Multivariate Normal Probabilities of Star-shaped Regions

By

Sharon L. Lohr

Technical Report No. 515

School of Statistics

University of Minnesota

July 16, 1988

Language

Fortran 77

Description and Purpose

Let X be an n-variate normal random vector with mean zero and positive semidefinite covariance matrix Σ of rank q. This algorithm provides an approximation to

$$p = P\{X \in A\}, \tag{1}$$

where A may be any compact region which is star-shaped with respect to zero (i.e., if x is in A, then any point between zero and x is also in A). The user must supply a Fortran function describing the boundary of A.

Form the Cholesky decomposition $\Sigma = T'T$; if Σ is indefinite, the Cholesky factor chosen is the one given by Healy (1985). Then

$$P\{X \in A\} = P\{T'JZ \in A\},$$

where $Z \sim N(0, I_q)$ and J is the $n \times q$ matrix in which $J_{ij} = 1$ if the i^{th} diagonal entry of T is the j^{th} nonzero diagonal entry of T and zero otherwise. Now let Y be uniformly distributed on the unit q-sphere S. Then $T'JY$ is uniformly distributed on $T'JS$, the unit contour of constant density for the $N(0, \Sigma)$ distribution. For any realization y of Y, let $s(y)$ be the distance from the origin to the boundary A in the direction specified by the n-vector $T'Jy$ and let $r(y)$ be $s(y)$ divided by the length of $T'Jy$. Then

$$P\{X \in A | Y = y\} = P\{\chi_q^2 \leq r^2(y)\},$$

so

$$P\{X \in A\} = E_Y \left[P\{\chi_q^2 \leq r^2(Y)\}\right].$$

A simple estimate of the probability p is thus given by

$$\hat{p} = \frac{1}{k} \sum_{i=1}^{k} P\{\chi_q^2 \leq r^2(y_i)\} \tag{2}$$

where the y_i are randomly oriented vectors of length one.

The efficiency of the crude Monte Carlo estimate in (2) is improved by using the antithetic variates employed in Deàk's (1980a, 1980b) algorithms which calculate values of

the multivariate normal cumulative distribution function. Generate q vectors on the unit sphere using the multivariate normal distribution; Deàk (1979) showed that the multivariate normal method is faster than others such as polar transformations or acceptance/rejection from uniform variates generated in a hypercube when $q > 4$. From these, construct an orthonormal system $y_1, \ldots, y_q$ on the unit sphere using the Gram-Schmidt method. Then calculate $P\{\chi_q^2 \leq r^2(z_i)\}$ for the $2q$ vectors $\{y_1, \ldots, y_n, -y_1, \ldots, -y_n\}$ and the $2q(q-1)$ vectors given by $(\pm y_i \pm y_j)/\sqrt{2}$ for all combinations of $i \neq j$. Since these $2q^2$ vectors are evenly spaced on the unit q-sphere, for many asymmetric regions the variance of an estimate $\hat{p}$ produced by these vectors will be less than the variance of an estimate from $2q^2$ completely randomly-oriented vectors. The same is true for symmetric regions if the q^2 vectors $\{y_1, \ldots, y_n\}$ and $(y_i \pm y_j)/\sqrt{2}$ are used. In addition, only q random vectors need be generated for each iteration.

The algorithm determines the number of iterations to be performed by finding $\hat{p}_i$ according to (2) for a series of 100 sets of $2q^2$ vectors. It then estimates the variance of $\hat{p}_i$ from the first 100 iterations, and uses Cox's (1952) two-stage procedure to calculate the number of iterations needed to estimate the probability with user-determined standard error.

A Fortran function describing the boundary of the region A is supplied by the user. This function returns the value $f(x) = 0$ if x is a point on the boundary of A, returns a negative value if x is inside A, and returns a positive value if x is outside A. The distance $r(y)$ is then calculated to be the root r of the equation $f(r * T'y) = 0$; the Illinois version of the secant method described in Dowell and Jarrett (1971) is used to find the root. Although the Illinois method requires that the region A be compact, probabilities of quadrants and other infinite regions may be calculated by forcing the appropriate boundary of A to be sufficiently large.

