A Self-Modifying Extrapolation Method for Solving Ordinary Differential Equations

Dar D. Daily
Kansas State University, Manhattan, Kansas
Author's address: 8133 Dearborn Drive, Prairie Village, KS 66208

This paper outlines a program that searches for the predominant terms of the asymptotic error expansion of initial value problems in ordinary differential equations and uses this information in a self-modifying extrapolation process. During the integration process, using a ratio that Carl de Boor (1971) used in an integration program, the method seeks to recognize trends of change in the error expansion of the differential equation and to adjust the method of extrapolation. A basic algorithm used in the modifying process is presented along with a brief explanation. Also, a comparison made with the well-known rational extrapolation method shows rational extrapolation to be generally less efficient in terms of function evaluations but also demonstrates that the self-modifying method is generally not able to reduce its error to the level of rational extrapolation. A note, though, shows the self-modifying method to be superior to the regular Romberg extrapolation.

Key Words and Phrases: self-modifying extrapolation, rational extrapolation, modified midpoint method, Romberg integration, asymptotic error expansion, predominant, singularity, initial value problems in ordinary differential equations; CR Categories: 5.10, 5.17

A Computer Solution of Polygonal Jigsaw Puzzles

Ejvind Lynning
University of Arhus, Arhus, Denmark
Author's address: Brandeis University, Waltham, MA. 02154

A program to solve any jigsaw puzzle involving pieces of polygonal shape is described. An efficient solution has been found to depend on a number of ad hoc strategies, which are described in detail in the paper. The puzzles are solved by successively placing individual pieces in the region to be covered using a depth-first tree search algorithm. A formal representation of regions, pieces, and placings of pieces is defined. The main idea behind the chosen representation is to orient clockwise the polygons making up a region, and to orient counterclockwise the pieces to be placed. Placing a piece means computing a valid new region, i.e. one or more clockwise oriented polygons, constructed from the previous one by removing the part corresponding to the piece which is placed. The data structure and the procedures required to examine where pieces can be placed and how to perform the placing of the pieces are also described. All puzzles so far presented to the program have been successfully solved in a reasonable time.

Key Words and Phrases: artificial intelligence, problem solving, pattern recognition, puzzles, polygonal puzzles, jigsaw puzzles, backtrack programming, tree search algorithms; CR Categories: 3.6, 3.63, 3.64

Algorithms — L.D. Fosdick and A.K. Cline, Editors

Algorithm 450

Rosenbrock Function Minimization [E4]

Marek Machura* and Andrzej Mulawa†
[Recd. 22 March 1971]

* Institute of Automation and Measurements, Warsaw, Poland.
† Institute of Computing Machinery, Warsaw, Poland.

Key words and phrases: function minimization, Rosenbrock's method
CR Categories : 5.19
Language : Fortran

Description

Purpose. This subroutine finds the local minimum of a function of n variables for an unconstrained problem. It uses the method for direct search minimization as described by Rosenbrock [1].

Method. The local minimum of a function is sought by conducting cyclic searches parallel to each of the n orthogonal unit vectors, the coordinate directions, in turn. n such searches constitute one stage of the iteration process. For the next stage a new set of n orthogonal unit vectors is generated, such that the first vector of this set lies along the direction of greatest advance for the previous stage. The Gram-Schmidt orthogonalization procedure is used to calculate the new unit vectors.

Program. The communication to the subroutine *ROMIN* is solely through the argument list. The user must supply two additional subroutines *FUNCT* and *MONITOR*. The entrance to the subroutine is achieved by

CALL ROMIN (*N*, *X*, *FUNCT*, *STEP*, *MONITOR*)

The meaning of the parameters is as follows. *N* is the number of independent variables of the function to be minimized. *X*(*N*) is an estimate of the solution. On entry it is an initial estimate to be provided by the user; on exit it is the best estimate of the solution found. *FUNCT* (*N*, *X*, *F*) is a subroutine calculating the value *F* of the minimized function at any point *X*. *STEP* is an initial step length for all searches of the first stage. The subroutine *MONITOR* (*N*, *X*, *F*, *R*, *B*, *CON*, *NR*) supplies printouts of any parameter from the argument list and contains convergence criteria chosen by the user. (Different kinds of convergence criteria and their use are discussed in [1] and [4].) *R* is the actual number of function evaluations. *B* is the value of the Euclidean norm of the vector representing the total progress made since the axes were last rotated, i.e. the total progress in one stage. *CON* is a logical variable. At the

start of the subroutine *ROMIN CON* is set *.FALSE.*. If the convergence criteria are satisfied *CON* must be set *.TRUE.* in the subroutine *MONITOR*, which transfers control back to the main program. *NR* is the *MONITOR* index used as described in [3]. The *CALL* statement of the subroutine *MONITOR* with *NR* equal to 1 occurs once per function evaluation and with *NR* equal to 2 once per stage of the iteration process.

Test results. As a test example, the parabolic valley function

$$f(x_1, x_2) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$$

was chosen. This function attains its minimum equal to 0 at the point (1, 1). Starting from the point (−1.2, 1.0) the best estimate of the solution after 200 function evaluations as found by the subroutine *ROMIN* was $0.29774 \cdot 10^{-4}$ at the point (0.99513, 0.99053). The initial step length *STEP* was set equal to 0.1 [2].

References

1. Rosenbrock, H.H. An automatic method for finding the greatest or least value of a function. *Computer J. 3* (1960), 175–184.
2. Rosenbrock, H.H., Storey, C. *Computational Techniques for Chemical Engineers.* Pergamon Press, New York, 1966.
3. Rutishauser, H. *Interference with an ALGOL Procedure, in Annual Review in Automatic Programming, Vol. 2.* R. Goodman (Ed.), Pergamon Press, New York, 1961.
4. Powell, M.J.D. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer J. 6* (1964), 155–162.

