Stat Comput
DOI 10.1007/s11222-014-9468-y

Bivariate conditioning approximations for multivariate normal probabilities

Giang Trinh · Alan Genz

Received: 22 July 2013 / Accepted: 25 March 2014

Abstract New formulas are derived for multivariate normal probabilities defined for hyper-rectangular probability regions. The formulas use conditioning with a sequence of bivariate normal probabilities. The result is an approximate formula for multivariate normal probabilities which uses a product of bivariate normal probabilities. The new approximation method is compared with approximation methods based on products of univariate normal probabilities, using tests with random covariance-matrix/probability-region problems for up to twenty variables. The reordering of variables is studied to improve efficiency of the new method.

Keywords Multivariate normal · Bivariate conditioning approximation · Multinomial probit probability

1 Introduction

Many problems in applied statistical analysis require the computation of multivariate normal (MVN) probabilities. One large application area is transportation modeling, where the use of multinomial probit models requires the calculation of numerous probit choice probabilities (see, for example, Connors et al. 2014; Ochi and Prentice 1984; McFadden and Train 2000), which can be written as MVN probabilities in the form

$$\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) = \frac{1}{\sqrt{|\Sigma|\,(2\pi)^n}} \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} e^{-\frac{1}{2}\mathbf{x}^t \Sigma^{-1}\mathbf{x}} d\mathbf{x},$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$, $d\mathbf{x} = dx_n dx_{n-1} \cdots dx_1$, and Σ is an $n \times n$ symmetric positive definite covariance matrix. There are in general no "exact" methods for the computation of the MVN probabilities, so various methods (see Genz and Bretz 2009) have been developed to provide suitably accurate approximations. And now there are implementations in scientific computing environments of efficient simulation methods (see the R *pvmnorm* package and Matlab *mvncdf* function, for example), which can often provide highly accurate MVN probabilities. There are some types of problems, where (for example Σ or Σ^{-1} is tridiagonal, or Σ has the form $D + \mathbf{v}\mathbf{v}^t$ for some diagonal matrix D) very accurate approximations can be quickly computed. The simulation methods for the general MVN problem often take increasingly large computation times for accurate results as n increases, so there is still an interest in lower-cost and more direct approximation methods, as evidenced by the recent paper by Connors et al. (2014), where several approximation methods were compared.

The purpose of this paper is to consider generalizations of some approximation methods which use univariate conditioning. These univariate conditioning approximations were first studied by Mendell and Elston (ME, 1974; but see Kamakura 1989, for a more complete description), who found a low (computational) cost method which can be used for MVN probabilities. Connors et al. (2014) found the ME method had overall superior performance compared to other approximation methods (including the method developed by Joe 1995), so we focus on improvements to the ME approximations. The generalizations we study here use bivariate conditioning, with the goal of providing more accurate approximations without significantly increasing the computational cost, compared to a univariate conditioning method. In order to provide background for the new approximation methods, we first describe the basic univariate conditioning method.

G. Trinh · A. Genz (✉)
Department of Mathematics, Washington State University, Pullman, WA 99164-3113, USA
e-mail: alangenz@wsu.edu

Published online: 08 April 2014

Then we derive our bivariate conditioning methods, and finish with some tests comparing the different methods.

2 Univariate conditioning approximations

2.1 The basic univariate algorithm

We start with the Cholesky decomposition of $\Sigma = CC^t$, where C a lower triangular matrix. Then $\mathbf{x}^t\Sigma^{-1}\mathbf{x} = \mathbf{x}^t C^{-t}C^{-1}\mathbf{x}$, and if we use the transformation $\mathbf{x} = C\mathbf{y}$, we have $\mathbf{x}^t\Sigma^{-1}\mathbf{x} = \mathbf{y}^t\mathbf{y}$ with $d\mathbf{x} = |C|\,d\mathbf{y} = \sqrt{|\Sigma|}d\mathbf{y}$. The probability region for $\Phi(\mathbf{a}, \mathbf{b}; \Sigma)$ is now given by $\mathbf{a} \le C\mathbf{y} \le \mathbf{b}$. Taking advantage of the lower triangular structure of C, this set of inequalities can be rewritten in more detail in the form

$$
\begin{aligned}
a_1/c_{11} &\le y_1 \le b_1/c_{11} \\
(a_2 - c_{21}y_1)/c_{22} &\le y_2 \le (b_2 - c_{21}y_1)/c_{22} \\
&\vdots \\
\left(a_n - \sum_{m=1}^{n-1} c_{nm}y_m\right)/c_{nn} &\le y_n \le \left(b_n - \sum_{m=1}^{n-1} c_{nm}y_m\right)/c_{nn}.
\end{aligned}
$$

