We now prove separately the inequality for every term of the summation. Let us assume the thesis is wrong

$$\left[\left(\frac{x_{1,k+1}+x_{2,k+1}}{2}-\frac{x_{1,k}+x_{2,k}}{2}\right)^2+\left(\frac{y_{1,k+1}+y_{2,k+1}}{2}-\frac{y_{1,k}+y_{2,k}}{2}\right)^2\right]^{\frac{1}{2}}$$
$$>([(x_{1,k+1}-x_{1,k})^2+(y_{1,k+1}-y_{1,k})^2]^{\frac{1}{2}}$$
$$+[(x_{2,k+1}-x_{2,k})^2+(y_{2,k+1}-y_{2,k})^2]^{\frac{1}{2}}).$$

Squaring twice and simplifying, we obtain

$$[(x_{1,k+1}-x_{1,k})(y_{2,k+1}-y_{2,k})-(x_{2,k+1}-x_{2,k})(y_{1,k+1}-y_{1,k})]^2<0$$

which is contradictory. We can have equality to zero iff

$$\frac{y_{2,k+1}-y_{2,k}}{x_{2,k+1}-x_{2,k}}=\frac{y_{1,k+1}-y_{1,k}}{x_{1,k+1}-x_{1,k}},\qquad k=1,\cdots,n$$

i.e. if all the corresponding sides of the two polygons are parallel.

(c, d) We must prove that the minimal polygon has at least one vertex on the boundary of the corresponding domain. In fact, consider a minimal polygon: there will be at least a couple of adjacent noncollinear sides. The corresponding vertex must lie on the boundary, otherwise a shorter perimeter polygon could be found (see Figure 7 (a), (b)). Furthermore, from simple geometrical considerations it can be deduced that the normal to the boundary must bisect the angle between the two adjacent sides (Figure 7 (b)). Conversely, if a constraint is not active, i.e. a vertex is not on the boundary, the adjacent sides in the minimal polygon must be collinear.

(e) From the convexity properties of the domain D and of the function f it can be deduced that:

(i) all local minima are global;

(ii) any convex linear combination of minima is a minimum as well;

(iii) Jensen's relation (3) applied to two minima obviously holds with equality.

We have proved in (b) that in our case Jensen's relation holds with equality if and only if the corresponding sides of the two polygons are parallel. Now assume to have found a (global) minimum: any vertex between two noncollinear sides will satisfy the bisection property proved in (d). If no straight line segments are present on the boundary of domain C (e.g. if domain C is a circle), it is not possible to translate two adjacent noncollinear sides still satisfying the bisection property. Therefore, all minimal polygons have the vertices between two noncollinear sides in the same position. Thus they differ only in the position of vertices corresponding to nonactive constraints: the minimal reduced polygon is unique.

RECEIVED FEBRUARY, 1969

REFERENCES

1. LEDLEY, R. S. High-speed automatic analysis of biomedical pictures. *Science 143* (1964), 219–233.
2. FREEMAN, H. On the encoding of arbitrary geometric configurations. *IRE Trans. EC-10*, 2 (1961), 260–268.
3. FREEMAN, H., AND GLASS, J. M. On the quantization of line-drawing data. *IEEE Trans SSC-5*, 1 (1969), 70–79.
4. ROSENFELD, A., AND PFALZ, J. L. Sequential operations in digital picture processing. *J. ACM 13*, 4 (1966), 471–494.
5. MONTANARI, U. Sulla descrizione strutturale di immagini. Proc. Congresso AICA 1968, Napoli, Italy, Sept. 26–29, 1968.
6. MONTANARI, U. Continuous skeletons from digitized images. *J. ACM. 16*, 4 (Oct. 1969) 534–549.
7. ROSENFELD, A. Figure extraction. In *Automatic Interpretation and Classification of Images*, A. Grasselli (Ed.), Academic Press, New York (in press).
8. SAATY, T. L., AND BRAM, J. *Nonlinear Mathematics*. McGraw-Hill, New York, 1964.

Algorithms

L. D. FOSDICK, Editor

ALGORITHM 368
NUMERICAL INVERSION OF LAPLACE TRANSFORMS [D5]

HARALD STEHFEST* (Recd. 29 July 1968, 14 Jan. 1969 and 24 July 1969)

Institut f. angew. Physik, J. W. Goethe Universität, 6000 Frankfurt am Main, W. Germany

* The work forms part of a research program supported by the Bundesministerium für wissenschaftliche Forschung and the Fritz ter Meer-Stiftung.

KEY WORDS AND PHRASES: Laplace transform inversion, integral transformations, integral equations

CR CATEGORIES: 5.15, 5.18

procedure *Linv*(*P*, *N*, *T*, *Fa*, *V*, *M*);
value *N*, *T*;
integer *M*, *N*; **real** *T*, *Fa*; **array** *V*; **real procedure** *P*;
comment If a Laplace transform $P(s)$ is given in the form of a real procedure, *Linv* produces an approximate value *Fa* of the inverse $F(t)$ at T. *Fa* is evaluated according to

$$Fa=\frac{\ln 2}{T}\sum_{i=1}^{N}V_i\,P\left(\frac{\ln 2}{T}\,i\right).$$

N must be even. Since the V_i depend on N only, in case of repeated procedure calls with the same N the array V is to be evaluated only once. That is why the formal parameter M has been introduced: that part of the algorithm which computes the V_i is run through only if $M\neq N$, and after every call of *Linv* M equals N. At the first call M may be any integer different from N.

The calculation method originates from Gaver [2], who considered the expectation of $F(t)$ with respect to the probability density

$$f_n(a,t)=a\frac{(2n)!}{n!(n-1)!}(1-e^{-at})^n e^{-nat},\qquad a>0:$$

$$\bar{F}_n=\int_0^\infty F(t)f_n(a,t)\,dt \tag{1}$$
$$=a\frac{(2n)!}{n!(n-1)!}\sum_{i=0}^{n}\binom{n}{i}(-1)^i P((n+i)a).$$

$f_n(a,t)$ has the following properties:

1. $\int_0^\infty f_n(a,t)\,dt=1$,
2. modal value of $f_n(a,t)=\ln 2/a$
3. $\mathrm{var}(t)=1/a^2\sum_{i=0}^{n}1/(n+i)^2$.

