Algorithms supplement

Algorithm 102

CONVOLUTION INTEGRALS INVOLVING PROBABILITY DISTRIBUTION FUNCTIONS

D. J. McConalogue
Department of Computer Science
University College, Dublin 4

Author's note

The subroutines which follow have been developed to calculate certain convolution integrals involving probability distribution functions for use when closed analytical forms of the integrals do not exist or are computationally awkward. The integrals are of interest in applied probability theory, for instance, in certain expressions for systems characteristics studied in reliability theory. Two calculations are required and these can be conveniently expressed in terms of the standard notation for the convolution of two functions G and H, viz.

$$(G * H)(t) \equiv \int_0^t G(t - x)H'(x)\,dx \tag{1}$$

where H' is the derivative of H.

The first is the generation of approximations to the family of functions $F^{(n)}$ defined recursively by

$$F^{(n)}(t) = (F^{(n-1)} * F)(t) \quad n \geqslant 1 \tag{2}$$
$$= 1 \qquad\qquad n = 0$$

where F is a probability distribution function, and F' is assumed here to be given explicitly by a probability density function $f \in C^1[0, \infty)$. The $F^{(n)}$, being distribution functions, are positive and monotonic increasing with t. A widely used f for which closed forms do not exist for $n > 1$ is that of the Weibull distribution x,

$$f(t) = \lambda\alpha t^{\alpha-1}\exp(-\lambda t^{\alpha}),\ 0 \leqslant t < \infty,\ \lambda > 0,\ \alpha > 0. \tag{3}$$

The second is the calculation of an approximation to the integral

$$(G^{(r)} * H^{(s)})(t) \tag{4}$$

where $G^{(r)}$ and $H^{(s)}$ are the rth and sth members of families generated from the distribution functions G and H according to the recursive scheme (2).

The programs which follow are FORTRAN codings of an algorithm of Cleroux and McConalogue (1976) which appears to be the first for the problem. The heart of the algorithm is the choice of cubic spline interpolation to give a continuous representation of the approximation to $F^{(n)}$. This representation preserves the positivity and monotonicity of the function and provides a good approximation to both the function and its first derivative. In the form presented here, the approximation to $F^{(n)}$ is given by a set of $(m + 1)$ tabulated values $F_j^{(n)}$ at the points $jh, j = 0, 1, \ldots, m$, together with an associated set of $(m + 1)$ spline coefficients $C_j^{(n)}$. In the range $jh \leqslant t \leqslant jh + h$, $F^{(n)}$ is approximated by $y_j^{(n)}$, where

$$y_j^{(n)}(z) = \frac{(h - z)^3}{6h}C_j^{(n)} + \frac{z^3}{6h}C_{j+1}^{(n)} + (h - z)A_j^{(n)} + zA_{j+1}^{(n)}, \tag{5}$$

$$z = t - jh \text{ and } A_j^{(n)} = F_j^{(n)}/h - hC_j^{(n)}/6$$

The $C^{(n)}$ are obtained as solutions to a set of $(m + 1)$ simultaneous equations, $(m - 1)$ of the form

$$\tfrac{1}{4}C_{j-1}^{(n)} + C_j^{(n)} + \tfrac{1}{4}C_{j+1}^{(n)} = \frac{3}{2h^2}(F_{j-1}^{(n)} - 2F_j^{(n)} + F_{j+1}^{(n)}),$$
$$j = 1, \ldots, m - 1 \tag{6}$$

which arise from imposing first derivative continuity across the internal points of tabulation, together with

$$C_{m-2}^{(n)} - 2C_{m-1}^{(n)} + C_m^{(n)} = 0 \tag{7}$$

which imposes third derivative continuity across $(m - 1)h$, and

$$C_0^{(n)} + \tfrac{1}{2}C_1^{(n)} = \frac{3}{h}\left[\frac{F_1^{(n)} - F_0^{(n)}}{h} - \frac{dF^{(n)}}{dt}(0)\right] \tag{8}$$

which arises from setting the first derivative of the spline approximation at $t = 0$ equal to the analytically defined first derivative of $F^{(n)}$, which is

$$\frac{dF^{(n)}}{dt}(0) = f(0),\ n = 1,$$
$$= 0, \quad n > 1. \tag{9}$$

The approximation for $F^{(n+1)}$ is generated recursively from that for $F^{(n)}$. From (2) and (5),

$$F^{(n+1)}(jh) = 0, j = 0$$
$$= \int_0^{jh} F^{(n)}(jh - x)f(x)\,dx, j \geqslant 1$$
$$\approx \sum_{k=0}^{j-1}\int_{kh}^{kh+h} y_k^{(n)}(jh - x)f(x)\,dx. \tag{10}$$

$F_j^{(n+1)}$ is the approximate value of the sum in (10) obtained by evaluating each partial range integral by five-point Lobatto quadrature as the weighted sum of the product integral at

$$x = (k + [1 + s]/2)h,\ s = \pm 1, \pm\sqrt{3/7}, 0, \tag{11}$$

the corresponding weights being $h/20$, $49h/180$, $16h/45$.

A convenient practical scheme for this calculation is suggested by considering $F_j^{(n+1)}$ as the inner product of two vectors. If $\{v_i\}$ is the vector of the $(4m + 1)$ weighted values of f at the Lobatto nodes in ascending order, with $v_{4k} = hf(kh)/10$, $k \geqslant 1$, to take account of the double occurrence of the end values in the panel-by-panel quadrature, and $\{u_i^{(n)}\}$ the vector of the values of $y^{(n)}$ at the same points in the same order, then

$$F_j^{(n+1)} = \sum_{i=0}^{4j-1} v_i\,u_{4j-1}^{(n)} \tag{12}$$

The v_i being independent of n need be calculated once only and stored; the $u_i^{(n)}$ have to be calculated for each n.

Calculation of $F_j^{(1)}$ can be combined with the tabulation of the v_i using the relations

$$F_j^{(1)}(0) = 0$$
$$F_j^{(1)} = F_{j-1}^{(1)} + \int_{jh-h}^{jh} f(x)\,dx \tag{13}$$

which follow directly from (2), and evaluating the integrals by summing the v_i.

The calculation of the $u_i^{(n)}$ requires the evaluation of $y_j^{(n)}$ of equation (5) at $\alpha_1 h$, $h/2$ and $\alpha_2 h$, $\alpha_1 = (1 - \sqrt{3/7})/2$, $\alpha_2 = 1 - \alpha_1$ for m values of j. It is economical to calculate these as weighted sums of the adjacent F's and C's. The expressions are:

$$u_{4j+1}^{(n)} = \alpha_2 F_j^{(n)} + \alpha_1 F_{j+1}^{(n)} + w_1 C_j^{(n)} + w_2 C_{j+1}^{(n)}$$
$$u_{4j+2}^{(n)} = (F_j^{(n)} + F_{j+1}^{(n)})/2 - h^2(C_j^{(n)} + C_{j+1}^{(n)})/16 \tag{14}$$
$$u_{4j+3}^{(n)} = \alpha_1 F_j^{(n)} + \alpha_2 F_{j+1}^{(n)} + w_2 C_j^{(n)} + w_1 C_{j+1}^{(n)}$$

where $w_1 = -h^2(3 + \sqrt{3/7})/84$, and $w_2 = -h^2(3 - \sqrt{3/7})/84$.

Approximations to equation (4) are calculated by analytical convolution of the cubic spline representations of $G^{(r)}$ and $H^{(s)}$, say G_j, E_j and H_j, D_j defined at $jh, j = 0, 1, \ldots, m$. If g_k and h_k denote the continuous representations of the form (5) in the range $(kh, kh + h)$, it is straightforward to verify that

$$(G^{(r)} * H^{(s)})(jh) \approx \sum_{k=0}^{j-1}\int_0^h g_{j-k-1}(h - z)h_k(z)\,dz$$

$$= \frac{1}{24}\sum_{k=0}^{j-1}\Big\{(H_{k+1} - H_k)\,[12(G_{r-1} + G_r) - h^2(E_{r-1} + E_r)] \qquad (15)$$

$$+ h^2[h^2(E_r D_{k+1} - E_{r-1} D_k)/30 - (G_r - G_{r-1})(D_{k+1} + D_k)]\Big\}$$

where $r = j - k$

An error analysis of this algorithm is not possible, and it has to be tested empirically. Such a test is described in the reference. A working rule, based on experience on a CDC 6600 with 14 decimal digits is that for $0 \leqslant h \leqslant 0{\cdot}5$, 4-decimal place accuracy (adequate for probability calculations) can be expected and this accuracy does not decrease with increasing n. Subsequent experience suggests that similar accuracy can be expected from single-length working on an IBM 360. Since the accuracy increases as h decreases, an empirical test is always possible.

