following items:
   - order of real symmetric matrix A (number of rows or number of columns)
   - elements of matrix A if matrix is not initially tridiagonal
   - elements of vector D if the matrix A is initially tridiagonal where the ith element of D is the element (i, i) of matrix A.
   - elements of vector E if the matrix A is initially tridiagonal where the ith element of E is the element (i + 1, i) of matrix A.
   - initial origin shift
   - absolute or relative convergence criterion usage
   - convergence tolerance.

   Press CONTINUE after each entry.
3. The eigenvalues, as well as the eigenvectors, if desired, will be printed. Eigenvector i corresponding to eigenvalue i is contained in column i of the eigenvector matrix.

Example

User entries:

```
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
CONVERGENCE TOLERANCE= 1.E-5
```

Results:

```
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   5.233642E-17   3.162278E-01
-6.324555E-01  -2.559225E-16  -7.071068E-01
 3.162278E-01  -6.324555E-01  -3.530738E-16
 7.071068E-01
```

Chapter 6
Interpolation

Introduction

Description

This section has three interpolation programs.

Routines

- Dvdfc - computes a table of confluent divided differences.
- Bnewt - performs Newton interpolation with backward divided differences; used in conjunction with Dvdfc.
- Fnewt - performs a Newton interpolation with forward divided differences; used in conjunction with Dvdfc.
- Spline - computes a curve s(x) that passes through n data points (x_i, y_i).
- Cheby - fits the tabular function Y(X) (given as m points (X, Y)) by a polynomial

$$P = \sum_{i=0}^{n} A_i X^i$$

Confluent Divided Differences

Description

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:

$$\Delta f(X_i, X_{i+1}, \ldots, X_n) = (\Delta f(X_{i+1}, \ldots, X_n) - \Delta f(X_i, \ldots, X_{n-1}))/(X_n - X_i), \quad n = i + 2, \ldots, k.$$

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].

Program Usage

Driver Utilization

- File Name - "DVDFC", disc 2

The driver "DVDFC" sets up the necessary input parameters for the subprogram Dvdfc and prints out the resulting outputs. Either subprogram Bnewt, a Newton interpolator with backward divided differences, or subprogram Fnewt, a Newton interpolator with forward divided differences, may be called to locate points of interpolation.

Subprogram Utilization

- File Name - "Dvdfc", disc 2
- Calling Syntax
  FN Dvdfc (N, X(*), V(*), B(*))
- Input Parameters

N	The number of data points to be used in the calculation.
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(*)[1]	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(*)[1]	Vector containing backward divided differences. After Dvdfc is called, you will find $B_i = \Delta f(X_i, X_{i+1}, \ldots, X_{n-1}, X_n)$ for i = 1, 2, ..., n − 1, n.
FN Dvdfc	Value of the forward divided difference $\Delta f(X_1, X_2, \ldots, X_n)$.

Special Considerations and Programming Hints

- 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 $X_1, X_2, \ldots, X_n$ are in monotonic order, then the effect of roundoff upon any nth divided difference is not 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.
- 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.

1 B(*) is both an input and output parameter.

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, pp. 167-168.

User Instructions

1. Make sure Numerical Analysis flexible disc 2 is inserted correctly into the disc drive. Load file "DVDFC" and press RUN.
2. You will be asked to supply entries for the following items:
   - number of data points to be used in the calculation
   - initial x-value for each data point (matrix X)
   - initial value of f(x) for each data point (matrix V)
   - choice of backward or forward divided differences
   - domain value (z) of the point to be interpolated.

   Press CONTINUE after each entry.
3. The value of the interpolated point and its derivative, as well as an error estimate is printed.

Subprogram 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", disc 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:

$$B(N)+(Z-X(N))*\{B(N-1)+(Z-X(N-1)\{B(N-2)+\ldots+(Z-X(2))B(1)\}\ldots\}$$

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 roundoff during the execution of Bnewt.

Special Considerations and Programming Hints

- 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 =

  You may correct the data from the keyboard (e.g., N = 20, EXECUTE). When CONTINUE 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, page 6-4.

Subprogram 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", disc 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 $F_i = \Delta f(X_i, X_{i+1}, \ldots, X_n)$ 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:

$$F(1)+(Z-X(1))*\{F(2)+(Z-X(2))\{F(3)+\ldots+(Z-X(N-1))F(N)\}\ldots\}$$

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 roundoff during the execution of Fnewt.

Special Considerations and Programming Hints

- 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 =

  You may correct the data from the keyboard (e.g., N = 5, EXECUTE). When CONTINUE 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, page 6-4.

Examples

User entries:

```
NUMBER OF DATA POINTS= 11

 I          X(*)              Y(*)

 1    -5.000000E+00    -5.000000E+00
 2    -3.000000E+00    -3.000000E+00
 3    -1.000000E+00    -1.000000E+00
 4     1.000000E+00     1.000000E+00
 5     3.000000E+00     3.000000E+00
 6     5.000000E+00     5.000000E+00
 7     7.000000E+00     7.000000E+00
 8     9.000000E+00     9.000000E+00
 9     1.100000E+01     1.100000E+01
10     1.300000E+01     1.300000E+01
11     1.500000E+01     1.500000E+01
```