They imply that $\bar{F}_n$ converges to $F(\ln 2/a)$ for $n\to\infty$. $\bar{F}_n$ has the asymptotic expansion [2]

$$\bar{F}_n\sim F\left(\frac{\ln 2}{a}\right)+\frac{\alpha_1}{n}+\frac{\alpha_2}{n^2}+\frac{\alpha_3}{n^3}+\cdots.$$

For a given number N of P-values a much better approximation to $F(\ln 2/a)$ than $\bar{F}_{N-1}$ is attainable, and that by linear combination of $\bar{F}_1$, $\bar{F}_2$, $\cdots$, $\bar{F}_{N/2}$: requiring

$$\sum_{i=1}^{K}x_i(K)\frac{1}{(N/2+1-i)^k}=\delta_{k0},$$
$$k=0,1,\cdots,K-1,\quad K\leq N/2,$$

we find

$$x_i(K) = \frac{(-1)^{i-1}}{K!}\binom{K}{i} i\,(N/2+1-i)^{K-1}$$

and thus

$$\sum_{i=1}^{K} x_i(K)\bar{F}_{N/2+1-i} = F\left(\frac{\ln 2}{a}\right) + (-1)^{k+1}\,\alpha\,\frac{(N/2-K)!}{(N/2)!} + o\left(\frac{(N/2-K)!}{(N/2)!}\right).$$

Setting $K = N/2$, $a = \ln(2)/T$, and using (1) we get the expression the procedure evaluates:

$$Fa = \sum_{i=1}^{N/2} x_i(N/2)\bar{F}_{N/2+1-i} = \frac{\ln 2}{T}\sum_{i=1}^{N} V_i\,P\left(\frac{\ln 2}{T}\,i\right)$$

with

$$V_i = (-1)^{N/2+i}\sum_{k=\left[\frac{i+1}{2}\right]}^{Min(i,N/2)} \frac{k^{N/2+1}\,(2k)!}{(N/2-k)!k!(k-1)!(i-k)!(2k-i)!}.$$

(The method of "extrapolation to the limit," which Gaver [2] used, leads to less accurate results for the same N, because not so many powers of n cancel out. Moreover, with this method N must be a power of 2, so that in general one cannot make the best use of the available computer precision.)

Theoretically Fa becomes the more accurate the greater N. Practically, however, rounding errors worsen the results if N becomes too large, because V_i with greater and greater absolute values occurs. (This reflects the unboundedness of the inverse Laplace operator.) For given $P(s)$ and T the N at which the accuracy is maximal increases with the number of significant figures used. For fixed computer precision the optimum value of N is the smaller, i.e. the maximum accuracy is the greater, the faster $\bar{F}_n$ (see eq. (1)) converges to $F(T)$. In the following the term "smooth" is used to express that the rate of convergence is sufficiently great. An oscillating $F(t)$ certainly is not smooth enough unless the wavelength of the oscillations is large compared with the half-width of the peak which $f_{N/2}(\ln 2/T,\ t)$ has at T. No accurate results are to be expected, too, if $F(t)$ has discontinuities near T. If $F(t)$ behaves equally in the neighborhood of two different T-values the result at the smaller T-value will be the better one, because the peak of $f_n(\ln 2/T,\ t)$ broadens as T increases.

The only way to sharpen these qualitative statements is to apply *Linv* to many Laplace transforms the inverses of which are known. This was done with 50 transforms. The numbers of significant figures used ranged from 8 to 17 (IBM 7094, single and double precision, CDC 3300). The T-values lay between 0 and 50. It was found that with increasing N the number of correct figures first increases nearly linearly and then, owing to the rounding errors, decreases linearly. The optimum N is approximately proportional to the number of digits the machine is working with. Table I was calculated using 8-digit arithmetic and $N = 10$.

With double precision arithmetic and $N = 18$ the number of correct figures doubles. The chosen N-values are about the optimum N for all functions of the table. Evaluating an unknown function from its Laplace transform, one should, nevertheless, compare the results for different N, to see whether the function is smooth enough, what accuracy can be reached, and what the optimum N is. Even then it is risky to rely solely on the results of *Linv*. One ought to be sure a priori that the unknown function $F(t)$ has not any discontinuities, salient points, sharp peaks, or rapid oscillations. Moreover, the accuracy should be checked by employing other inversion techniques.