The subroutines

Experience in using the algorithm to calculate a number of systems characteristics has shown that it is convenient for the user to have it coded as three subroutines. These are:

1. *SPSPL*

This takes as input a set of equally spaced ordinates, F_j, $j = 0, 1, \ldots, m$, the spacing h, the number of points and the derivative at F_0, and returns the corresponding spline coefficients by solving the $(m + 1)$ simultaneous equations given by (6), (7) and (8). It is coded so as to economise on storage. Though mostly used in conjunction with CONVOL, it is sometimes required independently. The number of operations is O(m).

2. *CONVOL*

This produces the approximations to the successive $F^{(n)}$ of equation (2) on successive calls. On the first call, it tabulates the v_i of equation (12) via a user-supplied function subroutine to calculate the probability density function f, at the same time calculating the $F_j^{(1)}$ from equation (13). SPSPL is called to produce the $C_j^{(1)}$. The constants, w_1, w_2 and $-h^2/16$ of equations (14) required in subsequent calls are also calculated.

On the $(n + 1)$st call, $n \neq 0$, it takes as input the $F_j^{(n)}$, $C_j^{(n)}$ and returns the $F_j^{(n+1)}$, $C_j^{(n+1)}$ as output. The $F_j^{(n+1)}$ are produced via the summations (12). Here the summations for all values of j are done simultaneously instead of sequentially to avoid the necessity of storing the $(4m + 1)$ values of $u_i^{(n)}$ at the same time. This saving in storage is paid for by a slight increase in computing time due to the increased use of subscripted variables. The $C_j^{(n+1)}$ are again produced by SPSPL.

On the first call, the number of operations is O(m), on subsequent calls, O(m^2).

3. *CNVLTE*

This is a straightforward coding of the summation (15). It takes as input the cubic spline representation of two functions at $jh, j = 0, 1, \ldots, m$ and outputs their convulate, but not its spline coefficients, at the same points.

The number of operations is O(m^2).