Structure

subroutine mulnor(covar,n,f,sdev,rhigh,iseed,t,work,prob,ifault)

Formal parameters

covar	Real$(n(n+1)/2)$	input: lower triangle of the symmetric covariance matrix, stored row-wise as a one-dimensional array.
n	Integer	input: order of covariance matrix
f	Real	input: function specifying the boundary. The form of f is described under *Auxiliary subroutines.* The function f must be declared external in the calling program.
sdev	Real	input: desired standard deviation of estimated probability
rhigh	Real	input: radius of sphere circumscribing region A
iseed	Integer	input: seed for random number generator; iseed may be any non-zero integer.
t	Real$(n(n+1)/2)$	workspace:
work	Real$(n, n+2)$	workspace:
prob	Real	output: estimate of (1)
ifault	Integer	output: an error indicator equal to:
		: 0 if normal return;
		: 1 if n is less than one;
		: 2 if Σ is not positive semidefinite;
		: 3 if *rhigh* $<$ distance to boundary;
		: 4 if more than 500 iterations are required to calculate $r(y)$;
		: 5 if more than 100000 iterations are needed to estimate p to the desired precision.

Auxiliary subroutines

1. Subroutine *chol(a,n,nn,u,nullty,ifault)*, developed by Healy (1985), is used to find the Cholesky decomposition of the covariance matrix in a.

2. Subroutines *tdiag(n,tol,a,d,e,z,ifault)* and *lrvt(n,precis,d,e,z,ifault)*, from Sparks and Todd (1985), are used to find the minimum nonzero eigenvalue of the covariance matrix. Algorithm *tdiag* is modified to operate on a packed covariance matrix by changing the declaration *real a(n,n)* to *real a(*)* and replacing the lines

```
                                                    k = 0
      do 10 i=1,n                                   do 10 i=1,n
      do 10 j=1,i                  by               do 10 j=1,i
10    z(i,j) = a(i,j)                               k = k + 1
                                               10   z(i,j) = a(k)
```

3. Subroutine *orthp(f,rhigh,n,q,t,z,prob,ifault)* calculates $\hat{p}_i$ for each iteration.

4. Subroutine *zerovc(f,r,r0,r1,n,v,work,ifault)*, based on the algorithm in Dowell and Jarratt (1971), calculates the solution r to $f(r * vector) = 0$.

5. Function *chiprb(x,df)*, translated from the Pascal algorithm of Hill and Pike (1967), is used to calculate $P\{\chi^2_{df} > x\}$.

6. Function *f(n,v)* is the user-supplied function defining the boundary of A. The formal parameters are

n	Integer	input: order of covariance matrix
v	Real(n)	input: vector to be evaluated

The function should take the form

$$f(n,v)\begin{cases} = 0 & \text{if } v \text{ is on the boundary of } A \\ < 0 & \text{if } v \text{ is an interior point of } A \\ > 0 & \text{otherwise.} \end{cases}$$

The algorithm is fastest is f is continuous along rays from the origin; for example, if A is the unit cube, a good choice of f would be $f(n,v) = \max(|v_i|) - 1$. It will converge, however, even if f is defined to be -1 inside A, and +1 outside A.

7. The function *rnor* from the *CMLIB* public domain software library is used to generate random $N(0,1)$ observations in subroutines *mulnor* and *orthp*.

Constants The constants *big* and *small* in subroutine *chiprb* are set to a large and a small number, respectively. The constant *eta* in subroutines *mulnor*, *chol*, and *zerovc* should be set according to the precision of the arithmetic.

Restrictions

The storage requirement for the algorithm is roughly $2n^2$. Storage capacity on the machine may limit the orders of covariance matrices which may be treated. The storage problem may be avoided by working with single vectors rather than with orthonormal systems, although such a modification will slow the algorithm considerably. In the single precision version of the algorithm, the input value of the desired standard deviation of the estimate should be larger than .0005; at values smaller than .0005, numerical errors may begin to contaminate the result.

Precision

A double precision version can be prepared by changing the *real* declarations to *double precision* and changing the single precision statement functions *zerf(a) = erf(a), zsqrt(a) = sqrt(a), and zabs(a) = abs(a)* in the subroutines to their double precision equivalents. Note that since *rnor* produces a real-valued normal variate, the declaration of *rnor* as real should be retained; the statement function *znor(k) = rnor(k)* should be changed to *znor(k) = dfloat(rnor(k))*.

Accuracy and Timing

There are two main sources of numerical error in this algorithm. The first is in finding the point on the boundary of A corresponding to a directional vector. This error is controlled by the parameter *eta*; the maximum error from this source would occur if the boundary function were such that the distance to the boundary was consistently under- or over-estimated. Such functions, however, are exactly the ones for which few iterations are required to calculate the probability, since the value of $r(y)$ will be nearly constant relative to the convergence criterion. Thus if *eta* is close to the precision of the arithmetic, the error from the first source is negligible. The second

error comes from the fact that any Monte Carlo or pseudo Monte Carlo estimate is the realization of a random variable and hence has an associated standard deviation. A tolerable standard deviation for the computed probability is input by the user; because Cox's (1952) double-sampling procedure is used in the implementation with the first sample of size 100, the actual standard deviation of some of the estimates is less than the inputted value of the standard deviation.