TABLE I

T	$F(T)$	Fa	$F(T)$	Fa
	$F(t) = \frac{1}{\sqrt{\pi t}}$, $P(s) = \frac{1}{\sqrt{s}}$		$F(t) = -C - \ln(t)$, $P(s) = \ln(s)/s$	
1.0	0.56419	0.56555	−0.57722	−0.57782
2.0	0.39894	0.39912	−1.27036	−1.27084
3.0	0.32574	0.32655	−1.67583	−1.67544
4.0	0.28209	0.28278	−1.96351	−1.96392
5.0	0.25231	0.25174	−2.18665	−2.18727
6.0	0.23333	0.22989	−2.36898	−2.36870
7.0	0.21324	0.21322	−2.52313	−2.52270
8.0	0.19947	0.19956	−2.65666	−2.65740
9.0	0.18806	0.18814	−2.77444	−2.77390
10.0	0.17841	0.17796	−2.87980	−2.88091
	$F(t) = t^3/6$, $P(s) = 1/s^4$		$F(t) = e^{-t}$, $P(s) = 1/(s+1)$	
1.0	0.16667	0.16568	0.36788	0.36798
2.0	1.33333	1.32543	0.13534	0.13557
3.0	4.50000	4.47354	0.04979	0.05043
4.0	10.66667	10.60342	0.01832	0.01849
5.0	20.83333	20.70845	0.00674	0.00640
6.0	36.00000	35.78832	0.00248	0.00195
7.0	57.16667	56.82535	0.00091	0.00036
8.0	85.33333	84.82735	0.00034	−0.00006
9.0	121.50000	120.78473	0.00012	−0.00047
10.0	166.66667	165.66759	0.00005	−0.00020
	$F(t) = \sin(\sqrt{2}t)$, $P(s) = \sqrt{\frac{\pi}{2s^3}}\,e^{-1/(2s)}$		$F(t) = L_3(t)$, $P(s) = \frac{(s-1)^3}{s^4}$	
1.0	0.98777	0.98775	−0.66667	−0.66533
2.0	0.90930	0.91001	−0.33333	−0.32531
3.0	0.63816	0.63826	1.00000	1.02575
4.0	0.30807	0.30968	2.33333	2.39533
5.0	−0.02068	−0.02119	2.66667	2.78844
6.0	−0.31695	−0.31927	1.00000	1.21092
7.0	−0.56470	−0.57254	−3.66667	−3.32956
8.0	−0.75680	−0.76869	−12.33333	−11.82953
9.0	−0.89168	−0.91049	−26.00000	−25.28393
10.0	−0.97128	−0.98949	−45.66667	−44.88511

The inverses of the 50 test functions were also evaluated according to the inversion technique of Bellman et al. [1], which is based on the approximation of $F(t)$ by a polynomial in e^{-t}. It appeared that the algorithm *Linv* generally produces better results, i.e. the condition "$F(t)$ is everywhere smooth (in the sense described above)" is less restrictive than the condition "$F(-\ln(r))$ can be well approximated by a polynomial in $r = e^{-t}$ for $0 \leqq r \leqq 1$". The evaluation of the function $F(t) = t^2/2$ from its Laplace transform $P(s) = 1/s^3$ illustrates the difference between the two conditions: using *Linv* the inverse is correct within 0.1 percent, using the inversion technique described in [1] errors of hundreds of percents occur ($N = 10$, $0.1 < T < 10$).

The algorithm was successfully applied to renewal equations, differential-difference equations, and systems of partial differential equations. Reference [1] includes many other problems to which the algorithm can be applied.

References:

1. Bellman, R. E., Kalaba, R. E., and Lockett, J. *Numerical Inversion of the Laplace Transform.* American Elsevier, New York, 1966.
2. Gaver, D. P. Observing stochastic processes, and approximate transform inversion. *Oper. Res.* 14, 3 (1966), 444-459;

begin
integer i, ih, k, Nh, sn; **real** a; **array** $G[0{:}N]$, $H[1{:}N/2]$;
if $M = N$ **then go to** C;

```
  G[0] := 1;  Nh := N/2;
  for i := 1 step 1 until N do G[i] := G[i−1] × i;
  H[1] := 2/G[Nh−1];
  for i := 2 step 1 until Nh do
    H[i] := i ↑ Nh × G[2×i]/(G[Nh−i]×G[i]×G[i−1]);
  sn := 2 × sign (Nh−Nh÷2×2) − 1;
  for i := 1 step 1 until N do
  begin
    V[i] := 0;
    for k := (i+1) ÷ 2 step 1 until if i < Nh then i else Nh do
      V[i] := V[i] + H[k]/(G[i−k]×G[2×k−i]);
    V[i] := sn × V[i];
    sn := −sn
  end;
  M := N;
C: Fa := 0;  a := ln(2)/T;
  comment ln(2) should be replaced by its actual value
    0.69314... ;
  for i := 1 step 1 until N do
    Fa := Fa + V[i] × P(i×a);
  Fa := a × Fa
end
```

ALGORITHM 369
GENERATOR OF RANDOM NUMBERS SATISFYING THE POISSON DISTRIBUTION [G5]

HENRY E. SCHAFFER* (Recd. 27 Jan. 1969 and 16 July 1969)
North Carolina State University, Genetics Department, Raleigh, NC 27607

* This work was supported by Grants PR-00011 and GM-11546 of the National Institutes of Health.

KEY WORDS AND PHRASES: Poisson distribution, random number generator
CR CATEGORIES: 5.5

```
integer procedure poissrn (lambda);
  value lambda; real lambda;
comment At each call this procedure returns an observation
  from a Poisson distribution with parameter lambda. The rejec-
  tion method discussed by Kahn [1] is used. It requires an aver-
  age of lambda + 1 (pseudo) random numbers (uniformly dis-
  tributed on the 0, 1 interval) per call. For efficiency the random
  number generator should be coded in-line.
    This procedure is especially suitable when a small number of
  random numbers are needed from each of a large number of
  different Poisson distributions. This can occur when the Poisson
  parameter used in each call is itself chosen according to some
  probability distribution. Algorithm 342 [2] is more efficient for
  repeated use of the same value of the Poisson parameter.
    A value of −1 is returned to signal a value of lambda which
  is not positive. A value of −2 is returned to signal a value of
  lambda which is too large for the significance of the computer.
    I thank the referee for his suggestions and comments.
  REFERENCES:
1. KAHN, H. Applications of Monte Carlo. RM-1237-AEC, Rand
     Corp. 1956 (revised version).
2. SNOW, R. H. Algorithm 342, Generator of random numbers
     satisfying the Poisson distribution. Comm. ACM 11 (Dec.
     1968), 819;
if lambda ≤ 0.0 then poissrn := −1
else
begin
real z;
z := exp (−laamdb);
  if z = 0.0 then poissrn := −2
  else
  begin
    real t; integer k;
    real procedure random;
    begin
      comment The body of this procedure must be provided
        by the user to generate the uniformly distributed random
        numbers required by poissrn. The random number gen-
        erator is placed here rather than called as a global pro-
        cedure to decrease the time taken to obtain each random
        number. For the same reason a fast generator should be
        chosen. It is also important that this generator should
        have negligible serial correlation;
      ⟨procedure body⟩;
    end random;
    k := 0;  t := 1.0;
    for t := t × random while t > z do k := k + 1;
    poissrn := k
  end
end poissrn
```