Algorithm

```
      SUBROUTINE ROMIN(N, X, FUNCT, STEP, MONITR)
      INTEGER N, IP
      REAL STEP
      DIMENSION X(N)
      LOGICAL CON
      INTEGER I, J, K, L, P, R
      REAL FO, F1, B, BETY
      DIMENSION A(30), D(30), V(30,30), ALPHA(30,30), BETA(30),
     * E(30), AV(30)
C   THIS SUBROUTINE MINIMIZES A FUNCTION OF N VARIABLES
C   USING THE METHOD OF ROSENBROCK. THE PARAMETERS ARE
C   DESCRIBED AS FOLLOWS:
C     N IS THE NUMBER OF INDEPENDENT VARIABLES
C     X(N) IS AN ESTIMATE OF THE SOLUTION ( ON ENTRY -
C          AN INITIAL ESTIMATE, ON EXIT - THE BEST ESTIMATE
C          OF THE SOLUTION FOUND )
C     FUNCT(N,X,F) IS A ROUTINE PROVIDED BY THE USER TO
C          CALCULATE THE VALUE F OF THE MINIMIZED FUNCTION
C          AT ANY POINT X
C     STEP IS AN INITIAL STEP LENGTH FOR ALL COORDINATE
C          DIRECTIONS AT THE START OF THE PROCESS
C     MONITR (N,X,F,R,B,CON,NR) IS A ROUTINE PROVIDED BY
C          THE USER FOR DIAGNOSTIC AND CONVERGENCE PURPOSES
C     R IS THE ACTUAL NUMBER OF FUNCTION EVALUATIONS ( FOR
C          THE INITIAL ESTIMATE R=0 )
C     B IS THE VALUE OF THE EUCLIDEAN NORM OF THE VECTOR
C          REPRESENTING THE TOTAL PROGRESS MADE SINCE THE
C          AXES WERE LAST ROTATED
C     CON IS A LOGICAL VARIABLE. AT THE START OF THE
C          SUBROUTINE ROMIN CON=.FALSE. IF THE CONVERGENCE
C          CRITERIA OF THE ROUTINE MONITOR ARE SATISFIED
C          CON MUST BE SET .TRUE. TO STOP THE PROCESS
C     NR  IS THE MONITOR INDEX
C   INITIALIZE CON, E(I) AND R
C   E(I) IS A SET OF STEPS TO BE TAKEN IN THE CORRESPONDING
C   COORDINATE DIRECTIONS
      CON = .FALSE.
      DO 10 I=1,N
        E(I) = STEP
   10 CONTINUE
      R = 0
C   V(I,J) IS AN NXN MATRIX DEFINING A SET OF N MUTUALLY
C   ORTHOGONAL COORDINATE DIRECTIONS. V(I,J) IS THE UNIT
C   MATRIX AT THE START OF THE PROCESS
      DO 30 I=1,N
        DO 20 J=1,N
          V(I,J) = 0.0
          IF (I.EQ.J) V(I,J) = 1.0
   20   CONTINUE
   30 CONTINUE
      CALL FUNCT(N, X, FO)
C   START OF THE ITERATION LOOP
   40 DO 50 I=1,N
        A(I) = 2.0
        D(I) = 0.0
   50 CONTINUE
C   EVALUATE F AT THE NEW POINT X
   60 DO 130 I=1,N
        DO 70 J=1,N
          X(J) = X(J) + F(I)*V(I,J)
   70   CONTINUE
        R = R + 1
        CALL FUNCT(N, X, F1)
        CALL MONITR(N, X, F1, R, 0, CON, 1)
        IF (CON) GO TO 290
        IF (F1-FO) 80, 90, 90
C   THE NEW VALUE OF THE FUNCTION IS LESS THAN THE OLD ONE
   80   D(I) = D(I) + E(I)
        E(I) = 3.0*E(I)
        FO = F1
        IF (A(I).GT.1.5) A(I) = 1.0
        GO TO 110
C   THE NEW VALUE OF THE FUNCTION IS GREATER THAN OR EQUAL
C   TO THE OLD ONE
   90   DO 100 J=1,N
          X(J) = X(J) - F(I)*V(I,J)
  100   CONTINUE
        E(I) = -0.5*E(I)
        IF (A(I).LT.1.5) A(I) = 0.0
  110   DO 120 J=1,N
          IF (A(J).GE.0.5) GO TO 130
  120   CONTINUE
        GO TO 140
  130 CONTINUE
      GO TO 60
C   GRAM-SCHMIDT ORTHOGONALIZATION PROCESS
  140 DO 160 K=1,N
        DO 150 L=1,N
          ALPHA(K,L) = 0.0
  150   CONTINUE
  160 CONTINUE
      DO 190 I=1,N
        DO 180 J=1,N
          DO 170 L=I,N
            ALPHA(I,J) = ALPHA(I,J) + D(L)*V(L,J)
  170     CONTINUE
  180   CONTINUE
  190 CONTINUE
      B = 0.0
      DO 200 J=1,N
        B = B + ALPHA(I,J)**2
  200 CONTINUE
      B = SQRT(B)
C   CALCULATE THE NEW SET OF ORTHONORMAL COORDINATE
C   DIRECTIONS ( THE NEW MATRIX V(I,J) )
      DO 210 J=1,N
        V(1,J) = ALPHA(1,J)/B
  210 CONTINUE
      DO 280 P=2,N
        BETY = 0.0
        IP = P - 1
        DO 220 M=1,N
          BETA(M) = 0.0
  220   CONTINUE
        DO 250 J=1,N
          DO 240 K=1,IP
            AV(K) = 0.0
            DO 230 L=1,N
              AV(K) = AV(K) + ALPHA(P,L)*V(K,L)
  230       CONTINUE
            BETA(J) = BETA(J) - AV(K)*V(K,J)
  240     CONTINUE
  250   CONTINUE
        DO 260 J=1,N
          BETA(J) = BETA(J) + ALPHA(P,J)
          BETY = BETY + BETA(J)**2
  260   CONTINUE
        BETY = SQRT(BETY)
        DO 270 J=1,N
          V(P,J) = BETA(J)/BETY
  270   CONTINUE
  280 CONTINUE
C   END OF GRAM-SCHMIDT PROCESS
      CALL MONITR(N, X, FO, R, B, CON, 2)
      IF (CON) GO TO 290
C   GO TO THE NEXT ITERATION
      GO TO 40
  290 RETURN
      END
```

Algorithm 451

Chi-Square Quantiles [G1]

Richard B. Goldstein [Recd. 30 June 1971 and 20 March 1972]
Department of Mathematics, Providence College, Providence, R.I.

Key Words and Phrases: Chi-square statistic, asymptotic approximation, normal deviate, chi-square deviate, degrees of freedom
CR Categories: 5.12, 5.5
Language: Fortran

Description

The algorithm evaluates the quantile at the probability level *P* for the Chi-square distribution with *N* degrees of freedom. The

Fig. 1

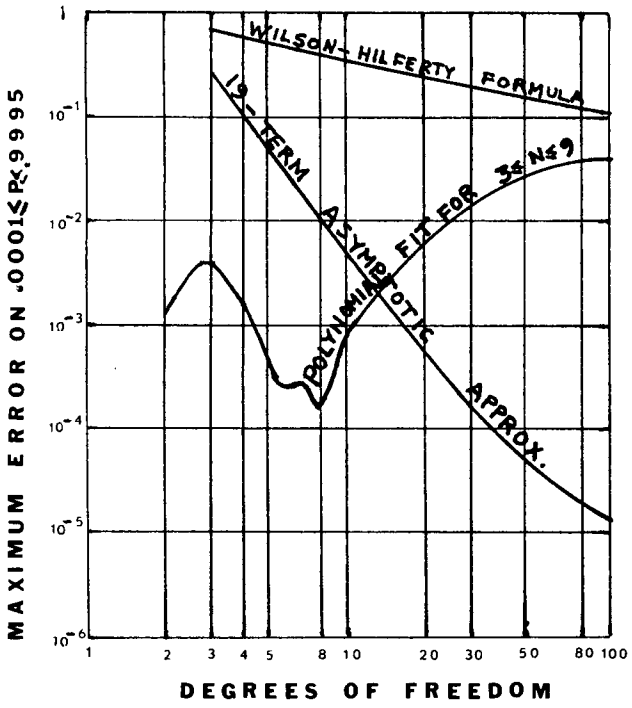
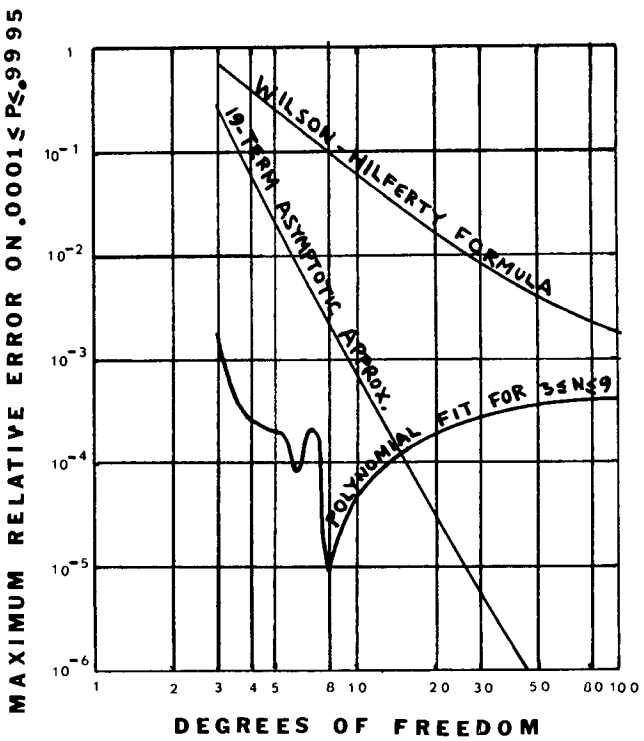


Fig. 2



quantile function is an inverse of the function

$$P(X \mid N) = (2^{N/2}\Gamma(N/2))^{-1} \int_{X(P)}^{\infty} Z^{\frac{1}{2}N-1} e^{-\frac{1}{2}Z}\, dZ \quad (x \geq 0, N \geq 1).$$

The function $GAUSSD(P)$ is assumed to return the normal deviate for the level P, e.g. -1.95996 for $P = .025$. The procedure found in Hastings [5] may be used, or for increased accuracy, the procedure found in Cunningham [3] may be used.

