Algorithms

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

**Submittal of an algorithm for consideration for publication in Communications of the ACM implies unrestricted use of the algorithm within a computer is permissible.**

# Algorithm 487

# Exact Cumulative Distribution of the Kolmogorov-Smirnov Statistic for Small Samples [S14]

John Pomeranz [Recd. 13 Mar. 1973]
Computer Sciences Department, Mathematical Sciences Building, Purdue University, West Lafayette, IN 47907*

Key Words and Phrases: Kolmogorov-Smirnov test, K-S statistic, goodness-of-fit testing
CR Categories: 8.1, 5.5
Language: Fortran

## Description

The algorithm calculates the exact cumulative distribution of the two-sided Kolmogorov-Smirnov statistic for samples with few observations. The general problem for which the formula is needed is to assess the probability that a particular sample comes from a proposed distribution. The problem arises specifically in data sampling and in discrete system simulation. Typically, some finite number of observations are available, and some underlying distribution is being considered as characterizing the source of the observations.

The statistic used here simply measures the maximum deviation between the proposed distribution and the empirical distribution derived from the sample. Elementary rules for calculating this deviation can be found in, e.g. Knuth [4, p. 41], Brunk [2, p. 267], or Miller and Freund [5, p. 222]. Simply put, let $S_N(x)$ be the fraction of the $N$ observations which are less than $x$. Let $F(x)$ be the proposed cumulative distribution of the source. Let

$$K_N = \sqrt{N} \times \max_x | S_N(x) - F(x) |.$$

Usually $K_N$ is called a two-sided Kolmogorov-Smirnov statistic. Omitting the absolute value signs gives a one-sided statistic. For computational ease we let $D_N = K_N/\sqrt{N}$ be the observed deviation, unweighted by $\sqrt{N}$.

* Present address: A.T. Kearney, Inc., 100 South Wacker Drive, Chicago, IL 60606.

The inputs to the function are the sample size $N$ and a critical value $D$. The function value is the exact probability $\Pr\{D_N < D\} = \Pr\{K_N < D\sqrt{N}\}$.

The formulas used in the function are obtained directly from Durbin [3, formulas (23) and (24)]. To validate the function, another was coded using matrices determined by Pomeranz [7], and the two were identical to eight decimal places. Then the function was used to generate Birnbaum's Table 1 [1, pp. 428–30] for $D = 1/N$, $2/N, \ldots, J/N, J = \min\{N, 15\}, 1 \le N \le 100$. Eight entries differed by $10^{-5}$, apparently from roundoff error [1, p. 440]. The final test was of Miller's Table 1 [6, pp. 113–15] of critical values in the extreme tail for $1 \le N \le 100$. (Miller's approximation is based on the one-sided statistic with doubled tail probabilities, which is accurate in the extreme tail.) Newton's method was used to determine the values of $D$, which yield cumulative probabilities of .8, .9, .95, .98 and .99, for each $N$. Miller's entries agreed within one in the fifth decimal place for probabilities other than .8 and within four in the fifth decimal place for the .8 probability. This supports Miller's claim [6, p. 120] and further allows the use of the column $\alpha = .10$ ($P = .80$) in his Table 1 when an error in $D$ of $4 \times 10^{-5}$ is acceptable. However, the two-sided statistic and the one-sided statistic [4, p. 44] are significantly different outside the tail. For example, with a sample size of 10, $\Pr\{K_{10} < .54\}$ is approximately .12, but at the same critical value for the one-sided statistic, the cumulative probability is .50.

Finally, using a CDC 6500, values were computed up to $N = 140$. The major limitation is the magnitude of the exponent required to represent $N^N$. Rearranging sums produced no changes.

## References
1. Birnbaum, Z.W. Numerical tabulation of the distribution of Kolmogorov's statistic for finite sample size. *J. Amer. Stat. Assoc.* 47, 259 (Sept. 1952), 425–4].
2. Brunk, H.D. *An Introduction to Mathematical Statistics.* Ginn and Company, Lexington, Mass., 1960.
3. Durbin, J. The probability that the sample distribution function lies between two parallel straight lines. *Ann. Math. Statist.* 39, 2 (Apr. 1968), 398–411.
4. Knuth, Donald E. *The Art of Computer Programming Volume 2/Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 1969.
5. Miller, Irwin, and Freund, John E. *Probability and Statistics for Engineers.* Prentice-Hall, Englewood Cliffs, N.J., 1965.
6. Miller, Leslie H. Table of percentage points of Kolmogorov statistics. *J. Amer. Stat. Assoc. 5*, 273 (Mar. 1956), 111–21.
7. Pomeranz, John E. Exact values of the two-sided Kolmogorov-Smirnov cumulative distribution for finite sample size. Tech. Rep. 88, Computer Sciences Department, Purdue U., Feb. 1973.