Results:

```
 I          B(*)              F(*)

 1    -5.000000E+00    -5.000000E+00
 2    -3.000000E+00     1.000000E+00
 3    -1.000000E+00     0.000000E+00
 4     1.000000E+00     0.000000E+00
 5     3.000000E+00     0.000000E+00
 6     5.000000E+00     0.000000E+00
 7     7.000000E+00     0.000000E+00
 8     9.000000E+00     0.000000E+00
 9     1.100000E+01     0.000000E+00
10     1.300000E+01     0.000000E+00
11     1.500000E+01     0.000000E+00

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= 7.450000E-07
```

You enter the value of Z, Fnewt computes the rest:

User entries:

```
NUMBER OF DATA POINTS= 13

 I          X(*)              Y(*)

 1     0.000000E+00      0.000000E+00
 2     1.000000E+00      2.588190E-01
 3     2.000000E+00      5.000000E-01
 4     3.000000E+00      7.071068E-01
 5     4.000000E+00      8.660254E-01
 6     5.000000E+00      9.659258E-01
 7     6.000000E+00      1.000000E+00
 8     7.000000E+00      9.659258E-01
 9     8.000000E+00      8.660254E-01
10     9.000000E+00      7.071068E-01
11     1.000000E+01      5.000000E-01
12     1.100000E+01      2.588190E-01
13     1.200000E+01      0.000000E+00
```

Results:

```
 I          B(*)              F(*)

 1     0.000000E+00      0.000000E+00
 2     2.588190E-01      2.588190E-01
 3     5.000000E-01     -3.819000E-03
 4     7.071068E-01     -2.739367E-03
 5     8.660254E-01      9.675833E-05
 6     9.659258E-01      8.015000E-06
 7     1.000000E+00     -3.108333E-07
 8     9.659258E-01     -1.000000E-08
 9     8.660254E-01      4.811508E-10
10     7.071068E-01     -1.102293E-12
11     5.000000E-01      1.488095E-12
12     2.588190E-01     -3.858025E-13
13     0.000000E+00      6.430041E-14

NEWTON INTERPOLATION WITH
     FORWARD DIVIDED DIFFERENCES:

Z= 1.5
INTERPOLATED PT.= 3.826835E-01
DERIVATIVE= 2.418710E-01
ERROR= 2.326118E-08

Z= 11.25
INTERPOLATED PT.= 1.950901E-01
DERIVATIVE=-2.567699E-01
ERROR= 2.893984E-07
```

Cubic Spline Interpolator

Description

This subprogram computes a curve s(x) that passes through n data points (x_i, y_i) that you supply.

The integral $\int_{x_1}^{x_n} s(t)dt$ is calculated, as well as s(x) and s'(x) for any point x in the interval [x_1, x_n].

Program Usage

Driver Utilization

- File Name - "SPLINE", disc 2

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

Subprogram Utilization

- File Name - "Spline", disc 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 = 0 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

- 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."
  Eps = N =

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

- 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 CONTINUE key.)
  d. Press: EXECUTE

- 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 to n − 1. The output arguments t_j, for j = 1 to Narg may be entered in any order.
- 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.
- 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) that you supply 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. You can specify the accuracy to which these equations are solved.

The formulae employed are:

$$\int_{x_i}^{x_{i+1}} s(x)dx \approx \left\{ \frac{1}{2}(x_{i+1} - x_i)(y_i + y_{i+1}) - \frac{1}{24}(x_{i+1} - x_i)^3 [s''(x_i) + s''(x_{i+1})] \right\}$$

$$s(t_j) = y_i + (t_j - x_i)\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) + (t_j - x_i)(t_j - x_{i+1})\frac{1}{6}[s''(x_i) + s''(x_{i+1}) + s''(t_j)]$$

$$s'(t_j) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{1}{6}[(t_j - x_i) + (t_j - x_{i+1})](s''(x_i) + s''(t_j)) + \frac{1}{6}$$

$$(t_j - x_i)(t_j - x_{i+1})\left(\frac{s''(x_{i+1}) - s''(x_i)}{x_{i+1} - x_i}\right)$$

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., Ed., "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.

User Instructions

1. Make sure Numerical Analysis flexible disc 2 is inserted correctly into the disc drive. Load file "SPLINE" and press RUN.
2. You will be asked to supply entries for the following items:
   - number of data points to be used in the calculation
   - number of domain points or 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 0.
   - error tolerance for the iterative solution of the simultaneous equations
   - x-value (in increasing order) and y-value for each data point
   - domain values to be evaluated in the cubic spline.

   Press CONTINUE after each entry.
3. The program will print the resulting interpolated functional values, the values of the derivative and the value of the integral over the interval $[x_i, x_n]$.

Example

User entries:

```
# OF DATA POINTS= 5
# OF DOMAIN POINTS= 4
ERROR TOLERANCE= 1.E-6


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
```

Results:

```
INTEGRAL FROM -3.000000E+00 TO
     7.000000E+00 =   4.984962E+01


DOMAIN VALUES     FUNCTION VALUES

-2.500000E+00       3.691810E+00
 0.000000E+00       7.752235E+00
 2.500000E+00       6.670259E+00
 5.000000E+00       1.343550E+00

        DERIVATIVE VALUES
           1.399138E+00
           1.573946E+00
          -3.141092E+00
           5.030651E-01
```

Chebyschev Polynomial

Description

This subprogram fits the tabular function Y(X) (given as m points (X, Y)) by a polynomial

$$P = \sum_{i=0}^{n} A_i X^i.$$

This polynomial is the best polynomial approximation of Y(X) in the Chebyschev sense.

Program Usage

Driver Utilization

- File Name - "CHEBY", disc 2

The driver "CHEBY" sets up the necessary input parameters for the subprogram Cheby and prints out the coefficients of the resulting polynomial.