Then, using

$$
\begin{aligned}
a_i'(y_1, \ldots, y_{i-1}) &= \left(a_i - \sum_{m=1}^{i-1} c_{im}y_m\right)/c_{ii}, \\
b_i'(y_1, \ldots, y_{i-1}) &= \left(b_i - \sum_{m=1}^{i-1} c_{im}y_m\right)/c_{ii},
\end{aligned}
$$

we have

$$
\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) = \frac{1}{\sqrt{(2\pi)^n}} \int_{a_1'}^{b_1'} e^{-\frac{y_1^2}{2}} \cdots \int_{a_{n-1}'(y_1,\ldots,y_{n-1})}^{b_{n-1}'(y_1,y_{n-1})} e^{-\frac{y_n^2}{2}} d\mathbf{y}. \tag{1}
$$

This "conditioned" form for MVN probabilities has been used as the basis for several numerical approximation and simulation methods (see Genz and Bretz 2009), and the ME approximation method also uses this form.

A key feature of the ME method is the replacement of the y_i values in the limits for the integrals in Eq. (1) by truncated expected values. The motivation for these replacements is that these values are the average values that the y_i's would have if we simulated y_i's with values taken from the truncated univariate distributions. We will use standard notation for the univariate normal pdf $\phi(t) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}t^2}$, and cdf $\Phi(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-\frac{1}{2}t^2}dt$. We define a one-dimensional truncated expected value by

$$
E(a, b) = \frac{\int_a^b s e^{-\frac{s^2}{2}} ds}{\int_a^b e^{-\frac{s^2}{2}} ds} = \frac{\phi(a) - \phi(b)}{\Phi(b) - \Phi(a)},
$$

Then, define $\hat{a}_i$ and $\hat{b}_i$ to be the respective values of $a_i'(y_1, \ldots, y_{i-1})$, and $b_i'(y_1, \ldots, y_{i-1})$, where the y_j values are replaced by μ_j values for $j = 1, ..., i-1$, successively computed from the expected values $\mu_j = E(\hat{a}_j, \hat{b}_j)$. An approximation for the MVN problem can then be given by

$$
\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) \approx \hat{\Phi}_n(\mathbf{a}, \mathbf{b}; \Sigma) = \prod_{i=1}^{n} (\Phi(\hat{b}_i) - \Phi(\hat{a}_i)). \tag{2}
$$

This computation is detailed in the following algorithm.

Algorithm 2.1

1. **Input** $\mathbf{a}$, $\mathbf{b}$, Σ.
2. **Initialization:** compute Cholesky factor C for Σ and set $P = 1$.
3. **For** $j = 1, \ldots, n$,
   if $j > 1$, set $\mu_{j-1} = (\phi(\hat{a}_{j-1}) - \phi(\hat{b}_{j-1}))/U$;
   set $g_j = \sum_{m=1}^{j-1} c_{jm}\mu_m$, $(\hat{a}, \hat{b})_j = (a-g, b-g)_j/c_{jj}$,
   set $U = \Phi(\hat{b}_j) - \Phi(\hat{a}_j)$, and $P = PU$
   End (j loop).
4. **Output** $P \approx \Phi_n(\mathbf{a}, \mathbf{b}; \Sigma)$.

2.2 Variable reordering for improved accuracy

Algorithm 2.1 uses an ordering of the variables that is specified by the input Σ, but there are $n!$ possible orderings of the variables for $\Phi(\mathbf{a}, \mathbf{b}, \Sigma)$. These orderings do not change the MVN value as long as the integration limits and corresponding rows and columns of Σ are also permuted. Schervish (1984) originally proposed sorting the variables so that the variables with the shortest integration interval widths were the outer integration variables. This approach is expected to reduce the overall variation of the integrand and consequently to result in an easier numerical integration problem. Gibson et al. (GGE 1994) suggested an improved prioritization of the variables. They proposed sorting the variables so that the outermost integral variables have the smallest expected values. With this heuristic, the outer variables, which have the most influence on the innermost integrals, tend to have smaller variation, and this often reduces the overall variance for the resulting integration problem. Recent test results (Connors et al. 2014) have shown that this variable prioritized reordering, when combined with Algorithm 2.1, can often produce more accurate results. We will also consider this reordering with bivariate conditioning algorithms, so we provide details for the GGE reordering method here.

The GGE variable prioritization method first chooses the outermost integration variable by selecting the variable i so

that

$$i = \arg\min_{1\le i\le n}\left\{\Phi\left(\frac{b_i}{\sqrt{\sigma_{ii}}}\right) - \Phi\left(\frac{a_i}{\sqrt{\sigma_{ii}}}\right)\right\}.$$

The integration limits and the rows and columns of Σ for variables 1 and i are interchanged. Then the first column of the Cholesky decomposition C of Σ is computed using $c_{11} = \sqrt{\sigma_{11}}$ and $c_{i1} = \frac{\sigma_{i1}}{c_{11}}$ for $i = 2, \ldots, n$. Letting $\hat{a}_1 = \frac{a_1}{c_{11}}$, $\hat{b}_1 = \frac{b_1}{c_{11}}$, we set $\mu_1 = E(\hat{a}_1, \hat{b}_1)$.