Acknowledgement

This work was supported by National Research Council of Canada Grant No. A8321. I am grateful to the referee for a number of recommendations which have greatly improved the presentation.

```
C
C ------------------------------------------------------------------
C
C     SUBROUTINE SPSPL
C
C     PURPOSE
C
C           TO CALCULATE THE CUBIC SPLINE COEFFICIENTS FOR A SET OF
C           FUNCTION VALUES TABULATED AT A CONSTANT INTERVAL, WHERE THE
C           DERIVATIVE AT THE INITIAL POINT IS KNOWN.
C
C     USAGE
C
C           CALL SPSPL(Y,H,NPT,C,D)
C
C     DESCRIPTION OF THE PARAMETERS
C     Y       VECTOR OF DIMENSION AT LEAST NPT. Y(1) TO Y(NPT) HOLD VALUES
C             OF FUNCTION IN THE CORRECT SEQUENCE ON ENTRY.
C     H       REAL. INTERVAL OF ARGUMENT FOR WHICH FUNCTION IS TABULATED.
C     NPT     INTEGER. NUMBER OF DATA POINTS. MUST BE .GE. 4.
C     C       VECTOR OF DIMENSION AT LEAST NPT. C(1) TO C(NPT) HOLD SPLINE
C             COEFFICIENTS ON EXIT.
C     D       VECTOR OF DIMENSION AT LEAST NPT USED AS WORKSPACE. KNOWN
C             VALUE OF DERIVATIVE AT INITIAL POINT MUST BE SET IN D(1).
C
C     METHOD
C
C           SOLVES SIMULTANEOUS EQUATIONS BY GAUSSIAN ELIMINATION AND
C           BACK SUBSTITUTION.SPLINE APPROXIMATION GETS DERIVATIVE IN
C           D(1). C(NPT-1) IS CALCULATED DIRECTLY FROM 3-POINT FINITE-
C           DIFFERENCE APPROXIMATION TO 2ND DERIVATIVE.
C
C ------------------------------------------------------------------
C
      SUBROUTINE SPSPL(Y,H,NPT,C,D)
      DIMENSION Y(NPT),C(NPT),D(NPT)
      A = 1./H
C
C     RIGHT HAND SIDES OF EQUATIONS INITIALLY PUT IN C,S
C
      C(1) = 3.*A*(A*(Y(2) - Y(1)) - D(1))
      A = 1.5*A*A
      JM1 = 1
      DO 10 J=3,NPT
      JM2 = JM1
      JM1 = J - 1
   10 C(JM1) = A*(Y(JM2) - 2.*Y(JM1) + Y(J))
      C(JM1) = 2.*C(JM1)/3.
      C(JM2) = C(JM2) - .25*C(JM1)
C
C     GAUSSIAN ELIMINATION.
C
      B = .5
      A = C(1)
      DO 20 J=2,JM2
      D(J) = B
      B = 1./(4. - B)
      A = B*(4.*C(J) - A)
   20 C(J) = A
C
C     BACK SUBSTITUTION.
C
      I = JM2
      DO 30 J=2,JM2
      IM1 = I - 1
      A = C(IM1) - A*D(I)
      C(IM1) = A
   30 I = IM1
      C(NPT) = 2.*C(JM1) - C(JM2)
      RETURN
      END
C
C ------------------------------------------------------------------
C
C     SUBROUTINE CONVOL
C
C     PURPOSE
C
C           TO GENERATE ON NTH CALL CUBIC SPLINE APPROXIMATION TO F(N)(T)
C           DEFINED RECURSIVELY AS INTEGRAL FROM X=0 TO X=T OF
C           F(N-1)(T-X)*DF(X), N .GE. 1, AND = 1, N=0, WHERE DF IS A
C           P.D.F., CONTINUOUS AND WITH CONTINUOUS FIRST DERIVATIVE.
C           FUNCTION VALUES AND COEFFS ARE TABULATED AT NPT PTS K*H,
C           K=0,1,...,NPT-1.
C
C     USAGE
C
C           CALL CONVOL(DF,H,NPT,FN,CN,WTDF,WKSP,N)
C
C     DESCRIPTION OF PARAMETERS
C
C     DF      FUNCTION SUBROUTINE OF ONE ARGUMENT TO CALCULATE DF(X),
C             APPEARING IN 'EXTERNAL' STATEMENT IN CALLING PROGRAM.
C     H       REAL. STEP LENGTH, OR INTERVAL OF ARGUMENT FOR TABULATION.
C     NPT     INTEGER. NUMBER OF PTS MUST BE .GT. 4. MAXIMUM T = H*(NPT-1)
C             NPT MAY BE DECREASED BETWEEN CALLS, BUT NOT INCREASED.
C     FN      VECTOR OF DIMENSION AT LEAST NPT. MUST HOLD NPT VALUES FROM
C             T=0 TO T=H*(NPT-1) OF F(N-1)(T) ON ENTRY(EXCEPT WITH N=0).
C             HOLDS VALUES OF F(N)(T) ON EXIT.
C     CN      VECTOR OF DIMENSION AT LEAST NPT. ON ENTRY MUST HOLD CUBIC
C             SPLINE COEFFS FOR FN(N-1)(T) (EXCEPT WITH N=0), AND HOLDS
C             THOSE FOR F(N)(T) ON EXIT.
C     WTDF    VECTOR OF DIMENSION AT LEAST 4*NPT-3 TO HOLD WEIGHTED VALUES
C             OF DF AT LOBATTO NODES. NOT TO BE OVERWRITTEN BETWEEN CALLS.
C     WKSP    VECTOR OF DIMENSION AT LEAST NPT USED FOR WORKSPACE. MAY BE
C             USED BY CALLING PROGRAM BETWEEN CALLS.
C     N       NAME OF INTEGER REGISTER. MUST BE SET TO ZERO BY USER BEFORE
C             FIRST CALL (WHEN VALUES IN FN AND CN ARE IRRELEVANT) AND IS
C             AUTOMATICALLY INCREASED BY 1 AT EACH SUBSEQUENT CALL.
C
C     OTHER SUBROUTINES USED
C
C           CONVOL CALLS SPSPL WHICH MUST BE LOADED WITH IT.
C
C     COMMENTS
C
C           PARAMETERS DF AND H ARE NOT REFERRED TO AFTER FIRST CALL(N=0)
C           .IF DF NEEDS PARAMETERS OTHER THAN ITS ARGUMENT, THESE ARE
C           COMMUNICATED BY "COMMON". PRECISION INCREASES AS H DECREASES.
C           REASONABLE RESULTS CAN BE EXPECTED FOR H IN(.1,.5). SUGGESTED
C           INITIAL VALUE H=.25. CONVOL CAN BE USED TO CALCULATE
C           DIFFERENT INTEGRALS IN PARALLEL FOR THE SAME H IF EACH
C           FUNCTION ASSIGNED TO DF HAS ITS OWN SET OF REGISTERS ASSIGNED
C           TO FN, CN, WTDF AND N.
C
C     METHOD
C
C           F(1)(T) AT THE NPT POINTS IS GOT BY PANEL-BY-PANEL QUADRATURE
C           OF DF USING 5-PT LOBATTO FORMULA. FOR F(N), N .GT. 1,
C           CONTINUOUS REPRESENTATION OF F(N-1) IS USED TO CALCULATE
C           INTEGRALS BY SAME 5-PT LOBATTO FORMULA.
C
C ------------------------------------------------------------------
C
```

```
      SUBROUTINE CONVOL(DF,H,NPT,FN,CN,WTDF,WKSP,N)
      DIMENSION P(4),FN(NPT),CN(NPT),WTDF(NPT),WKSP(NPT)
C
      M = NPT
      IF(N.GT.0) GO TO 30
C
C     ENTRY POINT FOR N=0 .CALCULATE CONSTS FOR SUBSEQUENT CALLS. NUM.
C     COEFFS OF B1 AND B2 ARE -(3-RT)/84 AND -(3+RT)/84, RT=SQRT(3/7).
C     B,S USED AS WEIGHTS FOR CN,S IN INTERPOLATION.
C
      HH = H
      AK = HH*HH
      B1 = -.0279208*AK
      B2 = -.0435078*AK
      BM = -.0625*AK
C
C     LOBATTO NODES IN WKSP(1) TO WKSP(4)
C
      WKSP(2) = .5*HH
C
C     FOLLOWING NUM. COEFF IS (1+SQRT(3/7))/2.
C
      WKSP(3) = .827327*HH
      WKSP(1) = HH - WKSP(3)
      WKSP(4) = HH
C
C     LOBATTO WTS IN P(1) TO P(4). NUM.COEFFS ARE 49/180, 16/45 AND 1/20.
C
      P(1) = .272222*HH
      P(3) = P(1)
      P(2) = .355556*HH
      P(4) = .05*HH
      AL1 = DF(0.)
C
C     TABULATE DF AT LOBATTO NODES MULTIPLIED BY WEIGHTS. F(1)(T) IS GOT
C     AT SAME TIME BY PANEL-BY-PANEL SUMMATION.
C
      K = 1
      AK = P(4)*AL1
      WTDF(1) = AK
      FN(1) = 0.
      CK = 0.
      DO 20 J=2,M
      DO 10 L=1,4
      CKP = CK + WKSP(L)
      D = P(L)*DF(CKP)
      AK = AK + D
      K = K + 1
   10 WTDF(K) = D
      WTDF(K) = 2.*D
      FN(J) = AK
      AK = AK + D
   20 CK = CKP
C
C     DERIV. OF F(1)(T) AT T=0 SET FOR CALCULATION OF SPLINE COEFFS.
C
      WKSP(1) = AL1
      GO TO 80
C
C     ENTRY POINT FOR N .GE. 1. USE SPLINE APPROX. TO CALCULATE VALUES
C     OF F(N) AT NODES, ONE PANEL AT A TIME IN P(1) TO P(4).
C
   30 DO 40 J=1,M
   40 WKSP(J) = 0.
      CKP = CN(1)
      P(4) = 0.
      AL2 = 0.
      AL4 = 0.
      BL2 = CKP*B1
      BL4 = CKP*B2
      DO 70 J=2,M
      CK = CKP
      CKP = CN(J)
      AK = P(4)
      P(4) = FN(J)
      AL1 = AL4
      AL4 = .827327*P(4)
      AL3 = AL2
      AL2 = P(4) - AL4
      BL1 = BL4
      BL4 = CKP*B2
      BL3 = BL2
      BL2 = CKP*B1
      P(1) = AL1 + AL2 + BL1 + BL2
      P(2) = .5*(AK + P(4)) + BM*(CK + CKP)
      P(3) = AL3 + AL4 + BL3 + BL4
C
C     CALCULATE CONTRIBUTION OF(J-1)TH PANEL TO SUMS.
C
      NU = 0
      DO 60 K=J,M
      NL = NU + 1
      NU = NU + 4
      D = 0.
      I = 4
      DO 50 L=NL,NU
      D = WTDF(L)*P(I) + D
   50 I = I - 1
   60 WKSP(K) = WKSP(K) + D
C
C     ELIMINATE SMALL NEGATIVE VALUES.
C
      IF(WKSP(J).LT.0.) WKSP(J) = 0.
   70 FN(J) = WKSP(J)
   80 N = N + 1
C
C     CALCULATE SPLINE COEFFS INCORPORATING INITIAL DERIV. IN WKSP(1).
C
      CALL SPSPL(FN,HH,M,CN,WKSP)
      RETURN
      END
C
C-----------------------------------------------------------------------
C
C     SUBROUTINE CNVLTE
C
C     PURPOSE
C
C           TO CONVOLUTE TWO POSITIVE MONOTONIC INCREASING FUNCTIONS Y(T)
C           AND Z(T) GIVEN BY THEIR TABULATED VALUES AND CUBIC SPLINE
C           COEFFICIENTS AT T=K*H, K=0,1,..,NPT-1. CNVLTE RETURNS
C           INTEGRAL FROM X=0 TO X=K*H OF Y(K*H-X)*DZ/DT(X) FOR SAME K,S.
C
C     USAGE
C
C           CALL CNVLTE(Y,YSPL,Z,ZSPL,H,NPT,E)
C
C     DESCRIPTION OF THE PARAMETERS
C
C     Y, YSPL, Z, ZSPL AND E ARE VECTORS, DIMENSION .GE. NPT.
C     Y      Y(1) TO Y(NPT) HOLD TABULATED Y VALUES ON ENTRY.
C     YSPL   YSPL(1) TO YSPL(NPT) HOLD CORRESPONDING SPLINE COEFFICIENTS.
C     Z      Z(1) TO Z(NPT) HOLD TABULATED Z VALUES ON ENTRY.
C     ZSPL   ZSPL(1) TO ZSPL(NPT) HOLD CORRESPONDING SPLINE COEFFICIENTS.
C     H      THE INTERVAL OF TABULATION.
C     NPT    THE NUMBER OF TABULATED POINTS.
C     E      E(1) TO E(NPT) HOLD CALCULATED VALUES OF CONVOLUTION ON EXIT.
C            E MUST BE DIFFERENT FROM Y, YSPL,  Z, AND ZSPL.
C
C     METHOD
C
C           DIRECT PANEL-BY-PANEL CONVOLUTION OF THE SPLINE FORMS FOR Y
C           AND DZ/DT.
C
C-----------------------------------------------------------------------
C
      SUBROUTINE CNVLTE(Y,YSPL,Z,ZSPL,H,NPT,E)
      DIMENSION Y(NPT),YSPL(NPT),Z(NPT),ZSPL(NPT),E(NPT)
      HSQ = H*H
      HF = HSQ/30.
      E(1) = 0.
      DO 20 K=2,NPT
      U = 0.
      YL = Y(K)
      CL = YSPL(K)
      ZU = Z(1)
      DU = ZSPL(1)
      M = K
      DO 10 J=2,K
      M = M - 1
      YU = YL
      YL = Y(M)
      CU = CL
      CL = YSPL(M)
      ZL = ZU
      ZU = Z(J)
      DL = DU
      DU = ZSPL(J)
      YQ = 12.*(YL + YU) - HSQ*(CL + CU)
   10 U = U + (ZU-ZL)*YQ + HSQ*(HF*(CU*DU-CL*DL) - (YU-YL)*(DU+DL))
C
C     ELIMINATE SMALL NEGATIVE VALUES.
C
      IF(U.LT.0.) U = 0.
   20 E(K) = U/24.
      RETURN
      END
```


Reference

CLEROUX, R. and MCCONALOGUE, D. J. (1976). A Numerical Algorithm for Recursively-Defined Convolution Integrals Involving Distribution Functions, *Management Science*, Vol. 22, No. 10, pp. 1138-1146.