## Algorithm

```
      REAL FUNCTION PKS2(N, D)
      INTEGER N
C N IS THE SAMPLE SIZE USED.
      REAL D
C D IS THE MAXIMUM MAGNITUDE (OF THE DISCREPANCY
C BETWEEN THE EMPIRICAL AND PROPOSED DISTRIBUTIONS)
C IN EITHER THE POSITIVE OR NEGATIVE DIRECTION.
C PKS2 IS THE EXACT PROBABILITY OF OBTAINING A
C DEVIATION NO LARGER THAN D.
C THESE FORMULAS APPEAR AS (23) AND (24) IN
C J. DURBIN.  THE PROBABILITY THAT THE SAMPLE
C DISTRIBUTION FUNCTION LIES BETWEEN TWO PARALLEL
C STRAIGHT LINES. ANNALS OF MATHEMATICAL STATISTICS
C 39, 2(APRIL 1968),398-411.
      DOUBLE PRECISION Q(141), FACT(141), SUM, CI,
     * FT, FU, FV
      IF (N.EQ.1) GO TO 90
      FN = FLOAT(N)
      FND = FN*D
      NDT = IFIX(2.*FND)
      IF (NDT.LT.1) GO TO 100
      ND = IFIX(FND)
      NDD = MIN0(2*ND,N)
      NDP = ND + 1
      NDDP = NDD + 1
      FACT(1) = 1.
      CI = 1.
      DO 10 I=1,N
        FACT(I+1) = FACT(I)*CI
        CI = CI + 1.
```

```
 10 CONTINUE
    Q(1) = 1.
    IF (NDD.EQ.0) GO TO 50
    CI = 1.
    DO 20 I=1,NDD
       Q(I+1) = CI**I/FACT(I+1)
       CI = CI + 1.
 20 CONTINUE
    IF (NDP.GT.N) GO TO 80
    FV = FLOAT(NDP) - FND
    JMAX = IDINT(FV) + 1
    DO 40 I=NDP,NDD
       SUM = 0.
       FT = FND
       K = I
       FU = FV
       DO 30 J=1,JMAX
          SUM = SUM + FT**(J-2)/FACT(J)*FU**K/
    *         FACT(K+1)
          FT = FT + 1.
          FU = FU - 1.
          K = K - 1
 30    CONTINUE
       Q(I+1) = Q(I+1) - 2.*FND*SUM
       JMAX = JMAX + 1
       FV = FV + 1.
 40 CONTINUE
    IF (NDD.EQ.N) GO TO 80
 50 DO 70 I=NDDP,N
       SUM = 0.
       SIGN = 1.
       FT = 2.*FND
       DO 60 J=1,NDT
          FT = FT - 1.
          K = I - J + 1
          SUM = SUM + SIGN*FT**J/FACT(J+1)*Q(K)
          SIGN = -SIGN
 60    CONTINUE
       Q(I+1) = SUM
 70 CONTINUE
 80 PKS2 = Q(N+1)*FACT(N+1)/FN**N
    RETURN
 90 PKS2 = 2.*D - 1.
    RETURN
100 PKS2 = 0.
    RETURN
    END



    SUBROUTINE PRFAC
    DOUBLE PRECISION PF(4,40)
    DIMENSION DXA(4)
    COMMON DX, DXA, PF, J
    DATA I /1/
    DO 10 J=1,4
       IF (DXA(J).EQ.DX) RETURN
 10 CONTINUE
    J = I
    I = I + 1
    IF (I.EQ.5) I = 1
    DXA(J) = DX
    PF(J,1) = 1.
    DO 20 K=2,38
       PF(J,K) = (PF(J,K-1)*DX)/FLOAT(K-1)
 20 CONTINUE
    RETURN
    END



    FUNCTION CEIL(X)
    IF (X.GE.0.) GO TO 10
    I = -X
    CEIL = -I
    RETURN
 10 I = X + .99999999
    CEIL = I
    RETURN
    END



    FUNCTION PKS(N, EPS)
C CALCULATE THE CUMULATIVE DISTRIBUTION OF THE
C KOLMOGOROV-SMIRNOV STATISTIC USING THE FORMULAS OF
C JOHN POMERANZ. EXACT VALUES OF THE TWO-SIDED
C KOLMOGOROV- SMIRNOV CUMULATIVE DISTRIBUTION FOR
C FINITE SAMPLE SIZE. TECHNICAL REPORT NUMBER 88,
C COMPUTER SCIENCES DEPARTMENT, PURDUE UNIVERSITY,
C FEBRUARY 1973.
    DOUBLE PRECISION PF(4,40), U(40), V(40)
    DOUBLE PRECISION SUM
    DIMENSION DXA(4)
    COMMON DX, DXA, PF, L
    DATA MNP /40/
    FN = N
    RN = 1./FN
    K = EPS*FN + .00000001
    FK = K
    IF (ABS(FK-EPS*FN).GT..00000001) GO TO 10
    K = K - 1
    FK = K
 10 CONTINUE
```

```
    DEL = EPS - FK*RN
    XUP = RN - DEL
    XLO = DEL
    IF (ABS(XUP-XLO).LT..00000001) XUP = XLO
    XPREV = 0.
    DO 20 I=1,MNP
       U(I) = 0.
 20 CONTINUE
    U(K+1) = 1.
    IMIN = -K
 30 X = AMIN1(XUP,XLO)
    IF (X.GT..999999) X = 1.
    DX = X - XPREV
    JMIN = CEIL((X-EPS)*FN-.00000001)
    IF (ABS(FLOAT(JMIN)-(X-EPS)*FN).LT..00000001)
    *  JMIN = JMIN + 1
    JMAX = (X+EPS)*FN + .00000001
    IF (ABS(FLOAT(JMAX)-(X+EPS)*FN).LT..00000001)
    *  JMAX = JMAX - 1
    JMAX = JMAX - JMIN + 1
    CALL PRFAC
    DO 60 J=1,MNP
       SUM = 0.
       IF (J.GT.JMAX) GO TO 50
       I = 1
 40    IP = J - I + 1 + JMIN - IMIN

       SUM = SUM + U(I)*PF(L,IP)
       I = I + 1
       IF ((IMIN+I).LE.(JMIN+J)) GO TO 40
 50    V(J) = SUM
 60 CONTINUE
    DO 70 I=1,MNP
       U(I) = V(I)
 70 CONTINUE
    IMIN = JMIN
    XPREV = X
    IF (X.EQ.XUP) XUP = XUP + RN
    IF (X.EQ.XLO) XLO = XLO + RN
    IF (X.LT.1.) GO TO 30
    DO 80 I=1,N
       U(K+1) = U(K+1)*FLOAT(I)
 80 CONTINUE
    PKS = U(K+1)
    RETURN
    END
```