ALGORITHM 370
GENERAL RANDOM NUMBER GENERATOR [G5]

EDGAR L. BUTLER (Recd. 20 June 1969 and 11 Aug. 1969)
Texas A & M University, College Station, TX 77840

KEY WORDS AND PHRASES: random number generator, probability density function, transformation, cumulative density function
CR CATEGORIES: 5.13, 5.5

Introduction. The algorithm below will generate random numbers from any probability density function, whether it be analytical, hypothetical, or experimentally acquired. Although there are in existence some fast and some general routines, the fast ones are for specific densities whereas the general algorithms are slow. As an example of a general algorithm, IBM's GPSS [7] uses the transformation theory of random deviates [4] to generate random numbers from any density function which can be described by data points. The GPSS algorithm is simple, and its precision is dependent upon the degree of interpolation and the number of points used for estimating the transformation function.

The program below has made the transformation method more accurate than the GPSS routine by using 257 points and linear approximation to the probability density function. Speed was acquired by appropriate organization of necessary tables. A time estimate for the performance of an assembly language program of the algorithm RANDG on an IBM 360/65 is about 33μsec for each generation.

Initialization. The operation of RANDG is based on vectors Q and R which can be derived by RANDGI as indicated below. An explanation of the routine RANDGI will give the reader some insight into the theory of RANDG.

1. Let (x_i, y_i), $i = 1, 2, \cdots, n$ be coordinates describing the probability density function, $y = f(x)$.

2. Using the trapezoidal rule, find $p_i = \int_{x_1}^{x_i} f(x)\,dx$ so that $p(x)$ approximates the cumulative density function of $f(x)$.

3. Let $x = p^{-1}(v)$, the inverse cumulative density function.

4. Find $q_j = p^{-1}(v_j)$ by using Lagrange's quadratic interpolation formula on $p^{-1}(v)$ for values of $v_j = j/256$ and $j = 0, 1, 2, \cdots, 256$ [5].

5. Compute $f(q_j)$ and let $r_j = (f(q_{j+1})-f(q_j))/(f(q_{j+1})+f(q_j))$ for $j = 0, 1, 2, \cdots, 255$. The $|r_j|$ is the ratio of the triangular area to the total area of a trapezoid approximating the probability density function between $x = q_j$ and $x = q_{j+1}$ (Figure 1) and the sign of r_j is the sign of the derivative. If the vectors Q and R are available to the experimenter, it is not necessary to use RANDGI. It should also be noted that RANDGI need be used only once for a

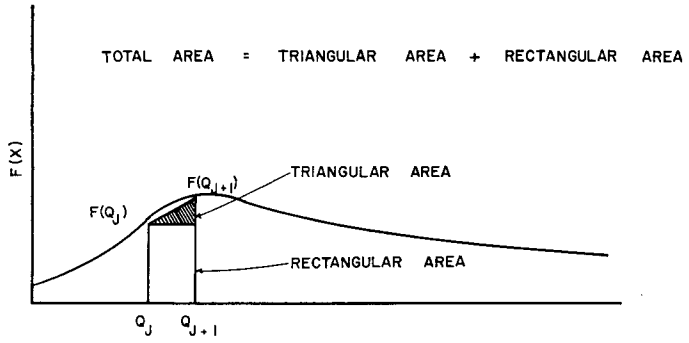

FIG. 1. Trapezoid approximating area under the probability density function from Q_J to Q_{J+1}

given density function and, therefore, does not usually affect the speed of generation.

Program. The routine RANDG then uses Q and R to generate the random ordinates in the following manner:

1. Select the jth interval with probability 1/256.

2. Let L_1 and L_2 be uniform random numbers on the interval (0, 1). It follows that $Y_1 = Q_j + (Q_{j+1} - Q_j)*L_1$ is uniformly random over the interval (Q_j, Q_{j+1}) and $Y_2 = Q_j + (Q_{j+1} - Q_j)*\max(L_1, L_2)$ is triangularly distributed on the same interval and is skewed left.

3. Let $P[Y=Y_1] = \mid R_j$ and $P[Y=Y_2] = 1 - \mid R_j$. Then Y is trapezoidally distributed with

$$f(Y) = \begin{cases} 2R_j(Y-Q_j)/(Q_{j+1}-Q_j)^2 \\ \quad + (1-R_j)/(Q_{j+1}-Q_j), & Q_j < Y < Q_{j+1}, \\ 0, & \text{otherwise.} \end{cases}$$

4. If $R_j < 0$ then use $Y_2 = Q_j + (Q_{j+1} - Q_j)*\min(L_1, L_2)$. The use of 256 intervals was arbitrary. For speed in assembly language on a binary machine a power of 2 should be used. It is possible that 128 or 64 values are adequate and the use of fewer than 256 would certainly save storage. (Note: Any good uniform random number generator may be used for selecting the interval and finding L_1 and L_2 [1, 2, 3, 6].)