Algorithm 103

COMPUTING THE BESSEL FUNCTIONS $Y_n(x + iy)$ *AND* $K_n(x + iy)$

Vijay K. Garg
Department of Mechanical Engineering,
Indian Institute of Technology, Kanpur

Author's note

Both ordinary and modified Bessel functions of the second kind and of integer order and complex argument arise in many areas of mathematical physics. For example, the solution for the propagation of a periodic disturbance through an elastic tube involves the complex Bessel functions $Y_0(x + iy)$ and $Y_1(x + iy)$ as well as $J_0(x + iy)$ and $J_1(x + iy)$. While computer programs for computation of the Bessel functions $J_n(x + iy)$ and $I_n(x + iy)$ are available (Scarton, 1971) for $n = 0$ (1) 10, that for Bessel function $K_n(x + iy)$ is available (Burrell, 1974) only for $n = 0$ and 1.

This note describes two FORTRAN subroutines for calculating in double precision the ordinary and modified Bessel functions of the second kind and integer order for any point in the complex plane. These programs have been tested for n up to 10.

Because $Y_n(z)$ of large modulus $|z|$, where $z = x + iy$, cannot be computed directly under certain asymptotic conditions (Abramowitz and Stegun, 1965, p. 364, Equation (9.2.6)), it is best to write it in terms of $K_n(z)$, for which there is no such problem.

The ascending series for $K_n(z)$ with z complex as given by Abramowitz and Stegun (1965, p. 375, Equation (9.6.11)) can be written in the recursive form as

$$K_n(z) = (-1)^{n+1}(\gamma + \ln \tfrac{1}{2} z) I_n(z) + \tfrac{1}{2} \sum_{k=0}^{n-1} T_k + \tfrac{1}{2}(-z/2)^n \sum_{k=0}^{\infty} (\phi_k + \phi_{k+n}) C_k , \quad (1a)$$

where $\gamma = 0{\cdot}57721566490153286060651 2,$ (1b)

$$T_k = -\frac{z^2/4}{k(n-k)} T_{k-1}, T_0 = \frac{(n-1)!}{(z/2)^n}, \quad (1c, d)$$

$$\phi_k = \sum_{m=1}^{k} \frac{1}{m}, \phi_0 = 0, \quad (1e, f)$$

$$C_k = \frac{z^2/4}{k(k+n)} C_{k-1}, C_0 = \frac{1}{n!}, \quad (1g, h)$$

Degenerate properties of $K_n(z)$ that are needed for computation are: (i) for $n \neq 0$, $K_n(z) \to \infty$ as $z \to 0$, whereas for $n = 0$, the real part of $K_n(z) \to \infty$ while the imaginary part $\to -\arg(z)$ as $z \to 0$; (ii) for n a negative integer, the relation (Abramowitz and Stegun, 1965, p. 375, Equation (9.6.6)) $K_{-n}(z) = K_n(z)$ holds; (iii) $K_n(z)$ is pure real only for z real and positive while it is complex for all other z.

$K_\nu(z)$ is a regular function of z throughout the z-plane cut along the negative real axis, and for fixed $z(\neq 0)$ it is an entire function of ν. Due to truncation and round-off error the effective radius of convergence for the ascending series is reduced to a value, say R_{max}, for a prescribed accuracy criterion. Thus, an asymptotic series expansion for $K_n(z)$ is required in order to compute the modified Bessel function for a modulus larger than R_{max}.

The asymptotic expansion for $K_\nu(z)$ (Gray and Mathews, 1922, p. 55) can be written in the recursive form as

$$K_n(z) = e^{-z}\left(\frac{\pi}{2z}\right)^{\frac{1}{2}}\left[\sum_{k=0}^{R-1} T_k + O\left(\frac{1}{|2z|^R}\right)\right] (|\arg z| < \pi), \quad (2a)$$

with $T_k = -\dfrac{(k+n-\frac{1}{2})(k-n-\frac{1}{2})}{2kz} T_{k-1}, T_0 = 1,$ (2b)

where ν has been replaced by integer n.

Although Equation (2a) is stated to hold for $-\frac{3\pi}{2} < \arg z < \frac{3\pi}{2}$, (Abramowitz and Stegun, 1965; Watson, 1944) it *cannot* hold for $\arg z = \pm\pi$, because for z real and negative ($\arg z = \pm\pi$), both the ascending series (equation 1(a)) and the analytic continuation (Abramowitz and Stegun, 1965, p. 376, Equation (9.6.31)) viz.

$$K_n(ze^{\pm\pi i}) = (-1)^n K_n(z) \mp \pi i I_n(z), \quad (3)$$

demand that $K_n(z)$ be a complex quantity while equation (2a) will make $K_n(z)$ a pure imaginary quantity. The correct range for the validity of equation (2a) is, thus, $|\arg z| < \pi$. For the given program, equation (2a) was used only for $-\frac{\pi}{2} \leqslant \arg z \leqslant \frac{\pi}{2}$, and for z in quadrants II and III, analytic continuation (equation (3)) was used along with equation (2a). Values of $K_n(z)$ for $\pi/2 < |\arg z| < \pi$ using equation (2a) agree with those calculated by the given program using analytic continuation.

Having computed $K_n(z)$, it is now easy to compute $Y_n(z)$ from the relations (Abramowitz and Stegun, 1965, p. 375, Equation (9.6.5); p. 361, Equation (9.1.36)),

$$Y_n(z) = \begin{cases} e^{-n\pi i/2}\left[i(-1)^n I_n(-iz) - \frac{2}{\pi} K_n(-iz)\right] \left(-\frac{\pi}{2} < \arg z \leqslant \pi\right), \\ (-1)^n [Y_n(-z) - 2i J_n(-z)] \\ \qquad \left(-\pi < \arg z \leqslant -\frac{\pi}{2}\right). \end{cases} \quad (4)$$

Methods for computation of $I_n(z)$ and $J_n(z)$ are given in Scarton (1971).

The above relations for $K_n(z)$ and $Y_n(z)$ have been programmed for an IBM 7044 computer using FORTRAN IV. Since single precision complex arithmetic did not result in enough accuracy due to truncation and round-off error, double precision arithmetic was used. Due to non-availability of the double precision complex arithmetic for the IBM 7044 software, a FORTRAN package was specially prepared for such routine operations as multiplication, division, etc.

Three methods were used to test the accuracy of the program. The first method involved checking the values of $K_n(z)$ and $Y_n(z)$ against known tabular data (Abramowitz and Stegun, 1965, Table 9, p. 417-431; NBS, 1950). For $K_0(z)$ and $K_1(z)$, the values were also compared with those obtained by using the program of Burrell (1974). In the second method, the values for $K_n(z)$ were substituted into Bessel's equation $LK_n(z) = \epsilon$, where $L = z(d/dz)(z(d/dz)) - z^2 - n^2$ and ϵ is an absolute error; ϵ would be zero if $K_n(z)$ were completely accurate. In the third method, the values of $K_n(z)$ obtained from the ascending series, equation (1), were compared to those obtained from the asymptotic series, equation (2), in the annular region where both series are valid. The range of parameters n and $z = Re^{i\theta}$ considered in the test program was $n = 0\,(1)\,10$, $R = 0\,(1)\,30$, and $\theta = -\pi\,(\pi/12)\,\pi$.

Defining ϵ_r to be the modulus of the ratio of the N-th term to the N-th partial sum, and relative error δ_r to be the ratio $|LK_n(z)/K_n(z)|$, it was observed that for $\epsilon_r = 10^{-15}$, δ_r varied with all the parameters n, R, and θ. Comparison of the results for the two series suggested that in order to obtain a minimum of 10 decimal places accuracy for $K_n(z)$, the ascending series should be used for $R \leqslant R_m$ while the asymptotic series should be used for $R > R_m$, where

$$R_m = \begin{cases} 8{\cdot}5 + \dfrac{3|\theta|}{\pi} + 0{\cdot}15n & \left(0 \leqslant |\theta| < \dfrac{\pi}{6}\right), \\ 8{\cdot}0 + \dfrac{6|\theta|}{\pi} + 0{\cdot}15n & \left(\dfrac{\pi}{6} \leqslant |\theta| < \dfrac{\pi}{3}\right), \\ 6{\cdot}0 + \dfrac{12|\theta|}{\pi} + 0{\cdot}15n & \left(\dfrac{\pi}{3} \leqslant |\theta| < \dfrac{\pi}{2}\right), \\ 15 - \dfrac{6|\theta|}{\pi} + 0{\cdot}1n & \left(\dfrac{\pi}{2} \leqslant |\theta| \leqslant \pi\right), \end{cases} \quad (5)$$

with θ as the principal value of the argument of z. Equation (5) is valid only for $0 \leqslant n \leqslant 10$ because values of n greater than 10 were not tried.