Subprogram Utilization

- File Name
- 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 0 to N; $P(X) = A_0 + A_1X + A_2X^2 + \ldots + A_nX^n$

Special Considerations and Programming Hints

- 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.
- X(*), Y(*) and A(*) must be dimensioned in the calling program: DIM X(1:M), Y(1:M), A(0:N)
- 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 Cheby."
  "# OF DATA POINTS MUST BE GREATER THAN DEG. OF POLY. + 1."
  M = N =

  The data may be corrected from the keyboard (e.g., M = 6, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.

Methods and Formulae

See the two references for a complete description of the methods 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, pp. 217-232.

User Instructions

1. Make sure Numerical Analysis flexible disc 2 is inserted correctly into the disc drive. Load file "CHEBY" and press RUN.
2. You will be asked to supply entries for the following items:
   - number of data points to be used in the calculation
   - degree of the polynomial desired for the approximation (for reasonable results, (<degree of polynomial>) < (<number of data points> − 1)
   − x and y values for each data point.

   Press CONTINUE after each entry.
3. The coefficients of the Chebyschev Polynomial will be printed where $P(X) = A_0 + A_1X + A_2X^2 + ... + A_nX^n$.

Examples

User entries:

```
# 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
```

Results:

```
COEFFICIENTS OF CHEBYSCHEV POLYNOMIAL:
[WHERE P(X)=A(0)+A(1)*X+A(2)*X^2+A(3)*X^3+...]

   A( 0)= 0.000000E+00
   A( 1)= 0.000000E+00
   A( 2)= 1.000000E+00
   A( 3)= 0.000000E+00
```

User entries:

```
# 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
```

Results:

```
COEFFICIENTS OF CHEBYSCHEV POLYNOMIAL:
[WHERE P(X)=A(0)+A(1)*X+A(2)*X^2+A(3)*X^3+...]

   A( 0)=-3.071429E-01
   A( 1)=-1.357143E-01
   A( 2)= 3.571429E-02
   A( 3)= 9.571429E-01
```

Chapter 7
Functions

Introduction

Description

This section contains several mathematical function subprograms that may be used occasionally. Since the use of these functions may be frequent and well-embedded in a main program, no drivers have been supplied with these subprograms. Their usage is explained in the following pages. You must supply the calling program.

Reference 1 below was used as the "cookbook" for these functions. Reference 2 may also be useful as a programming reference.

Routines

- FN Cosh - hyperbolic cosine
- FN Sinh - hyperbolic sine
- FN Tanh - hyperbolic tangent
- FN Gamma - gamma function
- FN Lgamma - log gamma function
- Cadd - addition of complex numbers
- Cmult - multiplication of complex numbers
- Cdivid - division of complex numbers
- Csqrt - square root of a complex number
- Cexp - exponential value of a complex number
- Clog - natural logarithm of a complex number
- Cabs - absolute value of a complex number
- Cinv - inverse of a complex number
- Ccos - cosine of a complex number
- Csin - sine of a complex number
- Ctan - Tangent of a complex number
- Ccosh - Hyperbolic cosine of a complex number
- Csinh - Hyperbolic sine of a complex number
- Ctanh - Hyperbolic tangent of a complex number
- Rec_pol - rectangular to polar conversion
- Pol_rec - polar to rectangular conversion
- Polyev - evaluation of a complex polynomial

References

1. Hart, J., et.al. Computer Approximations, New York: John Wiley and Sons, Inc., 1968.
2. Cody, William J., Jr. and Waite, William. Software Manual for the Elementary Functions, Englewood Cliffs, N.J.: Prentice Hall, Inc., 1980.

Hyperbolic Cosine

Description

This subprogram calculates the value of the hyperbolic cosine function.

Program Usage

Subprogram Utilization

- File Name - "Cosh", disc 2
- Calling Syntax
  FN Cosh(X)
- Input Parameter

X	Domain value of the function.

- Output Parameter

FN Cosh(X)	Value of the hyperbolic cosine at x.

Method and Formula

The standard formula used is:

$$\text{Cosh}(x) = (e^x + e^{-x})/2$$

Hyperbolic Sine

Description

This subprogram calculates the value of the hyperbolic sine function.

Program Usage

Subprogram Utilization

- File Name - "Sinh", disc 2
- Calling Syntax
  FN Sinh(X)
- Input Parameter

X	Domain value of the function

- Output Parameter

FN Sinh(X)	Value of hyperbolic sine at x.

Methods and Formulae

The hyperbolic sine function may be defined as follows:

$$\text{Sinh}(x) = (e^x - e^{-x})/2$$

However, 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:
   $\text{Sinh}(x) \approx .008420538263*x^5 + .166656747807*x^3 + 1.00000034157$
2. $.5 < x \leq 1$: $\text{Sinh}(x) \approx \frac{1}{2}[D(x) + D(x)/(D(x) + 1)]$ 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: $\text{Sinh}(x) \approx e^x/2$

The identity $\text{Sinh}(-x) = -\text{Sinh}(x)$ is used for negative domain values.

Hyperbolic Tangent

Description

This subprogram calculates the value of the hyperbolic tangent function.

Program Usage

Subprogram Utilization

- File Name - "Tanh", disc 2
- Calling Syntax
  FN Tanh(X)
- Input Parameter

X	Domain value of the function.