REFERENCES

1. HULL, T. E., AND DOBELL, A. R. Random number generators. *SIAM Rev. 4* (July 1962), 230–254.
2. LEWIS, P. A. W., GOODMAN, A. S., AND MILLER, J. M. A pseudo-random number generator for the System/360. *IBM Syst. J. 8*, 2 (1969), 136.
3. MARSAGLIA, G., AND BRAY, T. A. One-line random number generators and their use in combinations. *Comm. ACM 11* (Nov. 1968), 757–759.
4. MOOD, A. M. *Introduction to the Theory of Statistics*. McGraw-Hill, New York, 1950, pp. 107–108.
5. SALVADORI, M. G., AND BARON, M. L. *Numerical Methods in Engineering* (2nd ed.). Prentice-Hall, Englewood Cliffs, N. J., 1964, pp. 88–91.
6. WHITTLESEY, J. RB. A comparison of the correlational behavior of random number generators for the IBM 360. *Comm. ACM 11* (Sept. 1968), 641–644.
7. General purpose simulation System/360 user's manual. No. H20-0326-3 (1968), IBM, White Plains, N. Y., pp. 26–35.


```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C  SUBROUTINE RANDG
C
C  PURPOSE
C    COMPUTE RANDOM NUMBERS FROM ANY GENERAL DISTRIBUTION.
C
C  USAGE
C    CALL RANDG (L,X,R,Y)
C
C  DESCRIPTION OF PARAMETERS
C    INPUT
C      L  -A NON ZERO ODD RANDOM INTEGER
C      X  -VECTOR OF LENGTH 257 CONTAINING ORDINATE POINTS
C          SEPERATED BY EQUAL PROBABILITY ON DESIRED DISTRIBUTION.
C          (CAN BE CALCULATED IN RANDGI).
C      R  -VECTOR OF LENGTH 256 CONTAINING RATIOS OF DERIVATIVE*DX
C          TO AREA/DX FOR EACH ORDINATE POINT IN X.
C          (CAN BE CALCULATED IN RANDGI).
C    OUTPUT
C      Y  -RANDOM NUMBER
C
C  REMARKS
C    QUADRATIC APPROXIMATION OF CDF (CUMULATIVE DENSITY FUNCTION)
C    WHICH IMPLIES LINEAR APPROXIMATION OF PDF (PROBABILITY DENSITY
C    FUNCTION).
C
C  SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED
C    NONE DIRECTLY.  RANDGI MAY BE USED FOR INITIALIZATION.
C
C  METHOD
C    TABLE LOOK UP PLUS UNIFORM AND TRIANGULAR DISTRIBUTION
C    VARIABLES ARE USED.
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      SUBROUTINE RANDG (L,X,R,Y)
      DIMENSION X(257),R(256)
C
C  GENERATE TWO UNIFORM RANDOM NUMBERS ON INTERVAL (1 - 2**31)
C  ANY GOOD GENERATOR MAY BE SUBSTITUTED.
C
      L1=IABS(65539*L)
      L=IABS(65539*L1)
      L2=L
C
C  CALCULATE TWO UNIFORM RANDOM NUMBERS
C  K1 INTEGER ON INTERVAL (1 - 256)
C  AK2 REAL ON INTERVAL (0 - 1.0)
C
      K1=L1/8388608+1
      AK2=FLOAT(MOD(L1,8388608))*1.192093E-7
      IF(AK2-ABS(R(K1))) 8,8,30
    8 IF(R(K1)) 20,10,10
C
C  CALCULATE TRIANGULAR RANDOM SKEWED LEFT
C
   10 Y=X(K1)+(X(K1+1)-X(K1))*AMAX0(L1,L2)*4.656613E-10
      RETURN
C
C  CALCULATE TRIANGULAR RANDOM SKEWED RIGHT
C
   20 Y=X(K1)+(X(K1+1)-X(K1))*AMIN0(L1,L2)*4.656613E-10
      RETURN
C
```

```
C  CALCULATE UNIFORM RANDOM
C
   30 Y=X(K1)+(X(K1+1)-X(K1))*FLOAT(L2)*4.656613E-10
      RETURN
      END
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C  SUBROUTINE RANDGI
C
C  PURPOSE
C    COMPUTE INITIALIZING VECTORS FOR RANDG
C
C  USAGE
C    CALL RANDGI (N,X,Y,P,Q,R,IER)
C
C  DESCRIPTION OF PARAMETERS
C    INPUT
C      N  -NUMBER OF (X,Y) POINTS OF APPROXIMATION TO PDF
C          (PROBABILITY DENSITY FUNCTION)
C      X  -VECTOR OF LENGTH N CONTAINING ORDINATE OF PDF
C      Y  -VECTOR OF LENGTH N CONTAINING ABSCISSA OF PDF
C    OUTPUT
C      P  -WORK VECTOR OF LENGTH N
C      Q  -VECTOR OF LENGTH 257 CONTAINING ORDINATE POINTS
C          SEPERATED BY EQUAL PROBABILITY ON DESIRED DISTRIBUTION.
C      R  -VECTOR OF LENGTH 256 CONTAINING RATIOS OF DERIVATIVE*DX
C          TO AREA/DX FOR EACH ORDINATE POINT IN Q.
C      IER-ERROR INDICATOR
C            1 - ERROR IN SCALING. I.E. TOTAL AREA OF PDF NOT EQUAL TO
C                1.  ASSUMING ESTIMATION ERRORS A FUDGE FACTOR IS USED
C                TO SCALE A RESULT.
C            2 - DENSITY NOT POSITIVE. I.E. SOME Y(I) LT 0.  ABORT
C            3 - NOT IN SORT. I.E. SOME X(I) LT X(I-1).  ABORT
C            4 - SEARCH ERROR.  SHOULD NEVER OCCUR BECAUSE OF FUDGE
C                FACTOR USED.  THIS MEANS SOME P(I) NOT LARGE ENOUGH
C                FOR SEARCH OF PROPER Q.  INVESTIGATION IS NEEDED.
C
C  REMARKS
C    NONE
C
C  SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED
C    NONE
C
C  METHOD
C    LINEAR APPROXIMATION OF PDF TO FIND CDF (CUMULATIVE DENSITY
C    FUNCTION) AND ICDF (INVERSE CDF).  QUADRATIC INTERPOLATION ON
C    ICDF TO FIND Q AND R.
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      SUBROUTINE RANDGI (N,X,Y,P,Q,R,IER)
      DIMENSION X(300),Y(300),P(300),Q(257),R(256)
C
C  CALCULATE CUMULATIVE PROBABILITIES
C
      IER=0
      IF(Y(1)) 5,10,10
C
C  ERROR 2
C
    5 IER=2
      RETURN
   10 P(1)=0.0
      DO 15 I=2,N
        IF(Y(I)) 5,11,11
   11   IF(X(I)-X(I-1)) 6,12,12
C
C  ERROR 3
C
    6   IER=3
        RETURN
   12   P(I)=(Y(I)+Y(I-1))*(X(I)-X(I-1))*0.5+P(I-1)
   15   CONTINUE
      IF(P(N)-0.996094) 7,7,16
   16 IF(P(N)-1.003906) 3,7,7
C
C  ERROR 1
C
    7 IER=1
    3 F=1.0/P(N)
      DO 4 I=2,N
    4   P(I)=P(I)*F
C
C  CALCULATE X POINTS FOR EQUAL-DISTANT CUMULATIVE PROBABILITIES
C
      V=0.0
      Q(1)=X(1)
      T1=Y(1)
      J1=2
  100 DO 150 I=2,257
        IF(I-257) 102,103,103
  102   V=V+3.90625E-3
C
C  LOCATE BEST POINT FOR INTERPOLATION
C
        DO 101 J=J1,N
          IF(P(J)-V) 101,104,105
  101     CONTINUE
C
C  ERROR 4
C
        IER=4
  103   J=N
  104   Q(I)=X(J)
        T2=Y(J)
        GO TO 125
  105   IF(J-3) 113,108,107
  107   IF(J-N) 108,111,111
  108   IF((P(J)-V)-(V-P(J-1))) 110,110,111
  110   J1=J-1
        GO TO 120
  111   J1=J-2
        GO TO 120
  113   J1=1
C
```

```
C  QUADRATIC INTERPOLATION OF P INVERSE FOR Q
C
  120   XT2=P(J1+2)-P(J1)
        XT3=P(J1+2)-P(J1+1)
        XT1=P(J1+1)-P(J1)
        XV1=V-P(J1)
        XV2=V-P(J1+1)
        XV3=V-P(J1+2)
        Q(I)=(XV3*XV2*X(J1))/(XT1*XT2)-(XV3*XV1*X(J1+1))/(XT1*XT3)+
     1    (XV2*XV1*X(J1+2))/(XT2*XT3)
C
C  LINEAR INTERPOLATION OF Y FOR T2 AND R
C
        T2=(Y(J)-Y(J-1))*(Q(I)-X(J-1))/(X(J)-X(J-1))+Y(J-1)
  125   R(I-1)=(T2-T1)/(T2+T1)
        T1=T2
        J1=J
  150   CONTINUE
      RETURN
      END
```