The Bessel functions $K_n(z)$ and $Y_n(z)$ are given by the subroutines *KNZ(N, A, B, C, D)* and *YNZ(N, A, B, C, D)* respectively where the arguments of these subroutines are to be interpreted as $C + iD = K_N(A + iB)$ and $C + iD = Y_N(A + iB)$ respectively. Arguments A, B, C, and D are all real double precision numbers while N is an integer.

Note that if the imaginary part of z is 0, the values of $K_n(z)$ and $Y_n(z)$ correspond to $\arg z = \pi$.

```
      SUBROUTINE KNZ(NN,AA,BB,C,D)
C     KNZ COMPUTES THE MODIFIED BESSEL FUNCTION OF THE SECOND KIND AND
C     INTEGER ORDER NN, SO THAT K(AA+I BB)=C+ID .
C     NN MAY BE A NEGATIVE INTEGER
      DOUBLE PRECISION AA,BB,A,B,C,D,R,NU,PI,ZZR,ZZI,S1R,S1I,S2R,S2I,RM,
     1T1R,T1I,T2R,T2I,PK,PKN,ABSDPC,DPFACT,CE,PY
      LOGICAL PLUS
      DATA PY,CE/3.14159265358979320,1.D-15/
      A=AA
      B=BB
      N=IABS(NN)
      NU=N
      R=ABSDPC(A,B)
C     TESTING IF THE ARGUMENT IS ZERO
      IF(N.EQ.0 .AND. R.LE.1.D-26) GO TO 16
      RM=R**N
      IF(RM.LE.1.D-26) GO TO 17
C     FINDING THE BOUNDARY RADIUS RM THAT SEPARATES THE ASCENDING SERIES
C     FROM THE ASYMPTOTIC EXPANSION (SEE EQUATION 5)
      D=DATAN2(B,A)
      PK=DABS(D)
      IF(PK.LT.PY/6.D0) GO TO 1
      IF(PK.LT.PY/3.D0) GO TO 2
      IF(PK.LT.PY/2.D0) GO TO 3
      RM=15.D0-6.D0*PK/PY+NU/10.D0
      GO TO 4
    1 RM=8.5D0+3.D0*PK/PY+0.15D0*NU
      GO TO 4
    2 RM=8.0D0+6.D0*PK/PY+0.15D0*NU
      GO TO 4
    3 RM=6.0D0+12.D0*PK/PY+0.15D0*NU
    4 IF(R.GT.RM) GO TO 12
C     ASCENDING SERIES SOLUTION (EQUATION 1).
      C=DLOG(R/2.D0)+0.57721566490153286D0
      CALL INUZ(NU,A,B,PK,RM)
      CALL MULDPC(C,D,PK,RM,C,D)
      ZZR=(A*A-B*B)/4.D0
      ZZI=A*B/2.D0
```

```
      IF(N.GT.0) GO TO 5
C  FINITE SUMMATION FOR N=0 IN EQN. 1A, BEING TRIVIAL, IS HANDLED SEPARATELY
      S1R=0.D0
      S1I=0.D0
      PKN=0.D0
      R=1.D0
      PI=0.D0
      GO TO 9
   5  IF(N.GT.1) GO TO 6
C  SIMILARLY, FINITE SUMMATION FOR N=1 IN EQUATION 1A IS HANDLED SEPARATELY
      CALL DIVDPC(2.D0,0.D0,A,B,S1R,S1I)
      PKN=1.D0
      R=A/2.D0
      PI=B/2.D0
      GO TO 9
C  FINITE SUMMATION IN EQN. 1A FOR N.GT.1
   6  T1R=DPFACT(N-1,1)
      CALL DPCZN(A/2.D0,B/2.D0,N,R,PI)
      CALL DIVDPC(T1R,0.D0,R,PI,T1R,T1I)
      S1R=T1R
      S1I=T1I
      L=N-1
      DO 7  K=1,L
      PK=K
      PK=PK*(PK-NU)
      T1R=T1R/PK
      T1I=T1I/PK
      CALL MULDPC(T1R,T1I,ZZR,ZZI,T1R,T1I)
      S1R=S1R+T1R
   7  S1I=S1I+T1I
      PKN=1.D0
      DO  8 K=2,N
      PK=K
   8  PKN=PKN+1.D0/PK
C  INFINITE SUMMATION CALCULATION (EQN. 1A) - COMMON FOR ALL N
   9  PK=0.D0
      T2R=1.D0/DPFACT(N,1)
      T2I=0.D0
      S2R=PKN*T2R
      S2I=0.D0
      DO 10 K=1,200
      T1R=K
      PK=PK+1.D0/T1R
      PKN=PKN+1.D0/(T1R+NU)
      RM=T1R*(T1R+NU)
      T2R=T2R/RM
      T2I=T2I/RM
      RM=PK+PKN
      CALL MULDPC(T2R,T2I,ZZR,ZZI,T2R,T2I)
      S2R=S2R+T2R*RM
      S2I=S2I+T2I*RM
      CALL DIVDPC(T2R,T2I,S2R,S2I,T1R,T1I)
      RM=ABSDPC(T1R,T1I)*RM
      IF(RM.LE.CE) GO TO 11
  10  CONTINUE
      WRITE(6,18) RM
  11  PK=-1.D0
      IF(MOD(N,2).EQ.0) PK=1.D0
      CALL MULDPC(R,PI,S2R,S2I,S2R,S2I)
      C=-PK*C+S1R/2.D0+PK*S2R/2.D0
      D=-PK*D+S1I/2.D0+PK*S2I/2.D0
      RETURN
C  SOLUTION FOR LARGE Z BY ASYMPTOTIC EXPANSION (EQUATIONS 2 AND 3)
  12  PI=PY/2.D0
      T2R=1.D0
      T2I=0.D0
      S2R=1.D0
      S2I=0.D0
      PLUS=.TRUE.
      IF(A.GE.0.D0) GO TO 13
C  CASE FOR NEGATIVE REAL PART OF THE ARGUMENT IS HANDLED DIFFERENTLY
C  SEE DISCUSSION FOLLOWING EQNS. 2 AND 3
      A=-A
      B=-B
      PLUS=.FALSE.
C  SUMMING UP THE SERIES IN EQN. 2A
  13  DO 14 J=1,200
      PK=J
      PK=-((PK+NU-0.5D0)/(2.D0*PK))*(PK-NU-0.5D0)
      IF(J.GT.N .AND. DABS(PK).GT.R) GO TO 15
      CALL DIVDPC(T2R,T2I,A,B,T2R,T2I)
      T2R=T2R*PK
      T2I=T2I*PK
      S2R=S2R+T2R
      S2I=S2I+T2I
      CALL DIVDPC(T2R,T2I,S2R,S2I,T1R,T1I)
      RM=ABSDPC(T1R,T1I)
      IF(RM.LE.CE)GO TO 15
  14  CONTINUE
      WRITE(6,19)J,N,A,B,T2R,T2I,S2R,S2I,RM
C  CALCULATING KNZ ACCORDING TO EQN. 2A
  15  CALL DIVDPC(PI,0.D0,A,B,T2R,T2I)
      CALL SQTDPC(T2R,T2I,T2R,T2I)
      CALL MULDPC(T2R,T2I,S2R,S2I,T2R,T2I)
      CALL DPCEXP(-A,-B,S2R,S2I)
      CALL MULDPC(T2R,T2I,S2R,S2I,C,D)
      IF(PLUS) RETURN
      PK=-1.D0
      IF(MOD(N,2).EQ.0) PK=1.D0
      CALL INUZ(NU,A,B,T2R,T2I)
      RM=1.D0
      IF(B.GT.0.D0) RM=-1.D0
      C=C*PK+2.D0*PI*T2I*RM
      D=D*PK-2.D0*PI*T2R*RM
      RETURN
C  SOLUTION FOR ZERO ARGUMENT WHEN N=0
  16  C=-DLOG(R)
      D=-DATAN2(B,A)
      RETURN
C  SOLUTION FOR ZERO ARGUMENT WHEN N.NE.0