Execution time depends on the rank of the covariance matrix, on the complexity of the boundary function, and on the true value of p. Sample timings for various problems are presented in Table 1. All computations were performed on a Sun 3/50 workstation with 32-bit word length. All examples were run with a desired standard deviation of 0.0005

Related Algorithms

A very fast algorithm for calculating the cumulative distribution function bivariate normal probabilities is found in Donnelly (1973). For small dimensions, we recommend that a multivariate normal probability be calculated using quadrature methods. The algorithm of Schervish (1984), based on that of Milton (1972), efficiently and accurately calculates multivariate normal probabilities for rectangular regions and is much faster than *mulnor* when it can be applied. The present algorithm is most useful for calculating multivariate normal probabilities of oddly-shaped regions or in high dimensions, and for calculating probabilities for indefinite covariance matrices.

Acknowledgements

Work on this algorithm was partially supported by grant 135-1001 from the graduate school of the University of Wisconsin-Madison and by a summer research fellowhip from the graduate school of the University of Minnesota. All computations were performed on a Sun 3/50 workstation with MC68881 floating-point processor at the University of Minnesota. The author would like to thank Gary Oehlert for helpful discussions.

Σ	Region A	Time (sec)	exact p	estimated $\hat{p}$
10×10 identity	unit 10-cube	124	.0220	.0220
20×20 identity	20-cube, distance to face = 3	2457	.9474	.9481
$\begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$	unit square	44	.4980	.4981
$\begin{bmatrix} 1.0 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1.0 & 0.5 & 0.5 \\ 0.5 & 0.5 & 1.0 & 0.5 \\ 0.5 & 0.5 & 0.5 & 1.0 \end{bmatrix}$	$\{x \in \Re^4 : -\infty < x_i \leq 2\}$	25	.9285	.9284
$\begin{bmatrix} 1.0 & 0.3 \\ 0.3 & 1.0 \end{bmatrix}$	$\{x \in \Re^2 : x'\Sigma^{-1}x \leq 1\}$	9	.3935	.3935
$[1\,2\,3\,4]' \times [1\,2\,3\,4]$	unit sphere, radius 2	2	.2850	.2850
2×2 identity	regular pentagon, distance from 0 to vertex = 1	4	.3142	.3141
2×2 identity	regular pentagram, distance from 0 to point = 2.618	3	.6413	.6413

Table 1: Times required to estimate p. Upper bound on standard deviation is 0.0005.

References

Cox, D. R. (1952) Estimation by double sampling. *Biometrika*, **39**, 217-227.

Deàk, I. (1979) Comparison of methods for generating uniformly distributed random points in and on a hypersphere. *Problems in Control and Information Theory*, **8**, 105-113.

Deàk, I. (1980a) Computation of multiple normal probabilities, in *Recent Advances in Stochastic Programming*, P. Koll and A. Predorr, eds. New York: Springer-Verlag.

Deàk, I. (1980b) Three digit accurate multiple normal probabilities. *Numerische Mathematik*, **35**, 369-380.

Donnelly, T. G. (1973) Algorithm 462. Bivariate normal distribution. *Comm. ACM.*, **16**, 638.

Dowell, M. and Jarratt, P. (1971) A modified *regula falsi* method for computing the root of an equation. *BIT*, **11**, 168-174.

Healy, M. J. R. (1985) Triangular Decomposition of a symmetric matrix, in *Applied Statistics Algorithms*, P. Griffiths and I. D. Hill, eds. Chichester: Ellis Horwood.

Hill, I. D. and Pike, M. C. (1967) Algorithm 299, Chi-squared integral. *Comm. ACM.*, **10**, 243-244.

Milton, R. C. (1972). Computer evaluation of the multivariate normal integral. *Technometrics*, **14**, 881-889.

Schervish, M. J. (1984), Multivariate normal probabilities with error bound. *Applied Statist.*, **13**, 81-94. Correction (1985), **34**, 103-104.