The Wilson-Hilferty cubic formula [7] which is

$$\chi^2 \sim N\{1 - 2/9N + X\,(2/9N)^{\frac{1}{2}}\}^3$$

where X is the normal deviate can be extended to the 19-term asymptotic approximation:

$$\begin{aligned}\chi^2 \sim N\{&1 - 2/9N + (4X^4+16X^2-28)/1215N^2 \\ &+ (8X^6+720X^4+3216X^2+2904)/229635N^3 + \cdots \\ &+ (2/N)^{\frac{1}{2}}[X/3 + (-X^3+3X)/162N \\ &- (3X^5+40X^3+45X)/5832N^2 \\ &+ (301X^7-1519X^5-32769X^3-79349X)/7873200N^3 + \cdots]\}^3\end{aligned}$$

where X is the normal deviate by taking the cube root of the polynomial expansion in Campbell [2]. For $N = 1$

$$\chi^2 = \{GAUSSD(\tfrac{1}{2}P)\}^2$$

and for $N = 2$

$$\chi^2 = -2 \ln (P).$$

For $2 < N < 2 + 4|X|$, χ^2 was fit with polynomials of the same form as the asymptotic approximation:

$$\begin{aligned}\chi^2 \cong N\{&(1.0000886-.2237368/N-.01513904/N^2) \\ &+ N^{-\frac{1}{2}}X(.4713941+.02607083/N-.008986007/N^2) \\ &+ N^{-1}X^2(.0001348028+.01128186/N+.02277679/N^2) \\ &+ N^{-3/2}X^3(-.008553069-.01153761/N-.01323293/N^2) \\ &+ N^{-2}X^4(.00312558+.005169654/N-.006950356/N^2) \\ &+ N^{-5/2}X^5(-.0008426812+.00253001/N+.001060438/N^2) \\ &+ N^{-3}X^6(.00009780499-.001450117/N+.001565326/N^2)\}^3\end{aligned}$$

from Abramowitz and Stegun [1] for $P = .0001, .0005, \ldots, .995$ and Hald and Sinkbaek [4] for $P = .999, .9995$. The deviates for $N = 3, 4, \ldots, 9$ were made accurate within 10^{-6} by using Algorithm 299 of Hill and Pike [6].

For $N = 1$ and $N = 2$ the χ^2 deviate is as accurate as the *GAUSSD* and *ALOG* procedure of the system. For $.0001 \leq P \leq .9995$ and $N \geq 3$ the absolute error in χ^2 is less than .005 and the relative error is less than .0003. This is some 100 to 1000 times as accurate as the Wilson-Hilferty formula even for large N. Error curves for three approximations are shown in Figures 1 and 2.

The program was tested on an IBM/360 at Rhode Island College and resulted in the output of Table I.

Table I.

Table of Computed Values

Deg. Fr.	0.9995	0.9950	0.5000	0.0010	0.0001
1	0.000000	0.000039	0.454933	10.827576	15.135827
2	0.001000	0.010025	1.386293	13.815512	18.420670
3	0.015312	0.071641	2.365390	16.268982	21.106873
4	0.063955	0.206904	3.356400	18.467987	23.510040
5	0.158168	0.411690	4.351295	20.515503	25.744583
10	1.264941	2.155869	9.341794	29.589081	35.565170
15	3.107881	4.601008	14.338853	37.697662	44.267853
20	5.398208	7.433892	19.337418	45.314896	52.387360
50	23.460876	27.990784	49.334930	86.660767	95.969482
100	59.895508	67.327621	99.334122	149.449051	161.319733

References

1. Abramowitz, M., and Stegun, I. (Eds.) *Handbook of Mathematical Functions*, Appl. Math. Ser. Vol. 55. Nat. Bur. Stand., U.S. Govt. Printing Office, Washington, D.C., 1965, pp. 984–985.
2. Campbell, G.A., Probability curves showing Poisson's exponential summation. *Bell Syst. J. 2* (1923), 95–113.
3. Cunningham, S.W. From normal integral to deviate. In *Applied Statistics*. Vol. 18, Royal Statis. Soc., 1969, pp. 290–293.
4. Hald, O.O., and Sinkbaek, O.O. *Skandinavisk Akturarietidskrift* (1950), 168–175.
5. Hastings, C. Jr. *Approximations for Digital Computers*. Princeton U. Press, Princeton, N.J., 1958, p. 192.

6. Hill, I.D., and Pike, M.C. Algorithm 299, Chi-squared integral. *Comm. ACM, 10*, 4 (Apr., 1967), 243–244.
7. Hilferty, M.M., and Wilson, E.B. *Proc. Nat. Acad. Sci., 17* (1931), 684.
8. Riordan, J. Inversion formulas in normal variable mapping. *Annals of Math. Statist. 20* (1949), 417–425.

Algorithm

```
      FUNCTION CHISQD(P, N)
      DIMENSION C(21), A(19)
      DATA C(1)/1.565326E-3/, C(2)/1.060438E-3/,
     * C(3)/-6.950356E-3/, C(4)/-1.323293E-2/,
     * C(5)/2.277679E-2/, C(6)/-8.986007E-3/,
     * C(7)/-1.513904E-2/, C(8)/2.530010E-3/,
     * C(9)/-1.450117E-3/, C(10)/5.169654E-3/,
     * C(11)/-1.153761E-2/, C(12)/1.128186E-2/,
     * C(13)/2.607083E-2/, C(14)/-0.2237368/,
     * C(15)/9.780499E-5/, C(16)/-8.426812E-4/,
     * C(17)/3.125580E-3/, C(18)/-8.553069E-3/,
     * C(19)/1.348028E-4/, C(20)/0.4713941/, C(21)/1.0000886/
      DATA A(1)/1.264616E-2/, A(2)/-1.425296E-2/,
     * A(3)/1.400483E-2/, A(4)/-5.886090E-3/,
     * A(5)/-1.091214E-2/, A(6)/-2.304527E-2/,
     * A(7)/3.135411E-3/, A(8)/-2.728484E-4/,
     * A(9)/-9.699681E-3/, A(10)/1.316872E-2/,
     * A(11)/2.618914E-2/, A(12)/-0.2222222/,
     * A(13)/5.406674E-5/, A(14)/3.483789E-5/,
     * A(15)/-7.274761E-4/, A(16)/3.292181E-3/,
     * A(17)/-8.729713E-3/, A(18)/0.4714045/, A(19)/1./
      IF (N-2) 10, 20, 30
   10 CHISQD = GAUSSD(.5*P)
      CHISQD = CHISQD*CHISQD
      RETURN
   20 CHISQD = -2.*ALOG(P)
      RETURN
   30 F = N
      F1 = 1./F
      T = GAUSSD(1.-P)
      F2 = SQRT(F1)*T
      IF (N.GE.(2+INT(4.*ABS(T)))) GO TO 40
      CHISQD=(((((((C(1)*F2+C(2))*F2+C(3))*F2+C(4))*F2
     * +C(5))*F2+C(6))*F2+C(7))*F1+((((((C(8)+C(9)*F2)*F2
     * +C(10))*F2+C(11))*F2+C(12))*F2+C(13))*F2+C(14)))*F1 +
     * (((((C(15)*F2+C(16))*F2+C(17))*F2+C(18))*F2
     * +C(19))*F2+C(20))*F2+C(21)
      GO TO 50
   40 CHISQD=(((A(1)+A(2)*F2)*F1+((A(3)+A(4)*F2)*F2
     * +A(5))*F2+A(6)))*F1+(((((A(7)+A(8)*F2)*F2+A(9))*F2
     * +A(10))*F2+A(11))*F2+A(12)))*F1 + (((((A(13)*F2
     * +A(14))*F2+A(15))*F2+A(16))*F2+A(17))*F2*F2
     * +A(18))*F2+A(19)
   50 CHISQD = CHISQD*CHISQD*CHISQD*F
      RETURN
      END
```

Algorithm 452

Enumerating Combinations of *m* Out of *n* Objects [G 6]

C.N. Liu and D.T. Tang [Recd. 7 July 1971 and 1 May 1972]
IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598

Key Words and Phrases: permutations, combinations
CR categories: 5.30
Language: Fortran

Description

NXCBN can be used to generate all combinations of *m* out of *n* objects. Let the binary *n*-vector of *m*1's and $(n - m)$ 0's representing a combination of *m* out of *n* objects be stored in an integer array, say *IC*(*n*). If *NXCBN* (*n, m, IC*) is called, a binary vector representing a new combination is made available in the array *IC*(*n*). If *NXCBN* (*n, m, IC*) is called $\binom{n}{m}$ times successively, then all combinations will be generated.