  17  PK=2**(N-1)
      PI=DPFACT(N-1,1)
      PKN=NU*DATAN2(B,A)
      C= DCOS(PKN)*PK*PI/RM
      D=-DSIN(PKN)*PK*PI/RM
      RETURN
  18  FORMAT(1HO,20X,62HEVEN FOR 200 TERMS IN THE ASCENDING SERIES FOR K
     1NZ THE RATIO =,D11.4)
  19  FORMAT(1HO,5H*****,2I6,7D14.4,3X,5H*****)
      END


      SUBROUTINE YNZ (N,A,B,C,D)
C  YNZ COMPUTES THE BESSEL FUNCTION OF THE SECOND KIND AND INTEGER ORDER
C  N BY USE OF EQUATION 4, SO THAT Y(A+IB)=C+ID .
      DOUBLE PRECISION A,B,C,D,X,Y,NU,R,PI,ER,EI,JR,JI,IR,II,KR,KI
      LOGICAL PLUS
      DATA PI/3.141592653589793 2D0/
      NU=N
      X=A
      Y=B
      PLUS=.TRUE.
      R=-1.D0
      IF(MOD(N,2).EQ.0) R=1.D0
      ER=-NU*PI/2.D0
      CALL DPCEXP(0.D0,ER,ER,EI)
      IF(X.GT.0.D0 .AND. Y.GE.0.D0) GO TO 1
C  THE CASE (-PI.LT.ARG(Z).LE.(-PI/2)) IS HANDLED DIFFERENTLY
      X=-X
      Y=-Y
      CALL JNUZ(NU,X,Y,JR,JI)
      PLUS=.FALSE.
   1  CALL INUZ(NU,Y,-X,IR,II)
      CALL KNZ(N,Y,-X,KR,KI)
      C=-II*R-2.D0*KR/PI
      D=IR*R-2.D0*KI/PI
      CALL MULDPC(ER,EI,C,D,C,D)
      IF(PLUS) RETURN
      C=R*(C+2.D0*JI)
      D=R*(C-2.D0*JR)
      RETURN
      END


      SUBROUTINE INUZ(NI,XX,YY,E,F)
C  INUZ COMPUTES THE MODIFIED BESSEL FUNCTION OF THE FIRST KIND AND NI-TH
C  ORDER (NI IS RESTRICTED TO INTEGER VALUES BUT MUST BE ENTERED AS A
C  DOUBLE PRECISION NUMBER ALONG WITH XX,YY,E,F).  NI MAY BE NEGATIVE.
C  E AND F ARE THE DOUBLE PRECISION REAL AND IMAGINARY PARTS OF INUZ.
      DOUBLE PRECISION RM,E2,E3,E4,R,ABSDPC,NU,C1R,C1I,C2R,C2I,TEMP,SUMR
     1,SUMI,GAM12,TMN,TMNR,TMNI,DPFACT,TEMP1,N,PI,E5,TEMP3,TEMP4,E,F,X,Y
     2,NI,XX,YY
      LOGICAL CPLUS
      DATA E2,E3,E4,E5/1.D-29,1.D-37,1.D-15,1.D-13/
      X=XX
      Y=YY
      NU=DABS(NI)
C  IN THE FOLLOWING 0.01 IS ADDED TO ENSURE NU1=NU AFTER TRUNCATION
      NU1=NU+0.01D0
      R=ABSDPC(X,Y)
C  I(NI,Z) IS INDETERMINATE FOR Z=0 AND MUST BE HANDLED SEPARATELY.
      IF(R.GT.1.0D-20)GO TO 2
      F=0.0D0
      IF(NU1.EQ.0)GO TO 1
      E=0.0D0
      RETURN
   1  E=1.0D0
      RETURN
C  RM IS THE BOUNDARY RADIUS THAT SEPARATES THE ASCENDING SERIES FROM
C  THE ASYMPTOTIC EXPANSION (SEE EQN. 8, P. 298, REF. 1).
   2  RM=18.0D0+3.0D0*NU/7.0D0
      TMN=1.0D0
      TEMP1=1.0D0
      IF(R.GT.RM)GO TO 6
C  0.LT.MAG(Z).LE.RM SOLUTION (EQ.9.6.10, P.375, REF.3)
      CALL DPCZN(X/2.D0,Y/2.D0,NU1,C1R,C1I)
      C2R=(X*X-Y*Y)/4.D0
      C2I=X*Y/2.D0
      TEMP=DPFACT(NU1,1)
      TMNR=C1R/TEMP
      TMNI=C1I/TEMP
      N1=1
      NPNU=1+NU1
      SUMR=TMNR
      SUMI=TMNI
   3  TEMP=N1*NPNU
      TMNR=TMNR/TEMP
      TMNI=TMNI/TEMP
      CALL MULDPC(TMNR,TMNI,C2R,C2I,TMNR,TMNI)
      SUMR=SUMR+TMNR
      SUMI=SUMI+TMNI
      IF(ABSDPC(SUMR,SUMI).GE.E2)GO TO 4
      TEMP=TMN
      TMN=ABSDPC(TMNR,TMNI)
      IF(TEMP.GE.E3.OR.TMN.GE.E3)GO TO 5
      GO TO 16
   4  TEMP=TEMP1
      CALL DIVDPC(TMNR,TMNI,SUMR,SUMI,C1R,C1I)
      TEMP1=ABSDPC(C1R,C1I)
      IF(TEMP1.LT.E4.OR.TEMP.LT.E4)GO TO 16
```

```
    5  N1=N1+1
       NPNU=NPNU+1
       GO TO 3
C   MAG(Z).GT.RM SOLUTION ACCORDING TO EQNS.4 AND 5, P.296, REF. 1 .
    6  PI=3.1415926535897932D0
       IF(Y.GE.0.0D0)GO TO 7
C   AS THIS ASYMPTOTIC SOLUTION IS NOT VALID FOR ALL VALUES OF Z IN
C   THE THIRD AND FOURTH QUADRANTS, WE MUST RESTRICT ARG(Z) TO THE
C   INTERVAL (0.LE.ARG(Z).LE.PI).  IF ARG(Z) LIES IN QUADRANTS 3 OR 4,
C   WE MUST TRANSFORM Z INTO QUADRANTS 1 OR 2 USING ANALYTIC
C   CONTINUATION (EQ. 9.6.30, P. 376, REF.3).
       TEMP=PI+DATAN2(Y,X)
       CALL DPCEXP(0.0D0,TEMP,X,Y)
       X=X*R
       Y=Y*R
    7  CALL DIVDPC(-0.5D0,0.0D0,X,Y,C1R,C1I)
       TEMP3=NU-0.5D0
       TEMP4=-NU-0.5D0
       CPLUS=.FALSE.
    8  TMNR=1.0D0
       TMNI=0.0D0
       SUMR=TMNR
       SUMI=TMNI
       N=1.0D0
    9  TEMP=(1.D0+TEMP3/N)*(TEMP4+N)
       TMNR=TMNR*TEMP
       TMNI=TMNI*TEMP
       CALL MULDPC(TMNR,TMNI,C1R,C1I,TMNR,TMNI)
       SUMR=SUMR+TMNR
       SUMI=SUMI+TMNI
       IF(ABSDPC(SUMR,SUMI).GE.E2)GO TO 10
       TEMP=TMN
       TMN=ABSDPC(TMNR,TMNI)
       IF(TEMP.GE.E3.OR.TMN.GE.E3)GO TO 11
       GO TO 12
   10  TEMP=TEMP1
       CALL DIVDPC(TMNR,TMNI,SUMR,SUMI,C2R,C2I)
       TEMP1=ABSDPC(C2R,C2I)
       IF(TEMP1.LT.E4.OR.TEMP.LT.E4)GO TO 12
   11  N=N+1.0D0
       GO TO 9
   12  IF(CPLUS)GO TO 13
       CPLUS=.TRUE.
       C1R=-C1R
       C1I=-C1I
       TMN=1.D0
       TEMP1=1.D0
       R=SUMR
       RM=SUMI
       GO TO 8
   13  C2I=PI*(NU+0.5D0)-Y
       CALL DPCEXP(-X,C2I,C2R,C2I)
       CALL MULDPC(C2R,C2I,R,RM,C2R,C2I)
       CALL DPCEXP(X,Y,R,RM)
       CALL MULDPC(R,RM,SUMR,SUMI,R,RM)