---

# Algorithm 488

# A Gaussian Pseudo-Random Number Generator [G5]

Richard P. Brent [Recd. 9 Nov. 1973, and 19 Dec.1973]
Computer Centre, Australian National University,
Canberra, Australia

**Description**

*Introduction.* Successive calls to the Fortran function *GRAND* return independent, normally distributed pseudo-random numbers with zero mean and unit standard deviation. It is assumed that a Fortran function *RAND* is available to generate pseudo-random numbers which are independent and uniformly distributed on [0, 1). Thus, *GRAND* may be regarded as a function which converts uniformly distributed numbers to normally distributed numbers.

*Outline of the method. GRAND* is based on the following algorithm (Algorithm A) for sampling from a distribution with density function $f(x) = K \exp(-G(x))$ on $[a, b)$, where

$$0 \leq G(x) \leq 1 \tag{1}$$

on $[a, b)$, and the function $G(x)$ is easy to compute:

Step 1. If the first call, then take a sample $u$ from the uniform distribution on [0, 1); otherwise $u$ has been saved from a previous call.
Step 2. Set $x \leftarrow a + (b - a)u$ and $u_0 \leftarrow G(x)$.
Step 3. Take independent samples $u_1, u_2, \ldots$ from the uniform distribution on [0, 1) until, for some $k \geq 1, u_{k-1} \leq u_k$.