ALGORITHM 371
PARTITIONS IN NATURAL ORDER [A1]

J. K. S. McKay (Recd. 28 Apr. 1967)
California Institute of Technology, Mathematics Division, Pasadena, CA 91109.

KEY WORDS AND PHRASES: partitions, number theory
CR CATEGORIES: 5.39

```
procedure partition (p, k, last); integer n, k;
  integer array p; Boolean last;
comment Partition may be used to generate partitions in their
  natural (reverse lexicographical) order. On entry the first k
  elements of the global integer array p[1:n] should contain a par-
  tition, p[1] ≥ p[2] ≥ ··· ≥ p[k], of n into k parts. In order to ini-
  tialize m, the first entry must be made with last set true: this will
  result in p[1], p[2], ··· , p[k] and k remaining unaltered and last
  set false on exit. On all subsequent entries with last false, k is
  updated and p[1], p[2], ··· , p[k] will be found to contain the
  next partition of n with parts in descending order. On returning
  with the last partition, p[1] = p[2] = ··· = p[n], last is set
  true. To generate all partitions of n, p[1], k, last should be set
  to n, 1, true, respectively for the initial call: these variables
  must not be altered between successive calls for partition;
begin
  own integer m; integer t;
  if last then
  begin
    last := false;
    for m := 1 step 1 until k do
      if p[m] = 1 then go to c;
    m := k; go to c
  end;
  t := k - m;
  k := m;
  p[m] := p[m] - 1;
a: if p[k] > t then go to b;
  t := t - p[k];
  k := k + 1;
  p[k] := p[k-1];
  go to a;
b: k := k + 1;
  p[k] := t + 1;
  if p[m] ≠ 1 then m := k;
c: if p[m] = 1 then m := m - 1;
  if m = 0 then last := true;
end partition
```

ALGORITHM 372
AN ALGORITHM TO PRODUCE COMPLEX PRIMES, CSIEVE [A1]

K. B. Dunham (Recd. 29 July 1968 and 7 Oct. 1968)
Georgia Institute of Technology, School of Information Service, Atlanta, GA 30332