Next, i is chosen so that

$$i = \arg\min_{2\le i\le n}\left\{\Phi\left(\frac{b_i - c_{i1}\mu_1}{\sqrt{\sigma_{ii} - c_{i1}^2}}\right) - \Phi\left(\frac{a_i - c_{i1}\mu_1}{\sqrt{\sigma_{ii} - c_{i1}^2}}\right)\right\}.$$

The integration limits, rows and columns of Σ, and c_{12} and c_{i2} are interchanged. Then the second column of the Cholesky decomposition C of Σ is computed using $c_{22} = \sqrt{\sigma_{22} - c_{21}^2}$ and $c_{i2} = \frac{\sigma_{i2} - c_{21}c_{i1}}{c_{22}}$ for $i = 3, \ldots, n$. Let $\hat{a}_2 = \frac{a_2 - c_{21}\mu_1}{c_{22}}$, $\hat{b}_2 = \frac{b_2 - c_{21}\mu_1}{c_{22}}$, and set $\mu_2 = E(\hat{a}_2, \hat{b}_2)$.

At stage j, the j-th integration variable is chosen by selecting a variable i so that

$$i = \arg\min_{j\le i\le n}\left\{\Phi\left(\frac{b_i - \Sigma_{m=1}^{j-1} c_{im}\mu_m}{\sqrt{\sigma_{ii} - \Sigma_{m=1}^{j-1} c_{im}^2}}\right) - \Phi\left(\frac{a_i - \Sigma_{m=1}^{j-1} c_{im}\mu_m}{\sqrt{\sigma_{ii} - \Sigma_{m=1}^{j-1} c_{im}^2}}\right)\right\}.$$

The integration limits, rows and columns of Σ, and partially completed rows of C for variables j and i are interchanged. Then the jth column of C is computed using $c_{jj} = \sqrt{\sigma_{ii} - \Sigma_{m=1}^{j-1} c_{im}^2}$ and $c_{ij} = (\sigma_{ij} - \Sigma_{m=1}^{j-1} c_{im}c_{jm})/c_{jj}$, for $i = j+1, \ldots, n$. Letting $\hat{a}_j = (a_j - \Sigma_{m=1}^{j-1} c_{jm}\mu_m)/c_{jj}$, and $\hat{b}_j = (b_j - \Sigma_{m=1}^{j-1} c_{jm}\mu_m)/c_{jj}$, we set $\mu_j = E(\hat{a}_j, \hat{b}_j)$. The algorithm finishes when $j = n$, with $\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) \approx \hat{\Phi}_n(\mathbf{a}, \mathbf{b}; \Sigma)$ given by (2).

Algorithm 2.2

1. **Input** $\mathbf{a}$, $\mathbf{b}$, Σ.
2. **Initialization**: set $P = 1$.
3. **For** $j = 1, \ldots, n$,
   if $j > 1$, set $\mu_{j-1} = [\phi(\hat{a}_{j-1}) - \phi(\hat{b}_{j-1})]/U$;
   set $i = \arg\min_{j\le i\le n}$
   $\left\{\Phi\left(\frac{b_i - \Sigma_{m=1}^{j-1} c_{im}\mu_m}{\sqrt{\sigma_{ii} - \Sigma_{m=1}^{j-1} c_{im}^2}}\right) - \Phi\left(\frac{a_i - \Sigma_{m=1}^{j-1} c_{im}\mu_m}{\sqrt{\sigma_{ii} - \Sigma_{m=1}^{j-1} c_{im}^2}}\right)\right\}$;
   set $\Sigma = \Sigma\{j \rightleftharpoons i\}$, $\mathbf{a} = \mathbf{a}\{j \rightleftharpoons i\}$, $\mathbf{b} = \mathbf{b}\{j \rightleftharpoons i\}$,
   $C = C\{j \rightleftharpoons i\}$, $c_{jj} = \sqrt{\sigma_{jj} - \Sigma_{m=1}^{j-1} c_{jm}^2}$,
   and set $c_{ij} = (\sigma_{ij} - \Sigma_{m=1}^{j-1} c_{im}c_{jm})/c_{jj}$,
   for $i = j+1, \ldots, n$;
   set $\hat{a}_j = (a_j - \sum_{m=1}^{j-1} c_{jm}\mu_m)/c_{jj}$,
   $\hat{b}_j = (b_j - \sum_{m=1}^{j-1} c_{jm}\mu_m)/c_{jj}$,
   $U = \Phi(\hat{b}_J) - \Phi(\hat{a}_j)$, and $P = PU$
   End (j loop).
4. **Output** $P \approx \Phi_n(\mathbf{a}, \mathbf{b}; \Sigma)$.