**704**

Communications
of
the ACM

December 1974
Volume 17
Number 12

Step 4. Set $u \leftarrow (u_k - u_{k-1})/(1 - u_{k-1})$.

Step 5. If $k$ is even go to Step 2, otherwise return $x$.

The reason why Algorithm A is correct is explained in Ahrens and Dieter [2], Forsythe [4], and Von Neumann [6]. The only point which needs explanation here is that, at Step 4, we can form a new uniform variate $u$ from $u_{k-1}$ and $u_k$, thus avoiding an extra call to the uniform random number generator. This is permissible since at Step 4 it is clear (from Step 3) that $(u_k - u_{k-1})/(1 - u_{k-1})$ is distributed uniformly and independent of $x$ and $k$. (The fact that it is dependent on $u_k$ is irrelevant.)

Let $a_i$ be defined by $(2/\pi)^{\frac{1}{2}} \int_{a_i}^{\infty} \exp(-\frac{1}{2}t^2)dt = 2^{-i}$ for $i = 0, 1, \ldots$. To sample from the positive normal distribution (Algorithm B), we may choose $i \geq 1$ with probability $2^{-i}$ (easily done by inspecting the leading bits in a uniformly distributed number) and then use Algorithm A to generate a sample from $[a_{i-1}, a_i)$, with $G(x) = \frac{1}{2}(x^2 - a_{i-1}^2)$. It is easy to verify that condition (1) is satisfied, in fact

$$\frac{1}{2}(a_i^2 - a_{i-1}^2) < \log(2). \tag{2}$$

Finally, to sample from the normal distribution (Algorithm C), we generate a sample from the positive normal distribution and then attach a random sign.

*Comments on the method.* The algorithm is exact, apart from the inevitable effect of computing with floating-point numbers with a finite word-length. Thus, the method is preferable to methods which depend on the central limit theorem or use approximations to the inverse distribution function.

Let $N$ be the expected number of calls to a uniform random number generator when Algorithm A is executed. If the expected value of $k$ at Step 3 is $E$, and the probability that $k$ is even is $P$, then $N = E + NP$, so $N = E/(1 - P)$. From Forsythe [4, eq. (11)], $E = (b - a)^{-1} \int_a^b \exp(G(x))dx$ and

$$1 - P = \frac{1}{b-a} \int_a^b \exp(-G(x)) \, dx, \quad \text{so}$$

$$N = \int_a^b \exp(G(x)) \, dx \bigg/ \int_a^b \exp(-G(x)) \, dx. \tag{3}$$

From (3) and the choice of $a_i$, the expected number of calls to a uniform random number generator when Algorithm C is executed is

$$\sum_{i=1}^{\infty} 2^{-i} \int_{a_{i-1}}^{a_i} \exp(\frac{1}{2}(x^2 - a_{i-1}^2)) \, dx \bigg/ \int_{a_{i-1}}^{a_i} \exp(-\frac{1}{2}(x^2 - a_{i-1}^2)) \, dx$$
$$\simeq 1.37446. \tag{4}$$

This is lower than 4.03585 for the algorithm given in Forsythe [4], or 2.53947 for the improved version (*FT*) given in Ahrens and Dieter [2]. It is even slightly lower than 1.38009 for the algorithm $FL_4$ of [2], and $FL_4$ requires a larger table than Algorithm C. Thus, Algorithm C should be quite fast, and comparable to the best algorithms described by Ahrens and Dieter [1]. The number (4) could be reduced by increasing the table size (as in the algorithms $FL_4$, $FL_5$, and $FL_6$ of [2]), but this hardly seems worthwhile. Exact timing comparisons depend on the machine and uniform random number generator used. (If a very fast uniform generator is used, then Step 4 of Algorithm A may take longer than generating a new uniform deviate.)

The loss of accuracy caused by Step 4 of Algorithm A is not serious. We may say that $\log_2 (1 - u_{k-1})^{-1}$ "bits of accuracy" are lost, and in our application we have, from (2) and Step 3 of Algorithm A, $\log(2) > u_0 > \cdots > u_{k-1}$, so the number of bits lost is less than $\log_2 (1 - \log(2))^{-1} < 2$.