KEY WORDS AND PHRASES: primes, complex numbers
CR CATEGORIES: 5.39

```
procedure CSIEVE (m, PR, PI);
  value m; integer m; integer array PR, PI;
comment Primes can be defined in the complex domain, a + bi,
  where a and b are integers. A unity is ±1 or ±i. A unity times a
  prime is its associate. Primes are not unique among associates;
  but except for that ambiguity, all the ordinary rules of real
  primes, such as the unique factorization law, apply to complex
  primes.
    It can be shown that a complex integer is prime if and only if its
  conjugate is prime. Therefore it is sufficient to search for primes
  in the one-eighth plane area with a closed bound along y = 0
  and an open bound along x = y, where x is positive and y is less
  than x but nonnegative. Any prime found in that area has
  seven more associated primes: -x + yi, ±x - yi, ±y + xi,
  ±y - xi. A discussion of complex primes can be found in [1].
  It should be pointed out that numbers prime in the real domain
  are not necessarily prime in the complex domain, e.g. 2 =
  2 + 0i = (1+i)(1-i).
    Algorithms 35 [2], 310 [3], and 311 [4] generate real primes.
  The simplistic technique used by Algorithm 35 applies equally
  well to generating complex primes. Unfortunately the more
  efficient techniques of Algorithms 310 and 311 cannot easily be
  translated into complex prime sifters. This algorithm, CSIEVE,
  uses the result that a complex integer is prime if the square of
  its modulus is relatively prime to the square of the moduli of
  all previous primes. The procedure is called with a value m, the
  number of complex primes to generate, PR and PI, the real and
  imaginary parts of the prime list generated where PR > PI ≥ 0
  for each prime. The seven other associated primes must be gen-
  erated externally to CSIEVE.
  References:
  1. Hardy, G. H., and Wright, E. M. An Introduction to the
     Theory of Numbers. Clarendon Press, Oxford, 1954, Ch. 12.
  2. Wood, T. C. Algorithm 35, Sieve. Comm. ACM 4 (Mar. 1961),
     151.
  3. Chartres, B. A. Algorithm 310, Prime number generator 1.
     Comm. ACM 10 (Sept. 1967), 569.
  4. Chartres, B. A. Algorithm 311, Prime number generator
     2. Comm. ACM 10 (Sept. 1967), 570;
begin
  integer dn, nr, ni, sq, root, i, j, k;
  integer array PM[2:m];
  dn := PR[1] := PI[1] := PI[2] := 1; PM[2] := 5;
    j := PR[2] := 2;
```

```
  for nr := 3 step 1 until m do
  begin
    dn := 1 - dn;
    for ni := dn step 2 until nr - 1 do
    begin
      sq := nr × nr + ni × ni;
      root := entier (1.5×nr);
      for i := 2 step 1 until j do
      begin
        if ((sq÷PM[i]) × PM[i]) = sq then go to C;
        if root < PM [i] then go to A;
      end;
A:    for i := 2 step 1 until j do
      begin
        if PM[i] > sq then
        begin
          for k := j step -1 until i do
            PM[k+1] := PM[k];
          go to B;
        end
      end;
B:    PM[i] := sq; j := j + 1; PR[j] := nr; PI[j] := ni;
      if j = m then go to D;
C:    end
  end;
D:
end CSIEVE
```

The following Remark on Algorithm 282 by W. Gautschi and B. J. Klein relates to the paper by the same authors in the Numerical *Mathematics* department of this issue on pages 7–9.

REMARK ON ALGORITHM 282* [S22]
DERIVATIVES OF e^x/x, $\cos(x)/x$, AND $\sin(x)/x$

[Walter Gautschi, *Comm. ACM 9* (April 1966), 272]

WALTER GAUTSCHI AND BRUCE J. KLEIN (Recd. 12 May 1969)

Computer Sciences Department, Purdue University, Lafayette, IN 47907 and College of Arts and Sciences, Virginia Polytechnic Institute, Blacksburg, VA 24061

* Work supported by the National Aeronautics and Space Administration NASA under Grant NGR 15-005-039.

KEY WORDS AND PHRASES: recursive computation, successive derivatives, error control
CR CATEGORIES: 5.11, 5.12

For large values of x, and derivatives of order $n > x$, the first two procedures of Algorithm 282 incur substantial loss of accuracy. The reasons for this, as well as remedial measures, are described in the companion article [1]. The following revised procedures, based on this article, are believed to preserve accuracy as far as seems possible. Both procedures call upon the real procedure t of Algorithm 236 [2].

```
procedure dsubn (x, nmax, acc, machacc, d, error);
  value x, nmax, acc, machacc; integer nmax, acc, machacc;
  real x; array d; label error;
comment Given x ≠ 0, nmax, and the number machacc of decimal
  digits available in the mantissa of machine floating-point num-
  bers, this procedure generates the derivatives

        d_n(x) = (d^n/dx^n)(e^x/x),    n = 0, 1, 2, ···, nmax,

  to an accuracy of acc significant decimal digits, except near a
  zero of d_n(x), where some significance may be lost. The result
  d_n(x) is stored in d[n]. If x = 0, the procedure immediately
  exits to the label error;
begin
  integer n0, min, n, n1; real x1, e, a, q;
  Boolean bool1, bool2;
  if x = 0 then go to error;
  x1 := abs(x); n0 := x1; e := exp(x);
  d[0] := e/x;
  a := 1.1513 × (machacc-acc) - .3466;
  if a < 2 then a := 2;
  bool1 := x < 0 ∨ x1 ≦ a; bool2 := n0 < nmax;
  min := if bool2 then n0 else nmax;
  for n := 1 step 1 until if bool1 then nmax else min do
    d[n] := (e-n×d[n-1])/x;
  if (¬bool1) ∧ bool2 then
  begin
    n1 := 2.7183 × x1 ×
      t((x1+2.3026×acc+.6932)/(2.7183×x1))-1;
    if n1 < nmax then n1 := nmax;
    q := 1/x;
    for n := 1 step 1 until n1 + 1 do q := -n × q/x;
    for n := n1 step -1 until n0 + 1 do
    begin
      q := (e-x×q)/(n+1);
      if n ≦ nmax then d[n] := q
    end
  end
end dsubn;
procedure csubn (x, nmax, acc, machacc, c, error);
  value x, nmax, acc, machacc; integer nmax, acc, machacc;
  real x; array c; label error;
comment This procedure generates the derivatives

        c_n(x) = (d^n/dx^n)(cos x / x    for n = 0, 1, 2, ···, nmax),

  and stores them in the array c. The parameters acc, machacc
  have the same meaning as in the preceding procedure. There is
  an error exit if x = 0;
begin
  integer n0, min, n, n1; real x1, a, q; array tau[1:4];
  Boolean bool1, bool2;
  if x = 0 then go to error;
  x1 := abs(x); n0 := x1;
  tau[1] := -sin(x); tau[2] := -cos(x);
  tau[3] := -tau[1]; tau[4] := -tau[2];
  c[0] := tau[4]/x;
  a := 2.3026 × (machacc-acc) - .69315;
  if a < 3 then a := 3;
  bool1 := x1 ≦ a; bool2 := n0 < nmax;
  min := if bool2 then n0 else nmax;
  for n := 1 step 1 until if bool1 then nmax else min do
    c[n] := (tau[n-4×((n-1)÷4)]-n×c[n-1])/x;
  if (¬bool1) ∧ bool2 then
  begin
    n1 := 2.7183 × x1 × t((2.3026×acc+.6932)/(2.7183×x1))-1;
    if n1 < nmax then n1 := nmax;
    q := 1/x;
    for n := 1 step 1 until n1 + 1 do q := -n × q/x;
    for n := n1 step -1 until n0 + 1 do
```

```
    begin
      q := (tau[n+1-4×(n÷4)]-x×q)/(n+1);
      if n ≤ nmax then c[n] := q
    end
  end
end csubn
```