Note: we have used the notation $\Sigma = \Sigma\{i \rightleftharpoons j\}$ to indicate interchanges of rows and columns i and j for Σ; $\mathbf{a} = \mathbf{a}\{i \rightleftharpoons j\}$ indicates interchanging a_i and a_j in $\mathbf{a}$; $C = C\{i \rightleftharpoons j\}$ indicates interchanging partially completed rows i and j in C.

3 Bivariate conditioning approximations

We will derive several algorithms which use a bivariate conditioned form for $\Phi(\mathbf{a}, \mathbf{b}; \Sigma)$. These algorithms all depend on methods for fast and accurate bivariate normal (BVN) computations which are now available (see Drezner and Wesolowsky 1990; Genz 2004). The algorithms also depend on a bivariate decomposition for Σ, which we now describe.

3.1 LDL^t decomposition

In order to transform the MVN problem into an approximate sequence of BVN integrals, we define $k = \lfloor \frac{n}{2} \rfloor$ and use the covariance matrix decomposition $\Sigma = LDL^t$. If n is even

$$L = \begin{bmatrix} I_2 & O_2 & \cdots & O_2 \\ L_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & I_2 & O_2 \\ L_{k1} & \ldots & L_{k,k-1} & I_2 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & O_2 & \cdots & O_2 \\ O_2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & D_{k-1} & O_2 \\ O_2 & \ldots & O_2 & D_k \end{bmatrix},$$

where D_i, $L_{i,j}$, are 2×2 matrices for $i, j \in \{1, \ldots, k\}$, the D_i's are positive definite symmetric, and the O_2's are 2×2 zero matrices. If n is odd, there is an extra row in L, and final entry d_{nn} in D. For example, with

$$\Sigma = \begin{bmatrix} 2 & 1 & -1 & 1 & -2 \\ 1 & 2 & 1 & -1 & 2 \\ -1 & 1 & 4 & -3 & 1 \\ 1 & -1 & -3 & 4 & -1 \\ -2 & 2 & 1 & -1 & 16 \end{bmatrix},$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 \\ -2 & 2 & -1 & 1 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}. \tag{3}$$

This block decomposition can be recursively computed using the partitioning $\Sigma = \begin{bmatrix} \Sigma_{1,1} & R^t \\ R & \hat{\Sigma} \end{bmatrix}$, with $L = \begin{bmatrix} I_2 & O \\ M & \hat{L} \end{bmatrix}$, and $D = \begin{bmatrix} D_1 & O \\ O & \hat{D} \end{bmatrix}$, where $\Sigma_{1,1}$ is a 2×2 matrix. Then $D_1 = \Sigma_{1,1}$, $M = RD_1^{-1}$, $\hat{D} = \hat{\Sigma} - MD_1M^t$, and the decomposition procedure continues by applying the same operations to the $(n-2) \times (n-2)$ matrix $\hat{\Sigma}$. This is a 2×2

block form for the standard Cholesky decomposition algorithm. The details are given in the following algorithm, using Golub and Van Loan (2013) style array indexing notation.

Algorithm 3.1

1. **Input** Σ
2. **Initialize** $L = I_n$ ($n \times n$ identity),
   and $D = O_n$ ($n \times n$ zero matrix).
3. **For** $i = 1:2:n-2$,
   set $D(i:i+1, i:i+1) = \Sigma(i:i+1, i:i+1)$;
   set $L(i+2:n, i:i+1)$
   $= \Sigma(i+2:n, i:i+1)D(i:i+1, i:i+1)^{-1}$;
   set $\Sigma(i+2:n, i+2:n) = \Sigma(i+2:n, i+2:n)$
   $- L(i+2:n, i:i+1)D(i:i+1, i:i+1)$
   $L(i+2:n, i:i+1)^t$;
   End (i loop)
   set $D(i+2:n, i+2:n) = \Sigma(i+2:n, i+2:n)$.
4. **Output** L, D.

3.2 The basic bivariate conditioning algorithm

We assume that we have computed the LDL^t decomposition for Σ. Then we use the transformation $\mathbf{x} = L\mathbf{y}$, so that $d\mathbf{x} = |L|\, d\mathbf{y} = d\mathbf{y}$. The system of inequalities $\mathbf{a} \le L\mathbf{y} \le \mathbf{b}$ can be explicitly written as

$$a_j - \sum_{m=1}^{j-1} l_{jm} y_m \le y_j \le b_j - \sum_{m=1}^{j-1} l_{jm} y_m.$$