*Test results.* If $x$ is normally distributed then $u = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{x} \exp(-\frac{1}{2}t^2) \, dt$ is uniformly distributed on $(0, 1)$. Hence, standard tests for uniformity may be applied to the transformed variate $u$. Several statistical tests were performed, using a Univac 1108 with both single-precision (27-bit fraction) and double-precision (60-bit fraction). For example, we tested two-dimensional uniformity by taking $10^6$ pairs $(u, u')$, plotting them in the unit square, and performing the Chi-squared test on the observed num-

bers falling within each of 100 by 100 smaller squares. This test should show up any lack of independence in pairs of successive uniform deviates. We tested one-dimensional uniformity similarly, taking $10^6$ trials and subdividing $(0, 1)$ into 1,000 smaller intervals. The values of $\chi^2$ obtained were not significant at the 5 percent level. It is worth noting that the method of summing 12 numbers distributed uniformly on $(-1/2, 1/2)$ failed the latter test, giving $\chi^2_{999} = 1351$. (The probability of such a value being exceeded by chance is less than $10^{-11}$.)

Naturally, test results depend on the particular uniform generator *RAND* which is used. *GRAND* will not produce independent normally distributed deviates unless *RAND* supplies it with independent uniformly distributed deviates! For our tests we used an additive uniform generator of the form $u_n = u_{n-1} + u_{n-127} \pmod{2^w}$ with $w = 27$ or 60 (see Brent [3] and Knuth [5]), but a good linear congruential generator should also be adequate for most applications.

*Comparison with Algorithm 334.* The fastest exact method previously published in Communications is Algorithm 334 [7]. We timed function *GRAND*, subroutine *NORM* (a Fortran translation of Algorithm 334), and function *RAND* (the uniform random number generator called by *GRAND* and *NORM*). The mean execution times obtained from 500,000 trials on a Univac 1108 were 172, 376 and 59 $\mu$sec respectively. Since *NORM* returns two normally distributed numbers, *GRAND* was effectively 9 percent faster than *NORM*. Based on comparisons in [2], we estimate that the saving would be greater if both routines were coded in assembly language, for much of the execution time of *NORM* is taken up in evaluating a square-root and logarithm which are already coded in assembly language.

*GRAND* requires about 1.38 uniform deviates per normal deviate, and *NORM* requires $4/\pi + 1/2 \simeq 1.77$. Thus, we may estimate that if a uniform generator taking $U$ $\mu$sec per call were used, the time per normal deviate would be $(91 + 1.38U)$ $\mu$sec for *GRAND* and $(83 + 1.77U)$ $\mu$sec for *NORM*. Hence, *GRAND* should be faster for $U \geq 20$.

**References**
1. Ahrens, J.H., and Dieter, U. Computer methods for sampling from the exponential and normal distributions. *Comm. ACM 15*, 10 (Oct. 1972), 873–882.
2. Ahrens, J.H., and Dieter, U. Pseudo-random Numbers (preliminary version). Preprint of book to be published by Springer, Part 2, Chs. 6–8.
3. Brent, R.P. *Algorithms for Minimization Without Derivatives.* Prentice-Hall, Englewood Cliffs, N.J., 1973, pp. 163–164.
4. Forsythe, G.E. Von Neumann's comparison method for random sampling from the normal and other distributions. *Math. Comp. 26*, 120 (Oct. 1972), 817–826.
5. Knuth, D.E. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, Reading, Mass., 1969, pp. 26, 34, 464.
6. Von Neumann, J. Various techniques used in connection with random digits. In *Collected Works, Vol. 5*, Pergamon Press, New York, 1963, pp. 768–770.
7. Bell, J.R. Algorithm 334, Normal random deviates. *Comm. ACM 11*, 7 (July 1968), 498.

**Algorithm**

```
      FUNCTION GRAND(N)
C EXCEPT ON THE FIRST CALL GRAND RETURNS A
C PSEUDO-RANDOM NUMBER HAVING A GAUSSIAN (I.E.
C NORMAL) DISTRIBUTION WITH ZERO MEAN AND UNIT
C STANDARD DEVIATION. THUS, THE DENSITY IS  F(X) =
C EXP(-0.5*X**2)/SQRT(2.0*PI). THE FIRST CALL
C INITIALIZES GRAND AND RETURNS ZERO.
C THE PARAMETER N IS DUMMY.
C GRAND CALLS A FUNCTION RAND, AND IT IS ASSUMED THAT
C SUCCESSIVE CALLS TO RAND(0) GIVE INDEPENDENT
C PSEUDO- RANDOM NUMBERS DISTRIBUTED UNIFORMLY ON (0,
C 1), POSSIBLY INCLUDING 0 (BUT NOT 1).
C THE METHOD USED WAS SUGGESTED BY VON NEUMANN, AND
C IMPROVED BY FORSYTHE, AHRENS, DIETER AND BRENT.
C ON THE AVERAGE THERE ARE 1.37746 CALLS OF RAND FOR
C EACH CALL OF GRAND.
C WARNING - DIMENSION AND DATA STATEMENTS BELOW ARE
C            MACHINE-DEPENDENT.
C DIMENSION OF D MUST BE AT LEAST THE NUMBER OF BITS
C IN THE FRACTION OF A FLOATING-POINT NUMBER.
```