       C2R=C2R+R
       C2I=C2I+RM
C   CALCULATING GAMMA(N+1/2) ACCORDING TO EQN. 6.1.12, P. 255, REF. 3 .
       GAM12=1.7724538509055160D0
       IF(NU1.EQ.0)GO TO 15
       DO 14 I=1,NU1
       N=I
   14  GAM12=GAM12*(N-0.5D0)
   15  N1=2*NU1
       TEMP=((DPFACT(N1,NU1)/GAM12)/(2.D0**N1))/1.414213562373095D0
       CALL SQTDPC(X,Y,R,RM)
       CALL DIVDPC(TEMP,0.0D0,R,RM,R,RM)
       CALL MULDPC(R,RM,C2R,C2I,SUMR,SUMI)
       IF(YY.GE.0.0D0) GO TO 16
       X=XX
       Y=YY
       TEMP=-NU*PI
       CALL DPCEXP(0.0D0,TEMP,C1R,C1I)
       CALL MULDPC(C1R,C1I,SUMR,SUMI,SUMR,SUMI)
C   END OF ANALYTIC CONTINUATION
   16  E=SUMR
       F=SUMI
C   AS THE CONVERGENCE TENDS TO BE SLOW FOR COMPUTING THE SMALLER
C   OF THE REAL AND IMAGINARY PARTS OF I(NI,Z) FOR Z LYING ON THE REAL
C   OR IMAGINARY AXIS, WE EVOKE THE FINAL RESULT FOR THESE SPECIAL
C   CASES.  NAMELY, I(NI,Z) IS PURE REAL FOR AN INTEGER NI AND A REAL Z,
C   OR FOR AN EVEN INTEGER NI AND AN IMAGINARY Z.  ALSO, I(NI,Z) IS PURE
C   IMAGINARY FOR AN IMAGINARY Z IF NI IS AN ODD INTEGER.  HENCE
       N1=MOD(NU1,2)
       IF((DABS(Y).LE.E5).OR.(DABS(X).LE.E5.AND.N1.EQ.0)) F=0.D0
       IF(DABS(X).LE.E5.AND.N1.NE.0) E=0.D0
       RETURN
       END


       SUBROUTINE JNUZ(NU,XX,YY,E,F)
C   J(NU,Z) IS COMPUTED BY FIRST COMPUTING I(NU,ZP) WHERE ZP IS A TRANSFORMED
C   Z AND THEN USING EQN. 9.6.3, P.375, REF.3.  THE RESTRICTIONS FOR THE
C   ABOVE ARGUMENTS FOR JNUZ ARE IDENTICAL WITH SUBROUTINE INUZ.
       DOUBLE PRECISION NU,XX,YY,E,F,PI,C1R,C1I,C2R,C2I
       PI=3.1415926535897932D0
       IF(XX.LE.0.0D0.AND.YY.LT.0.0D0)GO TO 1
C   CASE 1 (SEE EQN. 7, P. 297, REF. 1)
       C1I=-PI*0.5D0
       C2I=NU*PI*0.5D0
       GO TO 2
C   CASE 2 (SEE EQN. 7, P. 297, REF. 1)
    1  C1I=PI*1.5D0
       C2I=-NU*PI*1.5D0
    2  CALL DPCEXP(0.0D0,C1I,C1R,C1I)
       CALL DPCEXP(0.0D0,C2I,C2R,C2I)
       CALL MULDPC(XX,YY,C1R,C1I,C1R,C1I)
       CALL INUZ(NU,C1R,C1I,C1R,C1I)
       CALL MULDPC(C2R,C2I,C1R,C1I,E,F)
       RETURN
       END


       SUBROUTINE MULDPC (A,B,C,D,E,F)
C   MULDPC MULTIPLIES (A+IB) WITH (C+ID) TO GET (E+IF) .
       DOUBLE PRECISION A,B,C,D,E,F,G
       G=A*C-B*D
       F=A*D+B*C
       E=G
       RETURN
       END


       SUBROUTINE DIVDPC(AA,BB,CC,DD,E,F)
C   DIVDPC DIVIDES (AA+IBB) BY (CC+IDD) TO GET (E+IF) .
       DOUBLE PRECISION D,E,F,AA,BB,CC,DD,G,DDCC
       IF(DABS(DD).GT.DABS(CC)) GO TO 1
       DDCC=DD/CC
       G=CC+DD*DDCC
       D=(AA+BB*DDCC)/G
       F=(BB-AA*DDCC)/G
       E=D
       RETURN
    1  DDCC=CC/DD
       G=DD+CC*DDCC
       D=(BB+AA*DDCC)/G
       F=(-AA+BB*DDCC)/G
       E=D
       RETURN
       END


       SUBROUTINE SQTDPC (AA,BB,C,D)
C   SQTDPC TAKES SQUARE ROOT OF (AA+IBB) TO GET (C+ID) .
       DOUBLE PRECISION C,D,AA,BB,ABSDPC,R,TH
       R=DSQRT(ABSDPC(AA,BB))
       IF(DABS(AA).LE.1.D-28)GO TO 1
       IF(DABS(BB).LE.1.D-28)GO TO 2
       TH=DATAN2(BB,AA)/2.D0
       C=R*DCOS(TH)
       D=R*DSIN(TH)
       RETURN
    1  C=R/1.414213562373095D0
       D=C
       IF(BB.LT.0.D0) D=-C
       RETURN
    2  IF(AA.LT.0.D0) GO TO 3
       C=R
       D=0.D0
       RETURN
    3  C=0.D0
       D=R
       IF(BB.LT.0.D0) D=-R
       RETURN
       END


       SUBROUTINE DPCZN(A,B,N,C,D)
C   DPCZN COMPUTES Z**N=(A+IB)**N=C+ID IN DOUBLE PRECISION
C   N IS RESTRICTED TO N=0,1,2,3,.....
       DOUBLE PRECISION A,B,C,D
       IF(N.GT.0)GO TO 2
       IF(DABS(A).GT.1.0D-28.OR.DABS(B).GT.1.0D-28)GO TO 1
       C=0.0D0
       D=0.0D0
       RETURN
    1  C=1.0D0
       D=0.0D0
       RETURN
    2  C=A
       D=B
       IF(N.EQ.1) RETURN
       DO 3 I=2,N
    3  CALL MULDPC(A,B,C,D,C,D)
       RETURN
       END


       SUBROUTINE DPCEXP(AA,BB,E,F)
C   DPCEXP COMPUTES EXP(AA+IBB) TO GET (E+IF) .
       DOUBLE PRECISION C,E,F,AA,BB,B
       C=DEXP(AA)
       B=C*DCOS(BB)
       F=C*DSIN(BB)
       E=B
       RETURN
       END


       DOUBLE PRECISION FUNCTION ABSDPC(A,B)
C   ABSDPC COMPUTES THE ABSOLUTE VALUE OF (A+IB) .
       DOUBLE PRECISION A,B,C,D
       C=DABS(A)
       D=DABS(B)
       IF(C.GT.D) GO TO 1
       ABSDPC=D*DSQRT(1.D0+(C/D)**2)
       RETURN
    1  ABSDPC=C*DSQRT(1.D0+(D/C)**2)
       RETURN
       END
```

```
      DOUBLE PRECISION FUNCTION DPFACT(N,M)
C  DPFACT COMPUTES (N FACTORIAL/M FACTORIAL) WHERE N AND M ARE POSITIVE
C  INTEGERS.  ALSO, M.LE.N .
      DOUBLE PRECISION TEMP,II
      TEMP=1.0D0
      IF(N.LE.1.OR.N.EQ.M) GO TO 2
      K=M+1
      DO 1 I=K,N
      II=I
    1 TEMP=TEMP*II
    2 DPFACT=TEMP
      RETURN
      END
```