for $j = 1, \ldots, n$, We also define $(\alpha, \beta)_j = (a - g, b - g)_j$, with $g_j = \sum_{m=1}^{j-1} l_{jm} y_m$, and let $\mathbf{y}_{2k} = (y_{2k-1}, y_{2k})^t$. Then

$$\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) = \frac{1}{\sqrt{|D|\,(2\pi)^n}} \int_{\alpha_1}^{\beta_1} \int_{\alpha_2}^{\beta_2} e^{-\frac{1}{2}\mathbf{y}_2^t D_1^{-1} \mathbf{y}_2} \cdots \int_{\alpha_{2k-1}}^{\beta_{2k-1}} \int_{\alpha_{2k}}^{\beta_{2k}} e^{-\frac{1}{2}\mathbf{y}_{2k}^t D_k^{-1} \mathbf{y}_{2k}} \times \begin{cases} d\mathbf{y} & \text{if } n = 2k; \\ \int_{\alpha_n}^{\beta_n} e^{-\frac{1}{2d_{nn}} y_n^2} d\mathbf{y} & \text{if } n = 2k+1. \end{cases}$$

Now let $y_i = \sqrt{d_{ii}}\, z_i$, for $i = 1, \ldots, n$. Then

$$\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) = \frac{1}{2\pi |\Omega_{12}|} \int_{a_1'}^{b_1'} \int_{a_2'}^{b_2'} e^{-\frac{1}{2}\mathbf{z}_2^t \Omega_{12}^{-1} \mathbf{z}_2} \cdots \frac{1}{2\pi |\Omega_{2k-1,2k}|} \int_{a_{2k-1}'}^{b_{2k-1}'} \int_{a_{2k}'}^{b_{2k}'} e^{-\frac{1}{2}\mathbf{z}_{2k}^t \Omega_{2k-1,2k}^{-1} \mathbf{z}_{2k}} \times \begin{cases} d\mathbf{z} & \text{if } n = 2k, \\ \frac{1}{\sqrt{2\pi}} \int_{a_n'}^{b_n'} e^{-\frac{1}{2}z_n^2} d\mathbf{z} & \text{if } n = 2k+1. \end{cases}$$

where $(a', b')_i = (\alpha, \beta)_i / \sqrt{d_{ii}}$, $\Omega_{2k-1,2k} = \begin{bmatrix} 1 & \rho_k \\ \rho_k & 1 \end{bmatrix}$, with $\rho_k = d_{2k-1,2k} / \sqrt{d_{2k-1,2k-1} d_{2k,2k}}$.

The outermost BVN integral is defined by $P_1 = \Phi_2((a_1', a_2'), (b_1', b_2'); \Omega_{12})$, and this can be computed quickly and accurately using available BVN software. In order to approximate the inner integrals, we need values for the z_1 and z_2 variables, which appear in the limits for the inner integrals. Following the motivation for the method described for the univariate algorithms, good overall choices for the z_1 and z_2 variables, should be the expected values for those variables, using the truncated bivariate distribution.

Muthén (1991) derived formulas for the moments (expected values) μ_1 and μ_2 of the corresponding truncated BVN distribution. The Muthén formulas are generalization of formulas for the $a_i = -\infty$ case which were derived earlier by Rosenbaum (1961). If we let $q_1 = \sqrt{1 - \rho_1^2}$, the moments μ_1, μ_2 are defined by

$$(\mu_1, \mu_2) = E((a_1', a_2'), (b_1', b_2'); \rho_1) = \frac{1}{2\pi P_1 q_1} \int_{a_1'}^{b_1'} \int_{a_2'}^{b_2'} (u, v) e^{-\frac{u^2 + v^2 - 2uv\rho_1}{2q_1^2}} dv du.$$

The Muthén formula for μ_1 is determined from

$$\begin{aligned} P_1 \mu_1 = \; & \rho_1 \phi(a_2') \Phi\left(\frac{a_1' - \rho_1 a_2'}{q_1}, \frac{b_1' - \rho_1 a_2'}{q_1} \right) \\ & + \phi(a_1') \Phi\left(\frac{a_2' - \rho_1 a_1'}{q_1}, \frac{b_2' - \rho_1 a_1'}{q_1} \right) \\ & - \rho_1 \phi(b_2') \Phi\left(\frac{a_1' - \rho_1 b_2'}{q_1}, \frac{b_1' - \rho_1 b_2'}{q_1} \right) \\ & - \phi(b_1') \Phi\left(\frac{a_2' - \rho_1 b_1'}{q_1}, \frac{b_2' - \rho_1 b_1'}{q_1} \right), \end{aligned} \tag{4}$$

using the univariate $\Phi(a, b) = \Phi(b) - \Phi(a)$. The μ_2 formula is the same, except for the interchanges $a_1' \rightleftharpoons a_2'$ and $b_1' \rightleftharpoons b_2'$. Note that the μ_i formulas depend only on easily computed univariate pdf and cdf values.