The algorithm has the following features; (a) each output binary *n*-vector differs from the input at exactly two positions—consequently each generated combination differs from the previous one by a single object: (b) the *n*-vectors generated by this subroutine form a closed loop of $\binom{n}{m}$ elements—therefore the initial combination may be specified arbitrarily, and the enumeration of any subset of $\binom{n}{m}$ combinations can be readily achieved. The second feature is not found in Chase's algorithm [1].

The algorithm underlying this procedure is based upon our study of properties of Gray codes. It can be shown that constant weight code vectors from a Gray code sequence are separated by a Hamming distance of 2. The mathematical analysis is contained in [2] and [3].

References

1. Chase, P.J. Algorithm 382, Combinations of *m* out of *n* objects. *Comm. ACM 13* (June 1970), 368.
2. Tang, D.T., and Liu, C.N. On enumerating *m* out of *n* combinations with minimal replacements. Proc. of Fifth Ann. Princeton Conf. on Info. Sci. and Sys., Mar. 1971.
3. Tang, D.T., and Liu, C.N. Distance-Two Cyclic Chaining of Constant-Weight Codes. *IEEETC. C-22*, 2 (Feb. 1973), 176–180.

Algorithm

```
      SUBROUTINE NXCBN(N, M, IC)
C EXPLANATION OF THE PARAMETERS IN THE CALLING SEQUENCE
C    N     THE TOTAL NUMBER OF OBJECTS
C    M     THE NUMBER OF OBJECTS TO BE TAKEN FROM N
C IF M=0, OR M>=N, EXIT WITH ARGUMENTS UNCHANGED
C    IC    AN INTEGER ARRAY.  IC CONTAINS AN N-DIMEN-
C          SIONAL BINARY VECTOR WITH M ELEMENTS SET TO 1
C          REPRESENTING THE M OBJECTS IN A COMBINATION
C THIS ALGORITHM IS PROGRAMMED IN ANSI STANDARD FORTRAN
      INTEGER IC(N)
C CHECK ENDING PATTERN OF VECTOR
      IF (M.GE.N .OR. M.EQ.0) GO TO 140
      N1 = N - 1
      DO 10 J=1,N1
        NJ = N - J
        IF (IC(N).EQ.IC(NJ)) GO TO 10
        J1 = J
        GO TO 20
   10 CONTINUE
   20 IF (MOD(M,2).EQ.1) GO TO 90
C FOR M EVEN
      IF (IC(N).EQ.1) GO TO 30
      K1 = N - J1
      K2 = K1 + 1
      GO TO 130
   30 IF (MOD(J1,2).EQ.1) GO TO 40
      GO TO 120
C SCAN FROM RIGHT TO LEFT
   40 JP = (N-J1) - 1
      DO 50 I=1,JP
        I1 = JP + 2 - I
        IF (IC(I1).EQ.0) GO TO 50
        IF (IC(I1-1).EQ.1) GO TO 70
        GO TO 80
   50 CONTINUE
   60 K1 = 1
      K2 = (N+1) - M
      GO TO 130
   70 K1 = I1 - 1
      K2 = N - J1
      GO TO 130
   80 K1 = I1 - 1
      K2 = (N+1) - J1
      GO TO 130
C FOR M ODD
   90 IF (IC(N).EQ.1) GO TO 110
      K2 = (N-J1) - 1
      IF (K2.EQ.0) GO TO 60
      IF (IC(K2+1).EQ.1 .AND. IC(K2).EQ.1) GO TO 100
      K1 = K2 + 1
      GO TO 130
  100 K1 = N
      GO TO 130
  110 IF (MOD(J1,2).EQ.1) GO TO 120
      GO TO 40
  120 K1 = N - J1
      K2 = MIN0((K1+2),N)
C COMPLEMENTING TWO BITS TO OBTAIN THE NEXT COMBINATION
  130 IC(K1) = 1 - IC(K1)
      IC(K2) = 1 - IC(K2)
  140 RETURN
      END
```

Algorithm 453

Gaussian Quadrature Formulas for Bromwich's Integral [D1]

Robert Piessens [Recd. 2 Aug. 1970 and 8 Feb. 1972]
Applied Mathematics Division, University of Leuven, Heverlee, Belgium

Key Words and Phrases: Gaussian quadrature, Bromwich's integral, complex integration, numerical inversion of the Laplace transform
CR Categories: 5.16, 5.13
Language: Fortran

Description

BROMIN calculates the abscissas $x_k^{(s)}$ and weights $w_k^{(s)}$ of the Gaussian quadrature formula

$$(1/2\pi j)\int_{c-j\infty}^{c+j\infty} e^{x}x^{-s}F(x)\,dx \simeq \sum_{k=1}^{N} w_k^{(s)}F(x_k^{(s)}) \qquad (1)$$

where c is an arbitrary real positive number, s is a real nonnegative parameter, and $F(x)$ must be analytic in the right-half plane of the complex plane.

Abscissas $x_k^{(s)}$ and weights $w_k^{(s)}$ are to be determined so that (1) is exact whenever $F(x)$ is a polynomial in x^{-1}, of degree $\leq 2N-1$.

The abscissas $x_k^{(s)}$ are the zeros of $P_{N,s}(x^{-1})$ where

$$P_{N,s}(u) = (-1)^N {}_2F_0(-N, N+s-1; \ -; \ u). \qquad (2)$$

Properties of $P_{N,s}(u)$ are studied in [1].

The quadrature formulas of even order have no real abscissas; those of odd order have exactly one real abscissa. All the abscissas have positive real parts and occur in complex conjugate pairs.

The zeros of (2) are calculated using Newton-Raphson's method. Finding an approximate zero as starting value for the iteration process is based on a certain regularity in the distribution of the zeros (see [1] and [2]). The starting values, used by *BROMIN* were tested for $s = 0.1(0.1)4.0$ and $N = 4(1)12$. Each abscissa was found to at least eight significant figures in at most six iteration steps.

The weights are given by

$$A_k = (-1)^{N-1}\frac{(N-1)!}{\Gamma(N+s-1)Nx_k^2}\left[\frac{2N+s-2}{P_{N-1,s}(x_k^{-1})}\right]^2 \qquad (3)$$

The polynomial (2) is evaluated by a three-term recurrence relation (see [1]). Due to roundoff errors, the accuracy of abscissas and weights decreases significantly for increasing N. In Table I we give for some values of s and N the moduli of the relative errors in the abscissas and weights, calculated by *BROMIN* (with $TOL = 0.1E-10$) on an IBM 370 computer in double precision (approximately 16 significant figures). For comparison we used the $16-S$ values given in [3].

Table I. Maximum Relative Errors in Abscissas and Weights

	Maximum error in abscissas		Maximum error in weights	
s	$N=6$	$N=12$	$N=6$	$N=12$
0.1	1.8×10^{-13}	1.7×10^{-9}	1.2×10^{-13}	2.3×10^{-8}
1.0	1.9×10^{-14}	5.3×10^{-11}	1.5×10^{-14}	6.4×10^{-10}
4.0	1.3×10^{-15}	2.3×10^{-12}	1.0×10^{-14}	4.3×10^{-11}

Note that the relative errors in the weights are larger than in the abscissas.

The use of complex arithmetic is avoided in *BROMIN* in order to facilitate the conversion to a double precision subroutine.

References

1. Piessens, R. Gaussian quadrature formulas for the numerical integration of Bromwich's integral and the inversion of the Laplace transform. *J. Eng. Math. 5* (Jan. 1971), 1–9.
2. Piessens, R. Some aspects of Gaussian quadrature formulas for the numerical inversion of the Laplace transform. *Comput. J. 14* (Nov. 1971), 433–435.
3. Piessens, R. Gaussian quadrature formulas for the numerical integration of Bromwich's integral and the inversion of the Laplace transform. Rep. TW1, Appl. Math. Div. U. of Leuven, 1969.