- Output Parameter

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, so $\text{Tanh}(x) \approx D(2x)/(2 + D(2x))$ 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 $\text{Tanh}(x) \approx 1$.

The identity $\text{Tanh}(-x) = -\text{Tanh}(x)$ allows for the absolute values used above.

Gamma Function

Description

For a given argument x, (x = 0, − 1, − 2, ...) function subprogram FN Gamma computes the value of the gamma function. For an argument x > 70.957, the log gamma function, FN Lgamma, should be used to avoid machine overflow.

Program Usage

Subprogram Utilization

- File Name - "Gamma", disc 2
- Calling Syntax - FN Gamma(X)
- Input Parameter

X	Function argument; X ≠0, − 1, − 2, − 3, ...; arguments must be in the range [− 69, 70.957] to avoid machine overflow.

- Output Parameter

FN Gamma(X)	Value of the gamma function evaluated at x.

Special Considerations and Programming Hints

- 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 0 OR A NEGATIVE INTEGER."

  Unless you are willing to change your argument value, there is no recovery from this error since the function is not able to handle zero or negative integer arguments.

- 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 you are willing to change your argument value, there is no recovery from this error. The argument must be between − 69 and 70.957.

Methods and Formulae

For a given argument x, ($x \neq 0, -1, -2, \ldots$) this function subprogram computes the value of the gamma function of x, $\Gamma(x)$. The recurrence $\Gamma(Z + n) = \prod_{k=0}^{n-1} (Z + k)\Gamma(Z)$ 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

$$\Gamma(x) = \prod_{k=0}^{n-1} (x + 2 + k)\Gamma(x + 2) \qquad \text{for } n > 0,$$

$$\frac{\Gamma(x + 2)}{\prod_{k=1}^{|n|} (x + 2 - k)} \qquad \text{for } n < 0.$$

For large x (in absolute value), the above computation is time consuming, and it is more economical to use the log gamma function.

Log Gamma Function

Description

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.

Program Usage

Subprogram Utilization

- File Name - "Lgamma", disc 2
- Calling Syntax
  FN Lgamma(X)
- Input Parameter

X	Domain value of the function; X must be a real, non-negative number.

- Output Parameter

FN Lgamma(X)	Value of the log gamma function evaluated at x.

Method and Formula

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:

$$\phi(x) = \frac{1}{x}R_{n,m}(1/x^2)$$

Addition with Complex Numbers

Description

Subprogram Cadd adds a set of complex numbers.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Cadd (N, A(*), B(*), Real, Imag)
- 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

Real	Real component of the sum of the complex numbers.
Imag	Imaginary component of the sum of the complex numbers.

Special Considerations and Programming Hints

- 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.

Multiplication with Complex Numbers

Description

Subprogram Cmult multiplies two complex numbers.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Cmult (A1, B1, A2, 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.

Division with Complex Numbers

Description

Subprogram Cdivid divides one complex number by another.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Cdivid (A1, B1, A2, 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.

Square Root of a Complex Number

Description

Subprogram Csqrt finds the square root of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Csqrt(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.

Method and Formula

$$R = \frac{A_2 + B_2 + A}{2}; \quad I = \frac{A_2 + B_2 - A}{2}$$

Special Considerations

- To keep the radical real, the positive sign will always be used with the $|a + ib|$.

Exponential Value of a Complex Number

Description

Subprogram Cexp finds the exponential value of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- 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.

Method and Formula

$R = e^{A} * \cos(B)$; $I = e^{A} * \sin(B)$

Special Considerations

- All of the trigonometric functions require that the argument of the function be given in radians.

Natural Logarithm of a Complex Number

Description

Subprogram Clog finds the natural logarithm of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- 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.

Method and Formula

$R = LOG(\sqrt{A^2 + B^2})$; I = atn (B/A) if $A > 0$
= atn (B/A) + acs (− 1) if $A < 0$ and $B \geq 0$
= atn (B/A) − acs (− 1) if $A < 0$ and $B < 0$
= asn (SGN (B)) if $A = 0$

Special Considerations

- All the trigonometric functions require that the argument of the function be given in radians.
- 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. Returned phase I is defined on the interval $[-\pi, +\pi]$.

Absolute Value of a Complex Number

Description

Subprogram Cabs finds the absolute value of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Cabs (X, Y, Cabs)
- Input Parameters

X	Real component of the complex number.
Y	Imaginary component of the complex number.

- Output Parameter

Cabs	Absolute value of the complex number x + iy.

Inverse of a Complex Number

Description

Subprogram Cinv finds the inverse of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in "Complx", disc 2
- Calling Syntax
  CALL Cinv (X, Y, R, I)
- Input Parameters

X	Real component of the complex number.
Y	Imaginary component of the complex number.

- Output Parameters

R	Real component of the inverse.
I	Imaginary component of the inverse.

Method and Formula

$R = X/(X^2 + Y^2); \quad I = -Y/(X^2 + Y^2)$

Cosine of a Complex Number

Description

Subprogram Ccos finds the cosine of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Ccos (X, Y, R, I)
- Input Parameters

X	Real component of the complex number
Y	Imaginary component of the complex number

- Output Parameters

R	Real component of the cosine.
I	Imaginary component of the cosine.

Method and Formula

$R = \cos(X)\cosh(Y); \quad I = -\sin(X)\sinh(Y)$

Special Considerations

- All of the trigonometric functions require that the argument of the function be given in radians.
- The cosecant function $(R + i * I) = \csc(X + i * Y)$ can be computed by using
  CALL Ccos (X, Y, R, I)
  CALL Cinv (R, I, R, I)

Sine of a Complex Number

Description

Subprogram Csin finds the sine of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Csin (X, Y, R, I)
- Input Parameters

X	Real component of the complex number.
Y	Imaginary component of the complex number.