Now, we can approximate the second outermost BVN integral by $P_2 = \Phi_2((\hat{a}_3, \hat{a}_4), (\hat{b}_3, \hat{b}_4); \Omega_{3,4})$ where $\hat{a}_i, \hat{b}_i$, are the values of a_i', b_i', with z_1, z_2 replaced by the moment values μ_1, μ_2. Then, with $q_2 = \sqrt{1 - \rho_2^2}$, μ_3 and μ_4 are defined by $(\mu_3, \mu_4) = E((\hat{a}_3, \hat{a}_4), (\hat{b}_3, \hat{b}_4); \rho_2)$ and computed using (4), and these new μ's can be used to replace z_3 and z_4 in the limits for the remaining inner integrals.

At the i-th stage in the algorithm,

$$P_i = \Phi_2((\hat{a}_{2i-1}, \hat{a}_{2i}), (\hat{b}_{2i-1}, \hat{b}_{2i}); \Omega_{2i-1,2i}),$$

where $\hat{a}_i$, $\hat{b}_i$, are the values of a'_i, b'_i, with $z_1, \ldots, z_{2i-2}$ replaced by the expected values $\mu_1, \ldots, \mu_{2i-2}$ (which were successively computed in pairs using $\hat{a}$ and $\hat{b}$ values). After the final (k-th) stage we have

$$\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma) \approx \prod_{i=1}^{k} P_i \times \begin{cases} 1 & \text{if } n = 2k; \\ (\Phi(\hat{b}_n) - \Phi(\hat{a}_n)) & \text{if } n = 2k+1. \end{cases} \quad (5)$$

In the following algorithm, to simplify the calculations, we use scaled moments e_i defined by $e_i = \mu_i \sqrt{d_{ii}}$.

Algorithm 3.2

1. **Input** $\mathbf{a}$, $\mathbf{b}$, Σ.
2. **Initialization:** set $P = 1$ and
   compute LDL^t for Σ with Algorithm 3.1,
3. **For** $i = 2 : 2 : n$
   set $g_j = \sum_{m=1}^{i-2} l_{jm} e_m$,
   $(\hat{a}, \hat{b})_j = (a - g, b - g)_j / \sqrt{d_{jj}}$, for $j = i-1, i$,
   and set $P = P\Phi_2((\hat{a}_{i-1}, \hat{a}_i), (\hat{b}_{i-1}, \hat{b}_i); \Omega_{i-1,i})$;
   If $i < n$, compute (e_{i-1}, e_i) (with Muthén (4),scaled);
   End (i loop).
   If n is odd, set $g_n = \sum_{m=1}^{n-1} l_{nm} e_m$,
   and $P = P\Phi((a - g, b - g)_n / \sqrt{d_{nn}})$.
4. **Output** $P \approx \Phi_n(\mathbf{a}, \mathbf{b}; \Sigma)$.

3.3 BVN approximation variable reordering

In this section we describe a generalization of the one-dimensional GGE variable prioritization algorithm. For this bivariate algorithm, we successively choose the integration variables to minimize the outermost BVN values. The first two variables are chosen by selecting i and j so that

$$\{i, j\} = \arg \min_{1 \le i < j \le n} \left\{ \Phi_2 \left(\left(\frac{a_i}{\sqrt{\sigma_{ii}}}, \frac{a_j}{\sqrt{\sigma_{jj}}}, \right), \left(\frac{b_i}{\sqrt{\sigma_{ii}}}, \frac{b_j}{\sqrt{\sigma_{jj}}}, \right); \Omega_{ij} \right) \right\},$$

where we are using a more general definition of $\Omega_{ij} = \begin{bmatrix} 1 & r_{ij} \\ r_{ij} & 1 \end{bmatrix}$, with $r_{ij} = \sigma_{ij} / \sqrt{\sigma_{ii}\sigma_{jj}}$. The integration limits, rows and columns of Σ for variables $\{1, 2\}$ and $\{i, j\}$ are interchanged. Columns 1 and 2 of L and the first 2×2 diagonal block of D are computed, Σ is updated using Algorithm 3.1, and the scaled expected values e_1, e_2 are computed from μ_1, μ_2, using (4).

At stage m, the m^{th} set of two variables is chosen by selecting i and j so that

$$\{i, j\} = \arg \min_{2m-1 \le i < j \le n} \left\{ \Phi_2 \left(\left(\frac{a_i - g_i}{\sqrt{\sigma_{ii}}}, \frac{a_j - g_j}{\sqrt{\sigma_{jj}}}, \right), \left(\frac{b_i - g_i}{\sqrt{\sigma_{ii}}}, \frac{b_j - g_j}{\sqrt{\sigma_{jj}}}, \right); \Omega_{ij} \right) \right\}.$$

The integration limits, and rows of L, rows and columns of Σ for variables $\{2m-1, 2m\}$ and $\{i, j\}$ are interchanged. The lower block of Σ is updated, and the new columns of L and D_m are computed, along with the scaled expected values e_{2k-1}, e_{2k}. After the final (k^{th}) stage, $\Phi_n(\mathbf{a}, \mathbf{b}; \Sigma)$ is approximated using (5).