Algorithm

```
      SUBROUTINE BROMIN(N, S, TOL, XR, XI, WR, WI, EPS, IER)
      DOUBLE PRECISION AK, AN, ARG, CI, CR, D, D1, D2, E, EPS,
     * FAC, FACTI, FACTR, PI, PR, QI, QR, RI, RR, S, T1, T2,
     * TOL, U, V, WI, WR, XI, XR, YI, YR, Z
      INTEGER IER, J, K, L, N, N1, NUM, NUP, IGNAL
      DIMENSION XR(N), XI(N), WR(N), WI(N)
C  THIS SUBROUTINE CALCULATES ABSCISSAS AND WEIGHTS OF THE
C  GAUSSIAN QUADRATURE FORMULA OF ORDER N FOR THE BROMWICH
C  INTEGRAL.  ONLY THE ABSCISSAS OF THE FIRST QUADRANT OF
C  THE COMPLEX PLANE, THE REAL ABSCISSA (IF N IS ODD) AND
C  THE CORRESPONDING WEIGHTS ARE CALCULATED.  THE OTHER
C  ABSCISSAS AND WEIGHTS ARE COMPLEX CONJUGATES.
C  INPUT PARAMETERS
C       N   ORDER OF THE QUADRATURE FORMULA.
C           N MUST BE GREATER THAN 2.
C       TOL REQUESTED RELATIVE ACCURACY OF THE ABSCISSAS.
C       S   PARAMETER OF THE WEIGHT FUNCTION.
C  OUTPUT PARAMETERS
C       XR AND XI CONTAIN THE REAL AND IMAGINARY PARTS OF
C          THE ABSCISSAS.  IF N IS ODD, THE REAL ABSCISSA
C          IS XR(1).
C       WR AND WI CONTAIN THE REAL AND IMAGINARY PARTS OF
C          THE CORRESPONDING WEIGHTS.
C       EPS IS A CRUDE ESTIMATION OF THE OBTAINED RELATIVE
C          ACCURACY OF THE ABSCISSAS.
C       IER IS AN ERROR CODE.
C          IF IER=0    THE COMPUTATION IS CARRIED OUT TO
C                      THE REQUESTED ACCURACY.
C          IF IER.GT.0 THE IER-TH ABSCISSA IS NOT FOUND.
C          IF IER=-1   THE COMPUTATIONS ARE CARRIED OUT,
C                      BUT THE REQUESTED ACCURACY IS NOT
C                      ACHIEVED.
C          IF IER=-2   N IS LESS THAN 3.
C  FUNCTION PROGRAMS REQUIRED
C       FUNCTION GAMMA(X) WHICH EVALUATES THE GAMMA
C          FUNCTION FOR POSITIVE X.
      IER = -2
      IF (N.LT.3) RETURN
      N1 = (N+1)/2
      L = N - 1
      AN = N
      IER = 0
      EPS = TOL
      ARG = 0.034D0*(30.D0+AN+AN)/(AN-1.D0)
      FACTR = DCOS(ARG)
      FACTI = DSIN(ARG)
      FAC = 1.D0
      AK = 0.D0
      DO 10 K=1,L
        AK = AK + 1.D0
        FAC = -FAC*AK
   10 CONTINUE
      FAC = FAC*(AN+AN+S-2.D0)**2/(AN*DGAMMA(AN+S-1.D0))
C  CALCULATION OF AN APPROXIMATION OF THE FIRST ABSCISSA
      YR = 1.333D0*AN + S - 1.5D0
      YI = 0.0D0
      IF (N-N1-N1) 30, 20, 20
   20 YI = YI + 1.6D0 + 0.07D0*S
C  START MAIN LOOP
   30 DO 140 K=1,N1
        E = TOL
        IGNAL = 0
        NUM = 0
        NUP = 0
C  NEWTON-RAPHSON METHOD
        D = YR*YR + YI*YI
        YR = YR/D
        YI = -YI/D
        GO TO 50
   40   IGNAL = 1
   50   QR = S*YR - 1.D0
        QI = S*YI
        PR = (S+1.D0)*((S+2.D0)*(YR*YR-YI*YI)-2.D0*YR) + 1.D0
        PI = 2.D0*(S+1.D0)*YI*((S+2.D0)*YR-1.D0)
        Z = 2.D0
        DO 60 J=3,N
          RR = QR
          RI = QI
          QR = PR
          QI = PI
          Z = Z + 1.D0
          U = Z + S - 2.D0
          V = U + Z
```

```
          D = (V*YR+(2.D0-S)/(V-2.D0))/U
          D1 = (Z-1.D0)*V/(U*(V-2.D0))
          D2 = V*YI/U
          PR = (V-1.D0)*(0R*D-0I*D2) + D1*RR
          PI = (V-1.D0)*(0I*D+0R*D2) + D1*RI
   60     C0NTINUE
          IF (IGNAL.EQ.1) G0 T0 100
          D = (YR*YR+YI*YI)*V
          D1 = ((PR+0R)*YR+(PI+0I)*YI)/D + PR
          D2 = ((PI+0I)*YR-(PR+0R)*YI)/D + PI
          D = (D1*D1+D2*D2)*AN
          T1 = PR*YR - PI*YI
          T2 = PI*YR + PR*YI
          CR = (T1*D1+T2*D2)/D
          CI = (T2*D1-T1*D2)/D
          YR = YR - CR
          YI = YI - CI
          NUM = NUM + 1
C  TEST 0F C0NVERGENCE 0F ITERATI0N PR0CESS
          IF (CR*CR+CI*CI-E*E*(YR*YR+YI*YI)) 40, 40, 70
C  TEST 0F NUMBER 0F ITERATI0N STEPS
   70     IF (NUM-10) 50, 50, 80
   80     E = E*10.D0
          IER = -1
          NUP = NUP + 1
          IF (NUP-5) 50, 50, 90
   90     IER = K
          RETURN
C  CALCULATI0N 0F WEIGHTS
  100     IF (EPS.GE.E) G0 T0 110
          EPS = E
  110     D = (0R*0R+0I*0I)**2
          D1 = YR*0R + YI*0I
          D2 = YI*0R - YR*0I
          WR(K) = FAC*(D1*D1-D2*D2)/D
          WI(K) = 2.D0*FAC*D2*D1/D
          D = YR*YR + YI*YI
          XR(K) = YR/D
          XI(K) = -YI/D
          IF (K+1-N1) 130, 120, 150
  120     FACTR = DC0S(1.5D0*ARG)
          FACTI = DSIN(1.5D0*ARG)
C  CALCULATI0N 0F AN APPR0XIMATI0N 0F THE (K+1)-TH ABSCISSA
  130     YR = (XR(K)+0.67D0*AN)*FACTR - XI(K)*FACTI - 0.67D0*AN
          YI = (XR(K)+0.67D0*AN)*FACTI + XI(K)*FACTR
  140 C0NTINUE
  150 RETURN
      END
```

Algorithm 454

The Complex Method for Constrained Optimization [E4]

Joel A. Richardson and J.L. Kuester* [Rec'd. Dec. 22, 1970 and May 5, 1971]
Arizona State University, Tempe, AZ 85281

Key Words and Phrases: optimization, constrained optimization, Box's algorithm
CR Categories: 5.41
Language: Fortran

Description

Purpose. This program finds the maximum of a multivariable, nonlinear function subject to constraints:

Maximize $F(X_1, X_2, \ldots, X_N)$
Subject to $G_k \leq X_k \leq H_k, \quad k = 1, 2, \ldots, M.$

The implicit variables $X_{N+1}, \ldots, X_M$ are dependent functions of the explicit independent variables $X_1, X_2, \ldots, X_N$. The upper and lower constraints H_k and G_k are either constants or functions of the independent variables.

Method. The program is based on the "complex" method of M.J. Box [2]. This method is a sequential search technique, which has proven effective in solving problems with nonlinear objective functions subject to nonlinear inequality constraints. No derivatives are required. The procedure should tend to find the global maximum because the initial set of points is randomly scattered throughout the feasible region. If linear constraints are present or equality constraints are involved, other methods should prove to be more efficient [1]. The algorithm proceeds as follows:

(1) An original "complex" of $K \geq N + 1$ points is generated consisting of a feasible starting point and $K - 1$ additional points generated from random numbers and constraints for each of the independent variables: $X_{i,j} = G_i + r_{i,j}(H_i - G_i)$, $i = 1, 2, \ldots, N$, and $j = 1, 2, \ldots, K - 1$, where $r_{i,j}$ are random numbers between 0 and 1.

(2) The selected points must satisfy both the explicit and implicit constraints. If at any time the explicit constraints are violated, the point is moved a small distance δ inside the violated limit. If an implicit constraint is violated, the point is moved one half of the distance to the centroid of the remaining points: $X_{i,j}(\text{new}) = (X_{i,j}(\text{old}) + \bar{X}_{i,c})/2$, $i = 1, 2, \ldots, N$, where the coordinates of the centroid of the remaining points, $\bar{X}_{i,c}$, are defined by

$$\bar{X}_{i,c} = \frac{1}{K-1}\left[\sum_{j=1}^{K} X_{i,j} - X_{i,j}(\text{old})\right], \quad i = 1, 2, \ldots, N.$$

This process is repeated as necessary until all the implicit constraints are satisfied.