Sparks, D. N. and Todd, A. D. (1985) Latent roots and vectors of a symmetric matrix, in *Applied Statistics Algorithms*, P. Griffiths and I. D. Hill, eds. Chichester: Ellis Horwood.

```
      subroutine mulnor(covar,n,f,sdev,rhigh,iseed,t,work,prob,ifault)
c
c  Finds the probability that a normally distributed random
c  n-vector with mean 0 and covariance covar falls
c  in area enclosed by the external user-defined function f.
c
      integer n,iseed,ifault,
     *          k,ii,kk,q,iter,nn,nullty,i,j,int
      real f,sdev,rhigh,prob,p,var,zero,cox,eta,
     *       covar(*), work(n,*),t(*),znor
      real rnor
c
      external f
      znor(k) = rnor(k)
      data zero,cox,eta /0.0,1.02,1.0e-5/
c
c  Initialize random normal generator
c
      ifault = 0
      p = znor(iseed)
c
c  Find Cholesky decomposition of covariance matrix
c
      nn = n*(n+1)/2
      call chol(covar,n,nn,t,nullty,ifault)
      q = n - nullty
      if (ifault .ne. 0) return
c
c  Find smallest nonzero eigenvalue of covariance matrix
c
      call tdiag(n,eta,covar,work(1,1),work(1,2),work(1,3),ifault)
      call lrvt(n,eta,work(1,1),work(1,2),work(1,3),ifault)
      rhigh = cox*rhigh/work(nullty+1,1)
c
c  Transpose Cholesky factor, omitting columns of zeroes.
c  Resulting vector is packed form of T'J,
c  written rowwise beginning with last row.
c
      do 40 i = 1,n
      do 30 j = 1,i
         work(j,i) = zero
30    continue
40    continue
      ii = 0
      do 60 i = 1,n
```

```
      do 50 j = 1,i
         ii = ii + 1
         work(i,j) = t(ii)
50    continue
60    continue
      j = 0
      do 90 i = n,1,-1
         if (j .eq. nullty) go to 100
         if (work(i,i) .eq. zero) then
             j = j+1
             do 80 k = i+1,n
             do 70 kk = i, n-j
             work(k,kk) = work(k,kk+1)
70           continue
80           continue
         end if
90    continue
100   continue
      k = 0
      do 120 j = n,1,-1
      do 110 i = 1,min0(j,q)
         k = k + 1
         t(k) = work(j,i)
110   continue
120   continue
c
c  Take pilot sample; estimate variance of prob from orthp.
c
      prob = zero
      var = zero
      do 130 k=1,100
         call orthp(f,rhigh,n,q,t,work,p,ifault)
         if (ifault .ne. 0) return
         prob = prob + p
         var = var + p*p
130   continue
      prob = 0.01*prob
      var = 0.01*var - prob*prob
c
      iter = int(cox*var/(sdev*sdev)) + 1
      if (iter .gt. 100000) ifault = 5
c
      do 140 k=101,iter
         call orthp(f,rhigh,n,q,t,work,p,ifault)
         if (ifault .ne. 0) return
         prob = prob + (p - prob)/k
```

```
140     continue
c
        return
        end



        subroutine orthp(f,rhigh,n,q,t,z,prob,ifault)
c
        integer n,ifault,q,i,j,k,jj,ii,qp1,qp2
        real t(*),rhigh,z(n,*),prob,s,zprod,a,b,r,rt2rcp,zero,
     *            f,chiprb,zsqrt,znor
        real rnor,sqrt
c
        external f
        data zero,rt2rcp /0.0,0.707106781/
        zsqrt(s) = sqrt(s)
        znor(i) = rnor(i)
c
        prob = zero
        qp1 = q+1
        qp2 = q+2
c
c  Put randomly oriented orthonormal system of q-vectors in z.
c  Orthonormalization is achieved via the Gram-Schmidt procedure.
c
        do 20 i=1,q
           do 10 k=1,q
              z(k,i) = znor(0)
10         continue
20      continue
        s = zero
        do 30 k=1,q
           s = s + z(k,1)*z(k,1)
30      continue
        s = zsqrt(s)
        do 40 k=1,q
           z(k,1) = z(k,1)/s
40      continue
        do 100 i = 2,q
           do 70 j = 1,i-1
              zprod = zero
              do 50 k=1,q
                 zprod = zprod + z(k,i)*z(k,j)
50            continue
              do 60 k = 1,q
```

```
                z(k,i) = z(k,i) - zprod*z(k,j)
60            continue
70          continue
            s = zero
            do 80 jj=1,q
               s = s + z(jj,i)*z(jj,i)
80          continue
            s = zsqrt(s)
            do 90 k = 1,q