Algorithm 3.3

1. **Input** $\mathbf{a}$, $\mathbf{b}$, Σ.
2. **Initialization:** set $P = 1$, $\mathbf{g} = \mathbf{0}$, $L = O_n$.
3. **For** $s = 2 : 2 : n$
   set $\{i, j\} = \arg \min_{s-1 \le i < j \le n}$
   $\left\{ \Phi_2 \left(\left(\frac{a_i - g_i}{\sqrt{\sigma_{ii}}}, \frac{a_j - g_j}{\sqrt{\sigma_{jj}}}, \right), \left(\frac{b_i - g_i}{\sqrt{\sigma_{ii}}}, \frac{b_j = g_j}{\sqrt{\sigma_{jj}}}, \right); \Omega_{ij} \right) \right\}$;
   set $\Sigma = \Sigma\{s-1 \rightleftharpoons i, s \rightleftharpoons j\}$,
   $L = L\{s-1 \rightleftharpoons i, s \rightleftharpoons j\}$,
   $\mathbf{a} = \mathbf{a}\{s-1 \rightleftharpoons i, s \rightleftharpoons j\}$,
   $\mathbf{b} = \mathbf{b}\{s-1 \rightleftharpoons i, s \rightleftharpoons j\}$,
   $\mathbf{g} = \mathbf{g}\{s-1 \rightleftharpoons i, s \rightleftharpoons j\}$;
   compute $L(s+1 : n, s-1 : s)$
   and update $\Sigma(s+1 : n, s+1 : n)$;
   set $(\hat{a}, \hat{b})_j = (a-g, b-g)_j / \sqrt{d_{jj}}$, for $j = s-1, s$,
   and $P = P\Phi_2((\hat{a}_{s-1}, \hat{a}_s), (\hat{b}_{s-1}, \hat{b}_s); \Omega_{s-1,s})$;
   If $s < n$, compute (e_{s-1}, e_s), and
   $g_j = g_j + \sum_{m=s-1}^{s} l_{jm} e_m$, for $j = s+1, \ldots, n$;
   End (s loop).
   If n is odd, set $P = P\Phi((a-g, b-g)_n / \sqrt{\sigma_{nn}})$.
4. **Output** $P \approx \Phi_n(\mathbf{a}, \mathbf{b}; \Sigma)$.

Note: the time cost for this algorithm is much larger than Algorithm 3.2 because of the $O(n^3)$ BVN values that are needed in step 3 to determine optimal $\{i, j\}$ pairs.

3.4 BVN univariate variable reordering

In order to reduce the time complexity of Algorithm 3.3 the variables can be sorted using Algorithm 2.2 before computing the BVN approximation with Algorithm 3.2. Preliminary tests results showed that this approach improved the accuracy of numerical results, compared to Algorithm 3.2, with a significant reduction in computation time compared to Algorithm 3.3, so we also provide details for this modified algorithm.

If Algorithm 2.2 is used, the resulting C matrix satisfies $U\Sigma U^t = CC^t$ for some permutation matrix U, so we can determine L and D for the modified algorithm using the relation $LDL^t = CC^t$. If C_k, for $k = 1, \ldots, \lfloor \frac{n}{2} \rfloor$, is used to denote the diagonal 2×2 blocks of C, with $C_{k+1} = c_{nn}$ when n is odd, and C_D is used to denote the block diagonal matrix with the C_k's as diagonal entries, then $D = C_D C_D^t$ and $L = CC_D^{-1}$. The $\mathbf{a}$ and $\mathbf{b}$ vectors needed for the modified algorithm are the permuted $\mathbf{a}$ and $\mathbf{b}$ vectors available at the

end of Algorithm 2.2. This modified algorithm is given as Algorithm 3.4.

Algorithm 3.4

1. **Input** $\mathbf{a}$, $\mathbf{b}$, Σ.
2. **Initialization:** use Algorithm 2.2 to
   compute C and permuted $\mathbf{a}$ and $\mathbf{b}$;
   compute L and D from C and set $P = 1$.
3. **Step 3** from Algorithm 3.2
4. **Output** $P \approx \Phi_n(\mathbf{a}, \mathbf{b}; \Sigma)$.

Using the example Σ from Sect. 3.1 (Eq. 3) we computed $\Phi_5(-(4, 4, 4, 4, 4)^t, (2, 4, 2, 7, 1)^t; \Sigma)$ using several approximations, with results given in Table 1. This example illustrates typical results for the algorithms, with much more accurate results from the variable prioritized algorithms.