- Output Parameters

R	Real component of the sine.
I	Imaginary component of the sine.

Method and Formula

$R = \sin(X)\cosh(Y); \quad I = \cos(X)\sinh(Y)$

Special Considerations

- All of the trigonometric functions require that the argument of the function be given in radians.
- The secant function $(R + i * I) = \sec(X + i * Y)$ can be computed by using
  CALL Csin (X, Y, R, I)
  CALL Cinv (R, I, R, I)

Tangent of a Complex Number

Description

Subprogram Ctan finds the tangent of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Ctan (X, Y, R, I)
- Input parameters

X	Real component of the complex number
Y	Imaginary component of the complex number.

- Output Parameters

R	Real component of the tangent.
I	Imaginary component of the tangent

Method and Formula

$Z = (R + i * I) = \sin (X + i * Y)/\cos (X + i * Y)$

Special Considerations

- All of the trigonometric functions require that the argument of the function be given in radians.
- The cotangent function $(R + i * I) = \cot (X + i * Y)$ can be computed by using
  CALL Ctan (X, Y, R, I)
  CALL Cinv (R, I, R, I)

Hyperbolic Cosine of a Complex Number

Description

Subprogram Ccosh finds the hyperbolic cosine of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Ccosh (X, Y, R, I)
- Input Parameters

X	Real component of the complex number
Y	Imaginary component of the complex number

- Output Parameters

R	Real component of the hyperbolic cosine
I	Imaginary component of the hyperbolic cosine

Method and Formula

$R = \cosh(X) * \cos(Y); \quad I = \sinh(X) * \sin(Y)$

Special Consideration

- All of the trigonometric functions require that the argument of the function be given in radians.
- The hyperbolic cosecant $(R + i * I) = \text{csch}(X + i * Y)$ can be computed by using
  CALL Ccosh (X, Y, R, I)
  CALL Cinv (R, I, R, I)

Hyperbolic Sine of a Complex Number

Description

Subprogram Csinh finds the hyperbolic sine of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Csinh (X, Y, R, I)
- Input Parameters

X	Real component of the complex number.
Y	Imaginary component of the complex number.

- Output Parameters

R	Real component of the hyperbolic sine
I	Imaginary component of the hyperbolic sine

Method and Formula

R = sinh (X) * cos (Y); R = cosh (X) * sin (Y)

Special Consideration

- All of the trigonometric functions require that the argument of the function be given in radians.
- The hyperbolic secant (R + i * Y) = sech (X + i * Y) can be computed by using
  CALL Csinh (X, Y, R, I)
  CALL Cinv (R, I, R, I)

Hyperbolic Tangent of a Complex Number

Description

Subprogram Ctanh finds the hyperbolic tangent of a complex number.

Program Usage

Subprogram Utilization

- File Name - contained in "Complx", disc 2
- Calling Syntax
  CALL Ctanh (X, Y, R, I)
- Input Parameters

X	Real component of the complex number.
Y	Imaginary component of the complex number.

- Output Parameters

R	Real component of the hyperbolic tangent.
I	Imaginary component of the hyperbolic tangent.

Method and Formula

$Z = (R + i * I) = \sinh (X + i * Y)/\cosh (X + i * Y)$

Special Considerations

- All of the trigonometric functions require that the argument of the function be given in radians.
- The hyperbolic cotangent $(R + i * I) = \coth (X + i * Y)$ can be computed by using
  CALL Ctanh (X, Y, R, I)
  CALL Cinv (R, I, R, I)

Rectangular to Polar Conversion

Description

Subprogram Rec_pol finds the magnitude and the phase (in the angular unit of the calling context) of a complex number or point in rectangular coordinates.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL (A, B, Magn, Phase)
- Input Parameters

A	Real component of the complex number, or X-coordinate of the point.
B	Imaginary component of the complex number, or Y-coordinate of the point.

- Output Parameters

Magn	Magnitude (orAbsolute Value) of the complex number, or p-coordinate of the point.
Phase	Phase of the complex number, or 0-coordinate of the point, in angular unit of the calling context.

Method and Formula

$$\text{Magn} = \sqrt{A^2 + B^2}; \quad \text{Phase}^* = \begin{cases} \text{atn}(B/A) & \text{if } A > 0 \\ \text{atn}(B/A) + \text{acs}(-1) & \text{if } A < 0 \\ \text{asn}(\text{SGN}(B)) & \text{if } A = 0 \end{cases}$$

(* module $2 * \pi$ definition)

Special Considerations

- Unlike other complex functions, the angular unit mode can be chosen.
- Returned phase is defined on the interval $[+ 180°, + 180°]$ or $[-\pi, +\pi]$

Polar to Rectangular Conversion

Description

Subprogram Pol_rec finds the complex number (or the rectangular coordinates) of a point defined by its magnitude and phase (or polar coordinates) in the angular unit of the calling context.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- Calling Syntax
  CALL Pol_rec (A, B, Magn, Phase)
- Input Parameters

Magn	Magnitude of the complex number, or p-coordinate of the point.
Phase	Phase of the complex number, or Θ-coordinate of the point in angular unit of the calling context.

- Output Parameters

A	Real component of the complex number, or X coordinate of the point.
B	Imaginary component of the complex number or Y coordinate of the point.