(3) The objective function is evaluated at each point. The point having the lowest function value is replaced by a point which is located at a distance α times as far from the centroid of the remaining points as the distance of the rejected point on the line joining the rejected point and the centroid:

$$X_{i,j}(\text{new}) = \alpha(\bar{X}_{i,c} - X_{i,j}(\text{old})) + \bar{X}_{i,c}, \quad i = 2, \ldots, N.$$

Box [2] recommends a value of $\alpha = 1.3$.

(4) If a point repeats in giving the lowest function value on consecutive trials, it is moved one half the distance to the centroid of the remaining points.

(5) The new point is checked against the constraints and is adjusted as before if the constraints are violated.

(6) Convergence is assumed when the objective function values at each point are within β units for γ consecutive iterations.

Program. The program consists of three general subroutines (*JCONSX, JCEK*1, *JCENT*) and two user supplied subroutines (*JFUNC, JCNST*1). The use of the program and the meaning of the parameters are described in the comments at the beginning of subroutine *JCONSX*. All communication between the main program and subroutines is achieved in the subroutine argument lists. An iteration is defined as the calculations required to select a new point which satisfies the constraints and does not repeat in yielding the lowest function value.

Test results. Several functions were chosen to test the program. The calculations were performed on a CDC 6400 computer. Some examples:

1. Box Problem [2]
   Function: $F = (9 - (X_1 - 3)^2)X_2^3/27\sqrt{3}$
   Constraints: $0 \leq X_1 \leq 100$
   $0 \leq X_2 \leq X_1/\sqrt{3}$
   $0 \leq (X_3 = X_1 + \sqrt{3}X_2) \leq 6$
   Starting point: $X_1 = 1.0$, $X_2 = 0.5$
   Parameters: $K = 4$, $\alpha = 1.3$, $\beta = .001$, $\gamma = 5$, $\delta = .0001$

Computed results	Correct results:
$F = 1.0000$	$F = 1.0000$
$X_1 = 3.0000$	$X_1 = 3.0000$
$X_2 = 1.7320$	$X_2 = 1.7321$

   Number of iterations: 68
   Central processor time: 6 sec.

* The authors acknowledge financial support from a National Science Foundation summer fellowship and Arizona State University Grants Committee fellowship. Computer facilities were provided by the Arizona State University Computer Center and AiResearch Manufacturing Company.

2. Post Office Problem [3]

Function: $F = X_1X_2X_3$

Constraints: $0 \leq X_i \leq 42, \quad i = 1, 2, 3$

$0 \leq (X_4 = X_1 + 2X_2 + 2X_3) \leq 72$

Starting point: $X_1 = 1.0$, $X_2 = 1.0$, $X_3 = 1.0$

Parameters: $K = 6$, $\alpha = 1.3$, $\beta = .01$, $\gamma = 5$, $\delta = .0001$

Computed results:	Correct results:
$F = 3456$	$F = 3456$
$X_1 = 24.01$	$X_1 = 24.00$
$X_2 = 12.00$	$X_2 = 12.00$
$X_3 = 12.00$	$X_3 = 12.00$

Number of iterations: 72

Central processor time: 6 sec.

3. Beveridge and Schechter Problem [1]

Function: $F = -(X_1 - 0.5)^2 - (X_2 - 1.0)^2$

Constraints: $-2 \leq X_1 \leq 2$

$-\sqrt{2} \leq X_2 \leq \sqrt{2}$

$-4 \leq (X_3 = X_1^2 + 2X_2^2 - 4) \leq 0$

Starting point: $X_1 = 0.$, $X_2 = 0.$

Parameters: $K = 4$, $\alpha = 1.3$, $\beta = .00001$, $\gamma = 5$, $\delta = .0001$

Computed results:	Correct results:
$F = .0000$	$F = .0000$
$X_1 = .5035$	$X_1 = .5000$
$X_2 = .9990$	$X_2 = 1.0000$

Number of iterations: 40

Central processor time = 5 sec.

References

1. Beveridge, G.S., and Schechter, R.S. *Optimization: Theory and Practice*. McGraw-Hill, New York, 1970.

2. Box, M.J. A new method of constrained optimization and a comparison with other methods. *Comp. J. 8* (1965), 42–52.

3. Rosenbrock, H.H. An automatic method for finding the greatest or least value of a function. *Comp. J. 3* (1960), 175–184.