               z(k,i) = z(k,i)/s
90          continue
100       continue
c
c  Replace each column of z by trans(Cholesky factor)*column of z.
c
          do 130 i=1,q
             jj = 0
             do 120 j = n,1,-1
                s = zero
                do 110 k=1,min0(j,q)
                  jj = jj + 1
                  s = s + t(jj)*z(k,i)
110             continue
                z(j,i) = s
120          continue
130       continue
c
c  Estimate the probability that N(0,covar) is within the
c  boundary given by f.
c
          do 140 i = 1,q
             a = zero
             b = rhigh
             call zerovc (f,r,a,b,n,z(1,i),z(1,qp2),ifault)
             prob = prob - chiprb(r*r,q)
140       continue
c
          do 160 i = 1,q
             a = zero
             b = rhigh
             do 150 k = 1,n
               z(k,q+1) = -z(k,i)
150          continue
             call zerovc (f,r,a,b,n,z(1,qp1),z(1,qp2),ifault)
             prob = prob - chiprb(r*r,q)
160       continue
```

```
c
      do 210 i = 1,q-1
        do 200 j = i+1,q
           do 190 jj= -1,1,2
             do 180 ii = -1,1,2
                do 170 k=1,n
                   z(k,q+1) = rt2rcp*(z(k,i)*ii + z(k,j)*jj)
170             continue
                a = zero
                b = rhigh
                call zerovc (f,r,a,b,n,z(1,qp1),z(1,qp2),ifault)
                prob = prob - chiprb(r*r,q)
180          continue
190        continue
200     continue
210   continue
      prob = prob/(2*q*q) + 1.0
c
      return
      end



      subroutine zerovc (f,r,r0,r1,n,v,y,ifault)
c
c  Calculates the solution r to f(r*v) = 0.
c
      integer n,ifault
      real f,r,r0,r1,eta,v(n),y(n),f0,f1,fr,half,zabs
      real abs
      integer i,j
      external f
      data half, eta /0.5,1.0e-05/
      zabs(r) = abs(r)
c
      do 10 i=1,n
         y(i) = r0*v(i)
10    continue
      f0 = f(n,y)
      do 20 i=1,n
         y(i) = r1*v(i)
20    continue
      f1 = f(n,y)
      if (sign(half,f1) .eq. sign(half,f0)) then
         ifault = 3
      end if
```

```
      do 40 j = 1,500
         r = r1 - f1*(r1-r0)/(f1-f0)
         do 30 i=1,n
           y(i) = r*v(i)
30       continue
         fr = f(n,y)
         if (zabs(r-r1) .lt. eta) go to 50
         if (sign(half,fr) .ne. sign(half,f1)) then
           r0=r1
           f0=f1
         else
           f0 = f0 * half
         end if
         r1 = r
         f1 = fr
40    continue
      ifault = 4
50    continue
      return
      end




      real function chiprb(x,df)
c
c  Finds the probability that a chi-squared random variable
c  with df degrees of freedom exceeds x.
c
      real x,chiprb,zerf,zsqrt,
     *       a,sqrta,y,c,e,z,s,big,small,rtpirp,zero,one,half
      real erf,sqrt
      integer df,odd,n2,i
c
      parameter  (rtpirp =0.564189583548)
c  Note:   0.564189583548 = 1/sqrt(pi)
      data big/1.0e+10/,small/1.0e-10/,zero/0.0/,one /1.0/,half/0.5/
      zerf(a) = erf(a)
      zsqrt(a) = sqrt(a)
c
      s = zero
      if (x .gt. big) go to 50
      s = one
      if (x .lt. small) go to 50
c
      a = half*x
```

```
      y = exp(-a)
      odd = mod(df,2)
      if (odd .eq. 0) then
         s = y
         e = one
         z = zero
      else
         sqrta = zsqrt(a)
         s = one + zerf(-sqrta)
         e = rtpirp/sqrta
         z = -half
      end if
      if (df .lt. 3) go to 50
c
      n2 = df/2 - 1 + odd
      c = zero
      do 40 i = 1,n2
      z = z + 1
      e = e*(a/z)
      c = c + e
40    continue
      s = s + c*y
c
50    continue
      chiprb = s
      end



      real function f(n,v)
      real v(n),f
      integer n
      real f,cubsz
      integer i
      data cubsz /1.0/
c
c  Boundary of region is an n-dimensional cube.
c  Cubsz is the distance from the origin to the side of the cube
c
      f = 0.0
      do 20 i = 1,n
      f = amax0(f,abs(v(i)))
20    continue
      f = f - cubsz
      end
```