Method and Formula

A = Magn * cos (Phase); B = Magn * sin (Phase)

Special Considerations

- Unlike other complex functions, the angular unit mode can be chosen.

Evaluation of a Complex Polynomial

Description

Subprogram Polyev evaluates the complex polynomial

$(a_0 + b_0i) + (a_1 + b_1i)Z + \ldots + (a_n + b_ni)Z^n$ at a complex point.

Program Usage

Subprogram Utilization

- File Name - contained in file "Complx", disc 2
- 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(0) + Icoef(0)*I) + (Rcoef(1) + Icoef(1)*I)*Z + ... + (Rcoef(N) + Icoef(N)*I)*Z ↑ N; subscripted from 0 to N.
Icoef(*)	Vector containing the real components of the coefficients of the polynomial (Rcoef(0) + Icoef(0)*I) + (Rcoef(1) + Icoef(1)*I)*Z + ... + (Rcoef(N) + Icoef(N)*I)*Z ↑ N; subscripted from 0 to N.

- Output Parameters

Rval	Real component of the evaluated polynomial.
Ival	Imaginary component of the evaluated polynomial.

Chapter 8
Fourier Analysis

Introduction

Description

This section contains routines for computing Fourier series coefficients using equally or unequally spaced data, as well as a fast Fourier transform routine.

Routines

- Foureq - calculates the Fourier series coefficients using equally-spaced data points.
- Fourun - calculates the Fourier series coefficients using unequally-spaced data points.
- Fft - calculates a Fast Fourier Transform from a set of time domain points to a set of frequency domain points or vice versa.

Fourier Series Coefficients for Equally-spaced Data Points

Description

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.

Program Usage

Driver Utilization

- File Name - "FOUREQ", disc 2

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

Subprogram Utilization

- File Name - "Foureq", disc 2
- 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_i.
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 0 to N.
B(*)	Vector containing the B_R Fourier Series coefficients; subscripted from 1 to N.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- 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 is printed and the program pauses:

  "ERROR IN SUBPROGRAM Foureq."
  "ODD NUMBER OF DATA POINTS REQUIRED."
  Npts =

  The data may be corrected from the keyboard (e.g., Npts = 15, EXECUTE). When CONTINUE is pressed, the program will resume execution at the next line.

- For valid results, you should provide at least n data points to compute n coefficients.
- Y(*), A(*) and B(*) must all be dimensioned in the calling program.

Methods and Formulae

If $g(x) = f(x)\cos\left(\frac{2\pi ix}{T}\right)$ and $h(x) = f(x)\sin\left(\frac{2\pi ix}{T}\right)$ 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.

User Instructions

1. Make sure Numerical Analysis flexible disc 2 is inserted correctly into the disc drive. Load file "FOUREQ" and press RUN.
2. You will be asked to supply entries for the following items:
   - number of data points to be used in the calculation (must be an odd number)
   - highest coefficient desired in the Fourier Series (For valid results this should be less than or equal to the number of data points.)
   - initial domain value (first x-value)
   - increment between x-values
   - range values (y-values) for each data point.

   Press CONTINUE after each entry.
3. The program will calculate and print the Fourier Series Coefficients using the given data points.

Example

User entries:

```
# 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
```

Results:

```
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)= 3.389428E-16
   A( 4)= 2.900530E-01     B( 4)= 1.197223E-01
   A( 5)=-3.333335E-01     B( 5)= 9.622504E-02
```

Fourier Series Coefficients for Unequally-spaced Data Points

Description

This subprogram calculates the Fourier Series Coefficients for a function defined by discrete data points (x_i, y_i), $i = 1,\ldots,n$. 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 $i = 1,\ldots,n-1$.

Program Usage

Driver Utilization

- File Name - "FOURUN", disc 2

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

Subprogram Utilization

- File Name - "Fourun", disc 2
- 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 0 to N.
B(*)	Vector containing the B_k Fourier Series coefficients; subscripted from 1 to N.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

- For valid results, you should provide at least n data points to compute n coefficients.
- X(*), Y(*), A(*), B(*) must all be dimensioned in the calling subprogram.
- 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), $i = 1$ to 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:

$$a_i = \frac{2}{T} \int_{x_1}^{x_1+T} f(x)\cos\frac{2\pi i x}{T}\,dx \text{ for } i = 0 \text{ ton, and}$$

$$b_i = \frac{2}{T} \int_{x_1}^{x_1+T} f(x)\sin\frac{2\pi i x}{T}\,dx \text{ for } i = 1 \text{ to n.}$$

T specifies the period equivalent to $(x_i - x_1)$ 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:

$$A_j = \sum_{i=2}^{k-1} S_i \qquad \text{for } j = 1 \text{ to n}$$

$$B_j = \sum_{1=2}^{k-1} T_i \qquad \text{for } j = 1 \text{ to n}$$

where:

$$Q_i = \frac{(x_{i+1} - x_{i+2})\, y_i - (x_i - x_{i+2})\, y_{i+1} + (x_i - x_{i+1})\, y_{i+2}}{(x_{i+1} - x_{i+2})\,(x_i - x_{i+2})\,(x_i - x_{i+1})}$$

$$A = \frac{1}{2}\,(Q_i + Q_{i-1})$$

$$R_i = \frac{-(x_{i+1}^2 - x_{i+2}^2)\, y_i - (x_i^2 - x_{i+2}^2)\, y_{i+1} + (x_i^2 - x_{i+1}^2)\, y_{i+2}}{(x_{i+1} - x_{i+2})\,(x_i - x_{i+2})\,(x_i - x_{i+1})}$$