Algorithm

```
      SUBROUTINE JCONSX(N, M, K, ITMAX, ALPHA, BETA, GAMMA,
     * DELTA, X, R, F, IT, IEV2, K0, G, H, XC, L)
C PURPOSE
C   TO FIND THE CONSTRAINED MAXIMUM OF A FUNCTION OF
C   SEVERAL VARIABLES BY THE COMPLEX METHOD OF M. J. BOX.
C   THIS IS THE PRIMARY SUBROUTINE AND COORDINATES THE
C   SPECIAL PURPOSE SUBROUTINES (JCEK1, JCENT, JFUNC,
C   JCNST1).  INITIAL GUESSES OF THE INDEPENDENT VARIABLES,
C   RANDOM NUMBERS, SOLUTION PARAMETERS, DIMENSION LIMITS
C   AND PRINTER CODE DESIGNATION ARE OBTAINED FROM THE MAIN
C   PROGRAM.  FINAL FUNCTION AND INDEPENDENT VARIABLE
C   VALUES ARE TRANSFERRED TO THE MAIN PROGRAM FOR
C   PRINTOUT.  INTERMEDIATE PRINTOUTS ARE PROVIDED IN THIS
C   SUBROUTINE. THE USER MUST PROVIDE THE MAIN PROGRAM AND
C   THE SUBROUTINES THAT SPECIFY THE FUNCTION (JFUNC) AND
C   CONSTRAINTS (JCNST1). FORMAT CHANGES MAY BE REQUIRED
C   WITHIN THIS SUBROUTINE DEPENDING ON THE PARTICULAR
C   PROBLEM UNDER CONSIDERATION.
C USAGE
C   CALL JCONSX(N,M,K,ITMAX,ALPHA,BETA,GAMMA,DELTA,X,R,F,
C  1IT,IEV2,K0,G,H,XC,L)
C SUBROUTINES REQUIRED
C   JCEK1(N,M,K,X,G,H,I,KODE,XC,DELTA,L,K1)
C     CHECKS ALL POINTS AGAINST EXPLICIT AND IMPLICIT
C     CONSTRAINTS AND APPLYS CORRECTION IF VIOLATIONS ARE
C     FOUND
C   JCENT(N,M,K,IEV1,I,XC,X,L,K1)
C     CALCULATES THE CENTROID OF POINTS
C   JFUNC(N,M,K,X,F,I,L)
C     SPECIFIES OBJECTIVE FUNCTION (USER SUPPLIED)
C   JCNST1(N,M,K,X,G,H,I,L)
C     SPECIFIES EXPLICIT AND IMPLICIT CONSTRAINT LIMITS
C     (USER SUPPLIED).  ORDER EXPLICIT CONSTRAINTS FIRST
C DESCRIPTION OF PARAMETERS
C   N      NUMBER OF EXPLICIT INDEPENDENT VARIABLES - DEFINE
C          IN MAIN PROGRAM
C   M      NUMBER OF SETS OF CONSTRAINTS - DEFINE IN MAIN
C          PROGRAM
C   K      NUMBER OF POINTS IN THE COMPLEX - DEFINE IN MAIN
C          PROGRAM
C   ITMAX MAXIMUM NUMBER OF ITERATIONS - DEFINE IN MAIN
C          PROGRAM
C   ALPHA REFLECTION FACTOR - DEFINE IN MAIN PROGRAM
C   BETA  CONVERGENCE PARAMETER -  DEFINE IN MAIN PROGRAM
C   GAMMA CONVERGENCE PARAMETER - DEFINE IN MAIN PROGRAM
C   DELTA EXPLICIT CONSTRAINT VIOLATION CORRECTION - DEFINE
C          IN MAIN PROGRAM
C   X      INDEPENDENT VARIABLES - DEFINE INITIAL VALUES IN
C          MAIN PROGRAM
C   R      RANDOM NUMBERS BETWEEN 0 AND 1 - DEFINE IN MAIN
C          PROGRAM
C   F      OBJECTIVE FUNCTION - DEFINE IN SUBROUTINE JFUNC
C   IT     ITERATION INDEX - DEFINED IN SUBROUTINE JCONSX
C   IEV2   INDEX OF POINT WITH MAXIMUM FUNCTION VALUE -
C          DEFINED IN SUBROUTINE JCONSX
C   IEV1   INDEX OF POINT WITH MINIMUM FUNCTION VALUE -
C          DEFINED IN SUBROUTINE JCONSX AND JCEK1
C   K0     PRINTER UNIT NUMBER - DEFINE IN MAIN PROGRAM
C   G      LOWER CONSTRAINT - DEFINE IN SUBROUTINE JCNST1
C   H      UPPER CONSTRAINT - DEFINE IN SUBROUTINE JCNST1
C   XC     CENTROID - DEFINED IN SUBROUTINE JCENT
C   L      TOTAL NUMBER OF INDEPENDENT VARIABLES (EXPLICIT +
C          IMPLICIT) - DEFINE IN MAIN PROGRAM
C   I      POINT INDEX - DEFINED IN SUBROUTINE JCONSX
C   KODE   KEY USED TO DETERMINE IF IMPLICIT CONSTRAINTS ARE
C          PROVIDED - DEFINED IN SUBROUTINE JCONSX AND JCEK1
C   K1     DO LOOP LIMIT - DEFINED IN SUBROUTINE JCONSX
      DIMENSION X(K,L), R(K,N), F(K), G(M), H(M), XC(N)
      INTEGER GAMMA
      IT = 1
      WRITE (K0,99995) IT
      KODE = 0
      IF (M-N) 20, 20, 10
   10 KODE = 1
   20 CONTINUE
      DO 40 II=2,K
        DO 30 J=1,N
          X(II,J) = 0.
   30   CONTINUE
   40 CONTINUE
C CALCULATE COMPLEX POINTS AND CHECK AGAINST CONSTRAINTS
      DO 60 II=2,K
        DO 50 J=1,N
          I = II
          CALL JCNST1(N, M, K, X, G, H, I, L)
          X(II,J) = G(J) + R(II,J)*(H(J)-G(J))
   50   CONTINUE
        K1 = II
        CALL JCEK1(N, M, K, X, G, H, I, KODE, XC, DELTA, L, K1)
        WRITE (K0,99999) II, (X(II,J),J=1,N)
   60 CONTINUE
      K1 = K
      DO 70 I=1,K
        CALL JFUNC(N, M, K, X, F, I, L)
   70 CONTINUE
      KOUNT = 1
      IA = 0
C FIND POINT WITH LOWEST FUNCTION VALUE
      WRITE (K0,99998) (F(I),I=1,K)
   80 IEV1 = 1
      DO 100 ICM=2,K
        IF (F(IEV1)-F(ICM)) 100, 100, 90
   90   IEV1 = ICM
  100 CONTINUE
C FIND POINT WITH HIGHEST FUNCTION VALUE
      IEV2 = 1
      DO 120 ICM=2,K
        IF (F(IEV2)-F(ICM)) 110, 110, 120
  110   IEV2 = ICM
  120 CONTINUE
C CHECK CONVERGENCE CRITERIA
      IF (F(IEV2)-(F(IEV1)+BETA)) 140, 130, 130
  130 KOUNT = 1
      GO TO 150
  140 KOUNT = KOUNT + 1
      IF (KOUNT-GAMMA) 150, 240, 240
C REPLACE POINT WITH LOWEST FUNCTION VALUE
  150 CALL JCENT(N, M, K, IEV1, I, XC, X, L, K1)
      DO 160 J=1,N
        X(IEV1,J) = (1.+ALPHA)*(XC(J)) - ALPHA*(X(IEV1,J))
  160 CONTINUE
      I = IEV1
      CALL JCEK1(N, M, K, X, G, H, I, KODE, XC, DELTA, L, K1)
      CALL JFUNC(N, M, K, X, F, I, L)
C REPLACE NEW POINT IF IT REPEATS AS LOWEST FUNCTION VALUE
  170 IEV2 = 1
      DO 190 ICM=2,K
        IF (F(IEV2)-F(ICM)) 190, 190, 180
  180   IEV2 = ICM
  190 CONTINUE
      IF (IEV2-IEV1) 220, 200, 220
  200 DO 210 JJ=1,N
        X(IEV1,JJ) = (X(IEV1,JJ)+XC(JJ))/2.
  210 CONTINUE
      I = IEV1
      CALL JCEK1(N, M, K, X, G, H, I, KODE, XC, DELTA, L, K1)
      CALL JFUNC(N, M, K, X, F, I, L)
      GO TO 170
  220 CONTINUE
      WRITE (K0,99997) (X(IEV1,JB),JB=1,N)
      WRITE (K0,99998) (F(I),I=1,K)
      WRITE (K0,99996) (XC(J),J=1,N)
      IT = IT + 1
      IF (IT-ITMAX) 230, 230, 240
  230 CONTINUE
      WRITE (K0,99995) IT
      GO TO 80
  240 RETURN
99999 FORMAT(1H , 15X, 21H COORDINATES AT POINT, I4/8(F8.4, 2X))
99998 FORMAT(1H , 20X, 16H FUNCTION VALUES, /8(F10.4, 2X))
99997 FORMAT(1H , 20X, 16H CORRECTED POINT, /8(F8.4, 2X))
99996 FORMAT(1H , 21H CENTROID COORDINATES, 2X, 8(F8.4, 2X))
99995 FORMAT(1H , //10H ITERATION, 4X, I5)
      END

      SUBROUTINE JCEK1(N, M, K, X, G, H, I, KODE, XC, DELTA, L,
     * K1)
C PURPOSE
C   TO CHECK ALL POINTS AGAINST THE EXPLICIT AND IMPLICIT
C   CONSTRAINTS AND TO APPLY CORRECTIONS IF VIOLATIONS ARE
C   FOUND
C USAGE
C   CALL JCEK1(N,M,K,X,G,H,I,KODE,XC,DELTA,L,K1)
C SUBROUTINES REQUIRED
C   JCENT(N,M,K,IEV1,I,XC,X,L,K1)
C   JCNST1(N,M,K,X,G,H,I,L)
C DESCRIPTION OF PARAMETERS
C   PREVIOUSLY DEFINED IN SUBROUTINE JCONSX
```

```
      DIMENSION X(K,L), G(M), H(M), XC(N)
   10 KT = 0
      CALL JCNSTI(N, M, K, X, G, H, I, L)
C CHECK AGAINST EXPLICIT CONSTRAINTS
      DO 50 J=1,N
        IF (X(I,J)-G(J)) 20, 20, 30
   20   X(I,J) = G(J) + DELTA
        GO TO 50
   30   IF (H(J)-X(I,J)) 40, 40, 50
   40   X(I,J) = H(J) - DELTA
   50 CONTINUE
      IF (KODE) 110, 110, 60
C CHECK AGAINST THE IMPLICIT CONSTRAINTS
   60 CONTINUE
      NN = N + 1
      DO 100 J=NN,M
        CALL JCNSTI(N, M, K, X, G, H, I, L)
        IF (X(I,J)-G(J)) 80, 70, 70
   70   IF (H(J)-X(I,J)) 80, 100, 100
   80   IEV1 = I
        KT = 1
        CALL JCENT(N, M, K, IEV1, I, XC, X, L, K1)
        DO 90 JJ=1,N
          X(I,JJ) = (X(I,JJ)+XC(JJ))/2.
   90   CONTINUE
  100 CONTINUE
      IF (KT) 110, 110, 10
  110 RETURN
      END

      SUBROUTINE JCENT(N, M, K, IEV1, I, XC, X, L, K1)
C PURPOSE
C   TO CALCULATE THE CENTROID OF POINTS
C USAGE
C   CALL JCENT(N,M,K,IEV1,I,XC,X,L,K1)
C SUBROUTINES REQUIRED
C   NONE
C DESCRIPTION OF PARAMETERS
C   PREVIOUSLY DEFINED IN SUBROUTINE JCONSX
      DIMENSION X(K,L), XC(N)
      DO 20 J=1,N
        XC(J) = 0.
        DO 10 IL=1,K1
          XC(J) = XC(J) + X(IL,J)
   10   CONTINUE
        RK = K1
        XC(J) = (XC(J)-X(IEV1,J))/(RK-1.)
   20 CONTINUE
      RETURN
      END
```