**705**

Communications
of
the ACM

December 1974
Volume 17
Number 12

```
C THUS, ON MOST MACHINES THE DATA STATEMENT BELOW
C CAN BE TRUNCATED.
C IF THE INTEGRAL OF SQRT(2.0/PI)*EXP(-0.5*X**2) FROM
C A(I) TO INFINITY IS 2**(-I), THEN D(I) = A(I) -
C A(I-1).
      DIMENSION D(60)
      DATA D(1), D(2), D(3), D(4), D(5), D(6), D(7),
     * D(8), D(9), D(10), D(11), D(12), D(13),
     * D(14), D(15), D(16), D(17), D(18), D(19),
     * D(20), D(21), D(22), D(23), D(24), D(25),
     * D(26), D(27), D(28), D(29), D(30), D(31),
     * D(32) /0.67448975,0.47585963,0.38377116,
     * 0.32861123,0.29114282,0.26368432,
     * 0.24250845,0.22556744,0.21163416,
     * 0.19992426,0.18991075,0.18122518,
     * 0.17360140,0.16684190,0.16079672,
     * 0.15534971,0.15040938,0.14590257,
     * 0.14177003,0.13796317,0.13444176,
     * 0.13117215,0.12812596,0.12527909,
     * 0.12261088,0.12010356,0.11774170,
     * 0.11551189,0.11340234,0.11140272,
     * 0.10950385,0.10769761/
      DATA D(33), D(34), D(35), D(36), D(37), D(38),
     * D(39), D(40), D(41), D(42), D(43), D(44),
     * D(45), D(46), D(47), D(48), D(49), D(50),
     * D(51), D(52), D(53), D(54), D(55), D(56),
     * D(57), D(58), D(59), D(60)
     * /0.10597677,0.10433484,0.10276601,
     * 0.10126505,0.09982723,0.09844828,
     * 0.09712430,0.09585177,0.09462746,
     * 0.09344840,0.09231190,0.09121548,
     * 0.09015683,0.08913386,0.08814461,
     * 0.08718729,0.08626021,0.08536183,
     * 0.08449070,0.08364548,0.08282824,
     * 0.08202784,0.08125316,0.08049984,
     * 0.07976693,0.07905352,0.07835878,
     * 0.07768189/
C END OF MACHINE-DEPENDENT STATEMENTS
C U MUST BE PRESERVED BETWEEN CALLS.
      DATA U /0.0/
C INITIALIZE DISPLACEMENT A AND COUNTER I.
      A = 0.0
      I = 0
C INCREMENT COUNTER AND DISPLACEMENT IF LEADING BIT
C OF U IS ONE.
   10 U = U + U
      IF (U.LT.1.0) GO TO 20
      U = U - 1.0
      I = I + 1
      A = A - D(I)
      GO TO 10
C FORM W UNIFORM ON 0 .LE. W .LT. D(I+1) FROM U.
   20 W = D(I+1)*U
C FORM V = 0.5*((W-A)**2 - A**2). NOTE THAT 0 .LE. V
C .LT. LOG(2).
      V = W*(0.5*W-A)
C GENERATE NEW UNIFORM U.
   30 U = RAND(0)
C ACCEPT W AS A RANDOM SAMPLE IF V .LE. U.
      IF (V.LE.U) GO TO 40
C GENERATE RANDOM V.
      V = RAND(0)
C LOOP IF U .GT. V.
      IF (U.GT.V) GO TO 30
C REJECT W AND FORM A NEW UNIFORM U FROM V AND U.
      U = (V-U)/(1.0-U)
      GO TO 20
C FORM NEW U (TO BE USED ON NEXT CALL) FROM U AND V.
   40 U = (U-V)/(1.0-V)
C USE FIRST BIT OF U FOR SIGN, RETURN NORMAL VARIATE.
      U = U + U
      IF (U.LT.1.0) GO TO 50
      U = U - 1.0
      GRAND = W - A
      RETURN
   50 GRAND = A - W
      RETURN
      END
```