References

ABRAMOWITZ, M. and STEGUN, I. A. (Eds.) (1965). *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, DC.

BURRELL, K. H. (1974). *CACM*, Vol. 17, p. 524.

GRAY, A. and MATHEWS, G. B. (1922). *A Treatise on Bessel Functions and their Applications to Physics*, Second Edition by Gray, A. and MacRobert, T. M., MacMillan, reprinted by Dover Publications, 1966.

NATIONAL BUREAU OF STANDARDS (1950). *Tables of the Bessel Functions $Y_0(z)$ and $Y_1(z)$ for Complex Arguments*, Columbia University Press, New York.

SCARTON, H. A. (1971). *J. Computational Phys.*, Vol. 8, p. 295.

WATSON, G. N. (1944). *A Treatise on the Theory of Bessel Functions* 2nd ed., Cambridge Univ. Press.

Editorial Notes

Reprinting of Algorithms

Algorithms 1-9 (and subsequent comments on them) were originally published in *The Computer Bulletin* between September 1964 and December 1965. It has been decided to print these in *The Computer Journal* in order to provide a complete record in a more permanent form. This will be done as and when space permits, starting with the algorithms below.

Algorithm 1. MINTREE A. Obruča

There occur, in a wide variety of fields, many problems which can be represented by the vertices and lines of a graph. In the classical travelling salesman problem, for example, the vertices would denote the towns, and associated with the lines would be the time taken to travel from one town to another. In an electrical network, the vertices would be the terminal points of a circuit and the lines would represent the connecting wires.

One feature is common to all such problems: they can be represented by means of a matrix, the (i, j)th element of which denotes some property or cost associated with the line connecting points i and j.

The following procedure computes the connected subgraph covering all points such that the sum of the costs associated with all the lines in the subgraph is a minimum.

```
procedure mintree (cost, z, below, total); value cost, z;
          integer z; real total;
          integer array below [1: z]; array cost [1: z, 1: z];
comment Given a complete graph of z points, numbered from
one to z, whose undirected edges have associated distance values
given in the matrix cost [i, j], this procedure finds the minimal spanning
tree, that is, the tree covering all the z points such that the sum of the
distances associated with all the edges of the tree is a minimum.
The tree is described in the vector below [j] for 1 ⩽ j ⩽ z. As this
implies a sense of direction coming down the tree there is a point,
the root of the tree, which does not have a "below" and so below [root]
is put equal to zero. If the graph is incomplete, the corresponding
empty cost elements should be given a very high positive value, such as
10↑10.
The procedure is based on the article of H. Loberman and A.
Weinberger, JACM, 4 (1957). At the start of each cycle (label A)
the smallest element in the matrix is sought. On finding this, a test
is made to see if both nodes are in one sub-tree (using the procedure
root (a)). If they are, that element is replaced by 10↑10 so that it is
never chosen again. If one node lies in a subtree and the other does not,
that edge is added to the subtree. If the two nodes do not lie in any
subtree, they become a new subtree. If the two nodes lie in separate
subtrees, the procedure change (a, b) amalgamates the two subtrees.
In each case the appropriate cost element is replaced by 10↑10. The end
of the procedure is reached when z-1 edges of the tree have been found.
Only the lower triangle of the matrix cost [i, j] is used and mutilated
by the procedure;
begin integer m, n, a, b, count; real min;
      integer procedure root (a); value a; integer a;
          root:= if below [a] = 0 then a else
                                        root (below) [a]);
          procedure change (a, b); value a, b; integer a, b;
          begin integer m;
                for m:= if a = 0 then b else below [a]
                                        while a ≠ 0 do
                begin below [a]:= b;
                      b:= a; a:= m;
                end;
          end change;
      for m:= 1 step 1 until z do below [m]:= 0;
      count:= 0; total:= 0;
A:    min:= 10↑10;
      for m:= 2 step 1 until z do
      for n:= 1 step 1 until m-1 do
          if cost [m, n] < min then
              begin min:= cost [m, n];
                    a:= m; b:= n;
              end;
          if root (a) ≠ root (b) then
                    begin count:= count + 1;
                          change (a, b);
                          total:= total + min;
                          if count = z-1 then go to FINISH;
                    end;
          cost [a, b]:= 10↑10;
          go to A;
FINISH:
end mintree;
```

Note on Algorithm 1 MINTREE Computer Bulletin, Sept. 1964, 67

Anthony E. N. T. Pitman
Mathematics Department
Laporte Industries Limited.

In Algorithm 1, *MINTREE*, an error occurs in the specification part of the procedure heading.

ALGOL 60 syntax does not allow for the occurrence of a bound pair in an array specification.

To correct, delete:

integer array *below* [1 : *z*]; **array** *cost* [1 : *z*, 1 : *z*]; and substitute:

integer array *below*; **array** *cost*;

I have not attempted to test *MINTREE* on a computer as yet.

(This error was caused by a misunderstanding at the proof reading stage. Our apologies are due to Mr. Obruča, whose script was as in the corrected version above, and contained a steering program and sample results. Editor.)

Certification of Algorithm 1. MINTREE

M. H. Rogers, R. H. Thomason
University of Bristol Computer Unit.

MINTREE compiled and ran successfully on the Elliott 503, with the following alterations:

1. In the specification of the arrays *below* and *cost* the dimensions were deleted

2. Four semicolons immediately preceding **end** statements were deleted:
   i. $b:=a$; $a:=m$
   ii. **end**
   iii. $a:=m$; $b:=n$
   iv. **go to** *FINISH*
3. In accordance with the requirements of Elliott ALGOL, the labels *A* and *FINISH* were declared by a switch at the head of the block.

Remarks on Algorithm 1, MINTREE, Computer Bulletin, Sept. 1964, 67.

The following correspondence explains a rather interesting point arising out of a **for while** statement.

I. D. Hill
Medical Research Council
Statistical Research Unit.

In Algorithm 1, *MINTREE*, there appears to be something wrong with the **procedure** *change*. I am not sure just what the procedure is intended to do; is its sole purpose to change the value of *below* [*a*]? This is all that it does do, since *a* and *b* are called by value. In any case, the line

for *m*:= **if** *a* = 0 **then** *b* **else** *below* [*a*]
while *a* ≠ 0 **do**

does not seem sensible, even though correct ALGOL. There seems to be no point in setting *m* equal to *b* if *a* = 0, since the '**while** *a* ≠ 0' will immediately terminate the procedure, and *m* is local. This line in fact is identical in effect with

for *m*:= *below* [*a*] **while** *a* ≠ 0 **do**

Is this what the author intended?

Alex K. Obruča
The University of Newcastle upon Tyne
Computing Laboratory.

The execution of a **for while** statement in the ALGOL report (4.6.4.3) is:—

L3: *V*: = *E*;
if ¬ *F* **then go to** *Element exhausted*;
Statement S;
go to *L3*;

We see that the control variable is assigned its new value *before* the Boolean test is made. As the array *below* has a lower bound of one, the statement:

for *m*:= *below* [*a*] **while** *a* ≠ 0 **do**

will fail when *a* = 0. In order to avert this catastrophe the statement is written as:

for *m*:= **if** *a* = 0 **then** *b* **else** *below* [*a*]
while *a* ≠ 0 **do**

where *b* could have been any arithmetic expression. However, for clarity, the whole statement could just as well have been written as:

L: **if** *a* ≠ 0 **then begin**
m: = *below* [*a*];
below [*a*]:= *b*;
b := *a*; *a*:= *m*;
go to *L*
end;

The sole purpose of the procedure *change* is to change the value of *below* [*a*] (and any other of its successive *belows* in the subtree of which *a* is part).

Certification of Algorithm 1 *MINTREE*

J. Boothroyd
University of Tasmania Computing Centre

This algorithm has been tested using Elliott 503 ALGOL. The cost matrix used was that often found in pocket diaries, "Shortest Practicable Road Distances in the British Isles". The results agree with those obtained from an alternative procedure available in this Computing Centre.

The following changes are recommended:—

1. Remove the array parameter *cost* from the **value** list.
2. Alter the procedure *change* (*a*, *b*) to read,

```
procedure change (a, b); value a, b; integer a, b;
begin integer m;
    m:= a;
for a:= m while a ≠ 0 do
    begin m:= below [a]; below [a]:= b; b:= a end
end change;
```

This modification avoids the complex structure of the original version

for *m*:= **if** *a* = 0 **then** *b* **else** *below* [*a*] **while** *a* ≠ 0 **do** . . .

and in so doing improves both the clarity and the efficiency of the algorithm. This complex structure, incidentally, protects the bounds of the array *below* and avoids the attempted evaluation of *below* [0] which would result from the otherwise logically equivalent and simpler

for *m*:= *below* [*a*] **while** *a* ≠ 0 **do** . . .

Remark on Algorithm 1 MINTREE

Alex. K. Obruča
University of Newcastle upon Tyne

A slight misunderstanding may arise from Mr. J. Boothroyd's first recommendation in his Certification [*Computer Bulletin 9 (June 1965), 19*].

The array *cost* was included in the **value** list in order to avoid the mutilation of the original cost matrix in the program within which the procedure MINTREE would be used.

But if storage is at a premium, then the parameter *cost* may be removed from the **value** list and the original matrix restored, after use of MINTREE, by copying from the elements of the upper triangle into those of the lower triangle.