$$B = \frac{1}{2}\,(R_i + R_{i-1})$$

$$S_i = \frac{(2Ax_{i+1} + B)\cos\left[2\pi\left(\frac{x_{i+1}J}{x_k - x_1}\right)\right] - (2Ax_i + B)\cos\left[2\pi\left(\frac{x_i J}{x_k - x_1}\right)\right]}{\left(\frac{J}{x_k - x_1}\right)^2} +$$

$$\frac{y_{i+1}\left(\frac{J}{x_k - x_1}\right)^2 - 2A}{\left(\frac{2\pi J}{x_k - x_1}\right)^3}\sin\left[2\pi\left(\frac{x_{i+1}J}{x_k - x_1}\right)\right] -$$

$$\frac{y_i\left(\frac{J}{x_k - x_1}\right)^2 - 2A}{\left(\frac{2\pi J}{x_k - x_1}\right)^3}\sin\left[2\pi\left(\frac{x_i J}{x_k - x_1}\right)\right]$$

$$T_i = \frac{\left(2Ax_{i+1} + B\right)\sin\left[2\pi\mathrm{frc}\left(\frac{x_{i+1}J}{x_k - x_1}\right)\right] - \left(2Ax_i + B\right)\sin\left[2\pi\mathrm{frc}\left(\frac{x_i J}{x_k - x_1}\right)\right]}{\left(\frac{J}{x_k - x_1}\right)^2} -$$

$$\frac{y_{i+1}\left(\frac{J}{x_k - x_1}\right)^2 - 2A}{\left(\frac{2\pi J}{x_k - x_1}\right)^3} \cos\left[2\pi\left(\frac{x_{i+1} J}{x_k - x_1}\right)\right] +$$

$$\frac{y_i\left(\frac{J}{x_k - x_1}\right)^2 - 2A}{\left(\frac{2\pi J}{x_k - x_1}\right)^3} \cos\left[2\pi\left(\frac{x_i J}{x_k - x_1}\right)\right]$$

where J is the number of coefficient being computed and frc () indicates fractional part.

$$A_0 = \sum_{1=2}^{k=1} U_i$$

$$U_i = \frac{A(x_{i+1}^3 - x_i^3)}{3} + \frac{B(x_{i+1}^2 - x_i^2)}{2} + C(x_{i+1} - x_i)$$

where A, B are defined above and:

$$C = \frac{1}{2}(P_i + P_{i-1})$$

$$P_i = \frac{(x_{i+1} - x_{i+2})x_{i+1}\, x_{i+2}\, y_i - (x_i - x_{i+2})x_i\, x_{i+2}\, y_{i+1} + (x_i - x_{i+1})x_i\, x_{i+1}\, y_{i+2}}{(x_{i+1} - x_{i+2})(x_i - x_{i+2})(x_i - x_{i+1})}$$

Sine and cosine functional values are computed recursively with the following formulae:

$$\sin\left(\frac{2\pi x_i}{x_k - x_1}(J+1)\right) = \sin\left(\frac{2\pi x_i}{x_k - x_1}\right)\cos\left(\frac{2\pi x_i}{x_k - x_1} J\right) + \cos\left(\frac{2\pi x_i}{x_k - x_1}\right)\sin\left(\frac{2\pi x_i}{x_k - x_i} J\right)$$

$$\cos\left(\frac{2\pi x_i}{x_k - x_1}(J+1)\right) = \cos\left(\frac{2\pi x_i}{x_k - x_1}\right)\cos\left(\frac{2\pi x_i}{x_k - x_1} J\right) - \sin\left(\frac{2\pi x_i}{x_k - x_1}\right)\sin\left(\frac{2\pi x_i}{x_k - x_i} J\right)$$

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, McGraw-Hill, 1962, pp. 67-80.
3. Acton, Forman S., Numerical Methods that Work, Harper and Row, 1970, pp. 221-257.

User Instructions

1. Make sure Numerical Analysis flexible disc 2 is inserted correctly into the disc drive. Load file "FOURUN" and press RUN.
2. You will be asked to supply entries for the following items:
   - number of data points to be used in the calculation
   - highest Fourier Series Coefficient desired (For valid results, this should be less than or equal to the number of data points provided.)
   - discrete x-values (in increasing order) and y-values for each data point.

   Press CONTINUE after each entry.
3. The program will print the resulting A_k and B_k Fourier Series Coefficients.

Example

User entries:

```
# 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
```

Results:

A COEFFICIENTS	B COEFFICIENTS
5.000000E-01	
3.262358E-01	4.768298E-01
4.119709E-01	7.131732E-01
-1.202742E-16	6.307360E-01
-2.036138E-01	3.520154E-01
-1.614057E-01	9.246861E-02
-2.460994E-15	-7.230697E-04
1.122901E-01	6.417231E-02
9.651357E-02	1.666468E-01
-2.775558E-15	1.936027E-01
-7.365751E-02	1.275692E-01

Fast Fourier Transform

Description

This subprogram calculates a Fast Fourier Transform from a set of time domain points to a set of frequency domain points or the inverse Fast Fourier Transform, calculating the set of time domain points from a set of frequency domain points.