---

# Remark on Algorithm 420 [J6]

Hidden-Line Plotting Program [Hugh Williamson, Comm. ACM 15 (Feb. 1972) 100–103] and Remark on Algorithm 420 [T.M.R. Ellis, Comm. ACM 17 (June 1974), 324–325]

T.M.R. Ellis [Recd. 8 July 1974] Computing Services, University of Sheffield, England

There was an unfortunate printing error in my Remark on Algorithm 420 which made nonsense of the whole thing. The statement which should be inserted to correct the error discussed should, of course, be:

IF(F1.EQ.F2) GO TO 1005

and not: IF(F1.EQ.FZ) GO TO 1005 as printed.

# Remark on Algorithm 426
Merge Sort Algorithm [M1]
[C. Bron, Comm. ACM 15 (May 1972), 357–358]

C. Bron [Recd. 5 Nov. 1973]
Technological University of Twente, P.O. Box 217, Enschede, The Netherlands

A remark in [3 p. 158] suggested to the author that Algorithm 426 needs only very minor modifications in order to handle the sorting of records that are chained to begin with. The algorithm then rearranges the chain and needs no additional array to store chaining information. Furthermore, the assumption that we should be able to associate each of the integers from 1 to $n$ with each of the $n$ records to be sorted is no longer necessary [2].

References

1. Bron, C. Algorithm 426, Merge Sort Algorithm. Comm. ACM 15 (May 1972), 358.
2. Bron, C. An "In Situ" Merge Sort Algorithm. Tech. Note CB 64, Technological University of Twente, Enschede. The Netherlands.
3. Martin, W.A. Sorting. Comp. Surv. 3 (1971), 147–174.

---

# Remark on Algorithm 456 [H]
Routing Problem
[Zdeněk Fencl, Comm. ACM 16 (Sept. 1973), 572]

Gerhard Tesch [Recd. 15 Oct. 1973] VFW Vereinigte Flugtechnische Werke GMBH, 28 Bremen 1, Hunefeldstrasse 1–5, Germany and Zdeněk Fencl, M.I.T., Department of Urban Studies, R. 9–643, Cambridge, Mass.

Some confusion arose from the description of the algorithm capability. It should have been stated that the generated tour must pass through each of the n nodes once and only once, although this is the base for the definition of the traveling salesman problem. This algorithm solves an extended traveling salesman problem in which the end node does not have to be the start node. Such connections may be sought in the design automation of serial printed circuits as well as in transportation problems. The traveling salesman problem is discussed in [3, p. 232] and methods of solution are surveyed in [1].

The users who seek the shortest paths in electric networks (the shortest connection between the two specified nodes in a net without regard to the number of nodes to be connected) are referred to Ford's shortest path algorithm [2, p. 69] and Dantzig's shortest path algorithm [3, p. 175]. There is a set of three efficient Algol algorithms by J. Boothroyd [4] handling the shortest path problem as defined in [2, p. 69] and [3, p. 175]. These Algol algorithms can be modified so that even the number of nodes may be minimized or a restriction of some nodes may be imposed, etc.

Another type of shortest path algorithm is Lee's algorithm [5 and 6]. This algorithm is applicable for the orthogonal routing of printed circuit boards.

References
1. Bellmore, M., and Nemhauser, G.L. The traveling salesman problem: A survey. Oper. Res. 16 (1968), 538–558.
2. Berge, C. The Theory of Graphs and Its Applications. Wiley, New York, 1962.
3. Berge, C., and Ghouila-Houri, A. Programming, Games and Transportation Networks. Wiley, New York, 1965.
4. Boothroyd, J. Algorithms 22, 23, 24. Shortest path. Comp. J. 10 (1967), 306–308.
5. Lee, C.J. An algorithm for path connections and its applications. IEEE Trans. Elect. Comput. EC-10 (Sept. 1961), 346–365.
6. Akers, S.B. A modification of Lee's path connection algorithm. IEEE Trans. Elect. Comput. (Feb. 1967), 97–98.