Certification and Remark on Algorithm 404 [S14]

Complex Gamma Function [C.W. Lucas Jr. and C.W. Terril, *Comm. ACM 14* (Jan. 1971), 48]

G. Andrejková and J. Vinař, Computing Center, Šafarik University, Košice, Czechoslovakia

The following changes were made in the algorithm:

a. The function subroutine heading was changed to read

COMPLEX FUNCTION CGAMMA(Z)

in accordance with the standard.

b. The convergence tests following statement number 70 involve the computation of the quantity *REAL(TERM)/REAL(SUM)*. This can lead to overflow if Z is real and near to a pole. For these reasons the two statements were replaced by

*IF (ABS(REAL(TERM)) .GE. TOL*ABS(REAL(SUM))) GO TO* 80

and

*IF (ABS(AIMAG(TERM)) .GE. TOL*ABS(AIMAG(SUM))) GO TO* 100

c. For similar reasons the statement

*SUM = CLOG(PI/CSIN(PI*Z))−SUM*

was replaced by

*SUM = CLOG(PI)−CLOG(CSIN(PI*Z))−SUM*

With these modifications the algorithm was translated on MINSK 22M using the FEL Fortran compiler (with seven significant digits in single precision and 15 in double precision) and ran satisfactorily.

The following tests were performed:

a. The logarithms of *CGAMMA(Z)* for $z = x+iy$ with $x = 1.0(0.1)10.0$ and $y = 0.0(0.1)3.0$ were checked against the values given in [1]. An overall accuracy of five to six digits was observed. The imaginary part frequently had one more accurate digit than the real part.

b. The behavior in the vicinity of poles was tested by computing the values of *CGAMMA(Z)* in eight evenly spaced points on circles of decreasing diameter. The value of $1.E-7$ for the minimum diameter was found adequate.

c. The values of *CGAMMA(Z)* were computed for $z = x+iy$ with

1. $x = 0.0(1.0)23.0$, $y = 0.0$
2. $x = 0.0$, $y = 0.0(1.0)26.0$
3. $x = y = 0.0(1.0)25.0$
4. $x = -y = 0.0(1.0)25.0$
5. $-x = y = 0.0(1.0)12.0$
6. $-x = -y = 0.0(1.0)12.0$

in all cases the final value is the last for which the program did not run into overflow or, in the last two cases, try to take a logarithm of too small a number.

References

1. Table of gamma function for complex arguments. National Bureau of Standards, Applied Math. Series 34, August 1954.

Remark on Algorithm 357 [A1]

An Efficient Prime Number Generator [Richard C. Singleton, *Comm. ACM 10* (October, 1969), 563]

Richard M. De Morgan [Recd 8 August 1972], Digital Equipment Co. Ltd., Reading, England

On some Algol 60 implementations, the value of *ni* is destroyed between subsequent calls to the procedure. The second and third lines of the algorithm should be changed to make *ni* an **own integer**:

own integer *ij, ik, inc, j, ni, nj*;

integer *i, jqi, k*;

Remark on Algorithm 412 [J6]

Graph Plotter [Joseph Cermak, *Comm. ACM 14* (July 1971), 492–493]

Richard P. Watkins [Recd. 31 Jan. 1972], Mathematics Department, Royal Melbourne Institute of Technology, Melbourne, Australia 3000

This algorithm is not functionally identical to Algorithm 278 as claimed. If the $x[i]$ values are not uniformly spaced or if $m > L$, it is possible for two or more of them to correspond to the same printer line. In this case, the array *ind* will contain only the largest of the values of i and only one set of $y[i, j]$ values, corresponding to that value of i, will be plotted.

The array *ind* is redundant. The following changes enable *plotL* to take over the functions of *ind* (where all line numbers refer to lines relative to the label *escape*):

a. Line 4. Replace

for $i := 1$ **step 1 until** L **do** *plotL*[i] := 1

by

for $i := 1$ **step** 1 **until** L **do** $plotL[i] := 0$

b. Line 9. Replace

$plotL[r] := 0$; $ind[r] := i$

by

$plotL[r] := i$

c. Line 21. Replace

if $plotL[i] = 0$ **then**

by

if $plotL[i] > 0$ **then**

d. Line 24. Replace

$plotS\ [1 + entier(0.5 + q \times (y[ind[i]] - ymin))] := 3$

by

$plotS\ [1 + entier(0.5 + q \times (y[plotL[i]] - ymin))] := 3$

e. Line 27. Replace

$plotS\ [1 + entier(0.5 + q \times (y[ind[i],j] - ymin))] := j + 2$

by

$plotS[1 + entier(0.5 + q \times (y[plotL[i],j] - ymin))] := j + 2$

(The referee has noted that there is a typographical error on the fifth line before the line labeled *escape*. Replace

for $j :=$ **step** 1 **until** n **do**

by

for $j :=$ 1 **step** 1 **until** n **do**

He has also noted that the array declaration for *ind* should be deleted if the above changes are made.—L.D.F.)

Remark on Algorithm 424 [D 1]

Clenshaw-Curtis Quadrature [W.M.Gentleman, *Comm. ACM 15* (May 1972), 353–355.]

Albert J. Good [Recd. 19 December 1972] Systems, Science and Software, La Jolla, CA 92037

As published, this algorithm will not execute correctly under some compilers (e.g. Fortran V in the Univac 1108). One minor change is sufficient for proper operation: replace the variable *J REV* by the index *J8* inside the *DO* 120 loop.

The appearance of *J REV* and *J8* in an *EQUIVALENCE* statement is not meaningful since the memory location associated with a *DO* loop index does not always contain the current value of the index (this depends on the compiler).

Remark on Algorithm 428 [Z]

Hu-Tucker Minimum Redundancy Alphabetic Coding Method [J.M. Yohe, *Comm. ACM 15* (May 1972), 360–362]

J.G. Byrne [Recd. 26 June 1972] Department of Computer Science, Trinity College, Dublin 2, Ireland

Algorithm 428 was translated into Basic Fortran IV and run on IBM System 360/44 running under *RAX*. When the line just after the label *B2*:

if $i1 > n$ **then go to** *E1* **else**

was changed to

if $i > n$ **then go to** *E1* **else**

the algorithm gave correct results for the example given and for the example in Gilbert and Moore [1]. In the latter case the cost defined as

$$\frac{\sum_{i=1}^{N} Q(I)*L(I)}{\sum_{i=1}^{N} Q(I)}$$

and code lengths were correct.

When the L array was set to 1's on entry, the optimum (Huffman) codes were obtained, and they were the same as those given by the Schwartz and Kallick [2] method as claimed in the author's description.

Table I.

Size of alphabet	10	27	60
Time to find optimum alphabetic codes (secs)	0·02	0·14	0·62
Time to find optimum codes (secs)	0·02	0·08	0·34

Table I, which gives the cpu time required, shows that the algorithm is very fast for small alphabets and that the time is approximately proportional to n^2, as expected.

References

1. Gilbert, E.N., and Moore, E.F. Variable length binary encodings. *Bell Systems Tech. J. 38* (1959), 933–968.
2. Schwartz, E.S., and Kallick, B. Generating a canonical prefix encoding. *Comm. ACM 7* (Mar. 1964), 166–169.

Remark on Algorithm 429 [C2]

Localization of the Roots of a Polynomial [W. Squire, *Comm. ACM 15* (Aug. 1972), 776]

Edward J. Williams [Recd. 15 Sept. 1972] Computer Science Department, Ford Motor Company, P.O. Box 2053, Dearborn, MI 48121

Corrections are needed in the third paragraph. The theorem that the positive real roots of (1) are less than

$1 + [\max_{1 \le i \le n} |Ci|]^{1/m}$. . . should read

$1 + [\max_{1 \le i \le n\ Ci<0} |Ci|]^{1/m}$

Further, the four words "*RADIUS*" in this paragraph should be replaced by "*BOUND*".

References

1. Zaguskin, O.O. *Solution of Algebraic and Transcendental Equations*, Pergamon Press, New York, 1961, p. 21.