Program Usage

Driver Utilization

- File Name - "FFT", disc 2

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

Subprogram Utilization

- File Name - "Fft", disc 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	If Flg = 1, calculate Fast Fourier Transform from a set of time domain points to a set of frequency domain points. If Flg = − 1, calculate Inverse Fast Fourier Transform from a set of frequency domain points to a set of time domain points.
R(*)[1]	Vector subscripted from 1 to N; as time domain data, R(*) contains the odd-numbered data points, i.e., R_i = 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(*)[1]	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(*)[1]	Vector subscripted from 1 to N (see R(*) as an input parameter).
I(*)[1]	Vector subscripted from 1 to N (see I(*) as an input parameter).

1 R(*) and I(*) are both input and output parameters.

Special Considerations and Programming Hints

- 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 CONTINUE is pressed, the program will resume execution at the next line.

Data Points (Full Precision Arrays)	FFT	IFT
256	4.3 sec	4.0 sec
512	9.1 sec	8.7 sec
1024	19.6 sec	18.6 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:

$$Z(n) = X(n) + i\,Y(n) \qquad 0 \leq n < N \text{ data points}$$

The transform is:

$$Z(m) = X(m) + i\,Y(m)$$

where

$$X(m) = \frac{Z(m) + Z*(N - m)}{2}$$

$$Y(m) = \frac{Z(m) - Z*(N - m)}{2_i}$$

$Z*$ 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:

1. A transform of twice the length can be handled with no increase in storage for input data.
2. 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.

The transform of this is:

$$F(m) = \sum_{n=0}^{N-1} X(2n)w^{mn} + \sum_{n=0}^{N-1} Y(2n + 1)w^{mn}$$

$$= \sum_{p=0}^{N-1} X(p)w^{2mp} + \sum_{p=0}^{N-1} Y(p)w^{2mp}w^{m}$$

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:

1. Since F(0) and F(N) are real only, F(N) can be stored in the imaginary location of F(0), i.e., F(1).
2. $w^m = c^{-2\pi im/2N}$. This is half the minimum value of rotation normally used in an N point transform.
3. $*$ = complex conjugate.

References

1. Brigham, E.O., The Fast Fourier Transform, Englewood Cliffs, N.J.: Prentice Hall, Inc., 1974, Chapter 10.
2. Cooley, J.W., and Tukey, J.W., "An Algorithm for Machine Calculation of Complex Fourier Series," Math. Computation, Vol. 19 (April 1965), pp. 297-301.

User Instructions

1. Make sure Numerical Analysis flexible disc 2 is inserted correctly into the disc drive. Load file "FFT" and press RUN.
2. You can append subprogram Fft by answering "no" ("N") when asked, "HAS Fft ALREADY BEEN APPENDED (Y/N)?" This will cause file "Fft" to be linked to the end of the driver program.This only needs to be done when the driver program (FFT) is initially loaded or reloaded.
3. You will be asked to supply entries for the following items:
   If time data is to be entered:
   - number of time domain points (power of 2 from 4 to 1024) to be used in the calculations
   - time domain data points.

   The program will print the DC term, the maximum frequency, and the real and imaginary components of the resulting frequency data.
   or
   If frequency data is to be entered:
   - value of the number of coefficient pairs times 2 plus 2 (power of 2 between 4 and 1024)
   - DC term
   - maximum frequency term
   - real and imaginary coefficients for the frequency domain data points.

   The program will print the time domain data points obtained from the inverse Fast Fourier Transform.

Examples

- Time to frequency domain
- Time domain data was generated using the following formula:

$$\text{POINT}(I) = \text{SIN}((I-1)*\pi/8), \qquad \text{for } I = 1 \text{ to } 16.$$

User entries:

```
# 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
```

Results:

```
DC TERM= 0
MAX FREQUENCY= 0


FREQUENCY DOMAIN:

              REAL          IMAGINARY

1         2.775558E-17     -1.000000E+00
2         0.000000E+00      0.000000E+00
3         4.274748E-17      2.078397E-09
4         0.000000E+00      0.000000E+00
5        -9.825863E-17      1.538152E-08
6         0.000000E+00      0.000000E+00
7         2.775558E-17      2.786261E-08
```

- Time to frequency domain

User entries:

```
# 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
```

Results:

```
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         -1.339453E-06          -1.339453E-06
4          0.000000E+00           0.000000E+00
5          6.134476E-06          -6.134476E-06
6          0.000000E+00           0.000000E+00
7         -1.502234E-05          -1.502234E-05
```

- Frequency to time domain

User entries:

```
(# OF COEFFICIENT PAIRS*2)+2= 16


DC TERM= 0
MAX FREQUENCY TERM= 0


FREQUENCY DOMAIN DATA:

   REAL(    1)= 1.000000E+00
   REAL(    2)= 0.000000E+00
   REAL(    3)=-1.339500E-06
   REAL(    4)= 0.000000E+00
   REAL(    5)= 6.134500E-06
   REAL(    6)= 0.000000E+00
   REAL(    7)=-1.502200E-05


   IMAG(    1)=-1.000000E+00
   IMAG(    2)= 0.000000E+00
   IMAG(    3)=-1.339500E-06
   IMAG(    4)= 0.000000E+00
   IMAG(    5)=-6.134500E-06
   IMAG(    6)= 0.000000E+00
   IMAG(    7)=-1.502200E-05
```

Results:

```
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)= 1.110223E-16
   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)=-1.110223E-16
   DATA POINT(   16)= 5.411945E-01
```

Part No. 98821-13111
E0482

Printed in U.S.A.
First Edition, April 82