REFERENCES:

1. GAUTSCHI, WALTER, AND KLEIN, BRUCE J. Recursive computation of certain derivatives—A study of error propagation. *Comm. ACM 13* (Jan. 1970), 7–9.
2. GAUTSCHI, WALTER. Algorithm 236, Bessel functions of the first kind [S17]. *Comm. ACM 7* (Aug. 1964), 479–480.

REMARK ON ALGORITHM 347 [M1] AN EFFICIENT ALGORITHM FOR SORTING WITH MINIMAL STORAGE

[Richard C. Singleton, *Comm. ACM 12* (Mar. 1969), 185]

ROBIN GRIFFIN AND K. A. REDISH (Recd. 14 Apr. 1969 and 11 Aug. 1969)

McMaster University, Hamilton, Ontario, Canada

KEY WORDS AND PHRASES: sorting, minimal storage sorting, digital computer sorting

CR CATEGORIES: 5.31

The algorithm was tested on the CDC 6400 ALGOL compiler (version 1.1, running under the SCOPE operating system, version 3.1.4). One trial was made using an array of 5000 pseudorandom numbers; the results were correct.

The central processor time was about 6.9 seconds corresponding to a value for K (defined below) of about 110 microseconds.

It would be more in the spirit of ALGOL to follow QUICKERSORT [1] and give arrays *IL* and *IU* dynamic bounds. This involves changing line 4 on page 187 from

```
integer array IL, IU[0:8];
```

to

```
integer array IL, IU[0:ln(j-i+1)/ln(2)-0.9];
```

The FORTRAN subroutine given in the comments to the algorithm was tested on a CDC FORTRAN compiler (the RUN compiler version 2.3, running under the SCOPE operating system, version 3.1.4). Tests were made with each of the five initial orderings described with the algorithm for a variety of array lengths from 500 to 40,000. For integer arrays, the results were correct; but when the actual argument corresponding to the dummy argument A was a real array containing large positive and negative numbers, errors occurred. This does not invalidate the subroutine, but the comments should be changed to

```
C  SORTS INTEGER ARRAY A INTO INCREASING ORDER, FROM A(II) TO A(IJ)
C  ARRAYS IU(K) AND IL(K) PERMIT SORTING UP TO 2**(K+1) - 1 ELEMENTS
C  THE USER SHOULD CONSIDER THE POSSIBILITY OF INTEGER OVERFLOW
C  THE ONLY ARITHMETIC OPERATION ON THE ARRAY ELEMENTS IS SUBTRACTION
```

This gives enough information (and a hint) but leaves the responsibility for any abuse of American National Standards Institute (formerly USASI) FORTRAN where it belongs—with the user.

The subroutine was also tested on the IBM 7040 FORTRAN compiler (the IBFTC compiler running under the IBSYS operating system, version 9 level 10). The results were correct. The statement

```
INTEGER A, T, TT
```

was removed and the amended subroutine tested using similar, but real, arrays. The results were again correct; running times increased by up to 5 percent on the CDC 6400 and were unchanged on the IBM 7040.

Tables I and II summarize the information on running times in terms of K, where

$$\text{time} = Kn \log_2 n$$

(runs of other lengths are omitted for brevity).

TABLE I. SORTING TIMES
K in microseconds where time $= Kn \log_2 n$

Test	Method			
Original order and number of items	Burroughs 5500 ALGOL*	CDC 6400 FORTRAN (REAL)	CDC 6400 FORTRAN (INTEGER ARRAY)	IBM 7040 FORTRAN
Random uniform				
500	107	21.2	20.5	
1000	102	21.7	20.5	
5000		21.1	20.2	269
10000		21.1	20.1	263
40000		21.2	20.1	
Natural order				
500	65	12.9	12.5	
1000	62	13.1	12.4	146
5000		12.6	11.9	148
10000		12.7	12.0	
40000		12.9	12.1	
Reverse order				
500	67	14.3	13.4	
1000	63	13.9	13.4	
5000		13.4	12.7	158
10000		13.4	12.7	158
40000		13.5	12.8	
Sorted by halves				
500	163	34.8	32.6	
1000	173	37.1	35.1	
5000		39.5	37.2	465
10000		41.8	39.3	491
40000		46.6	44.1	
Constant value				
500	96	19.2	18.5	
1000	97	19.4	18.7	
5000		19.4	18.7	237
10000		19.9	19.0	241
40000		20.2	19.5	

* Calculated from Singleton's results

TABLE II. VALUES OF $n \log_2 n$

n	500	1000	5000	10000	40000
$10^{-6}\, n \log_2 n$	0.00448	0.00996	0.0614	0.1329	0.6115

For use as a library routine one slight change is recommended: JJ−II should be tested on entry and a suitable error message produced if negative. It would be possible to transfer "work" arrays to replace IU and IL thus allowing the user more control of storage allocation, but the additional instructions needed to handle the extra arguments reduce the saving and this is hardly worthwhile.

The authors would like to thank the referee for his helpful comments.

REFERENCE:

1. SCOWEN, R. S. Algorithm 271, Quickersort. *Comm. ACM 8* (Nov. 1965), 669–670.