We also completed another set of tests with this example to investigate the optimality of the variable prioritization heuristics we have used for Algorithms 3.3 and 3.4. We computed the Φ_5 approximations using Algorithm 2.2 with all of the 120 permutations of the integration variables. The best permutation gave us $\Phi_5 \approx 0.33443$ with error 0.00473, the same results as with Algorithm 3.3, but not significantly better than Algorithm 3.4.

We also checked the method developed by Joe (1995). Although this was found not to be as accurate as the ME method in the tests by Connors et al. (2014), this method uses some BVN values. The method uses a BVN value for the first two variables, and this BVN value is multiplied by a product of $n-2$ UVN values, corrected using additional information from the $\frac{(n+1)n}{2}$ BVN values computed using all possible pairs of variables. The complete Joe method uses averages, of these MVN approximations, over all of the $n!$ permutations of the variables. This method produced a more accurate Φ_5 value of 0.33008 (with error ≈ 0.0004), but this method takes $O(n!)$ time ($\approx 16\times$ more time than Algorithm 3.4 for Φ_5).

4 Randomized tests

We completed a series of tests to compare MATLAB implementations of the five algorithms discussed in this paper. For each $n = 3, \ldots, 20$, we generated 250 random $(\mathbf{b}, \Sigma)$ combinations. Each $\Sigma = QDQ^t$ was determined from a randomly generated $n \times n$ orthogonal matrix Q (see Stewart 1980) and a diagonal matrix with diagonal entries $d_i = u_i$, and each $\mathbf{b}$ vector had $b_i = nv_i$, with u_i, v_i uniform random from [0, 1]. We used $a_i = -\infty$ for all i for all tests. The results are given in Tables 2 and 3. Our choice for the random $\mathbf{b}$'s produced average MVN probabilities in the .7−.8 range. We used the Matlab **mvncdf** function to determine "exact" values for testing purposes.

Table 1 MVN approximations, with errors (below), for (3)

Approximation algorithms values (errors)					Exact(5d)
2.1	3.2	2.2	3.3	3.4	
0.51149	0.50806	0.33489	0.33443	0.33467	0.32970
(0.18179)	(0.17836)	(0.00519)	(0.00473)	(0.00497)	

Table 2 Average errors for MVN approximation algorithms

n	Algorithm average absolute errors, $\delta = 1$				
	2.1	3.2	2.2	3.3	3.4
3	0.01017	0.00629	0.00244	0.00063	0.00068
4	0.01528	0.00913	0.00197	0.00068	0.00069
5	0.01457	0.01215	0.00193	0.00071	0.00071
6	0.01940	0.01607	0.00135	0.00053	0.00053
7	0.01983	0.01701	0.00154	0.00075	0.00075
8	0.01716	0.01434	0.00101	0.00042	0.00044
9	0.01994	0.01821	0.00108	0.00045	0.00047
10	0.02290	0.01940	0.00135	0.00074	0.00076
11	0.02505	0.02281	0.00097	0.00049	0.00049
12	0.01846	0.01673	0.00090	0.00037	0.00037
13	0.02398	0.02178	0.00104	0.00056	0.00056
14	0.02086	0.01859	0.00086	0.00043	0.00043
15	0.02399	0.02231	0.00081	0.00043	0.00044
16	0.02301	0.02104	0.00086	0.00047	0.00047
17	0.02902	0.02736	0.00101	0.00044	0.00045
18	0.02406	0.02224	0.00070	0.00034	0.00034
19	0.02287	0.02133	0.00052	0.00027	0.00027
20	0.02547	0.02361	0.00081	0.00040	0.00040

Table 2 shows average errors for the different algorithms. he sorted variable algorithms (2.2, 3.3, 3.4) are much more accurate, compared to the unsorted variable algorithms, often by factors of more than 10. The sorted variable bivariate algorithms were more accurate then the sorted variable univariate algorithm with typical error reductions in the 50–75 % range.. We also computed standard errors for the averages and these were always at least a factor of 1/10 smaller than the displayed averages so we are confident that the averages displayed in the Table are representative of averages for larger samples sizes.

Table 3 shows average times in seconds for the different algorithms, using computations on a workstation with a 3.5GHz processor. The time differences for the algorithms which use univariate sorting (2.2, 3.4), with $O(n^2)$ time) compared to the unsorted (2.2, 3.4 $O(n)$ time) algorithms increase slowly with n, as is expected. The computation time increases are much more significant for Algorithm 3.3, which

Table 3 Ave. times for MVN approximation algorithms

n	Algorithm average times (in seconds)					
	2.1	3.2	2.2	3.3	3.4	mvncdf
3	0.0003	0.0005	0.0004	0.0007	0.0006	0.0065
4	0.0003	0.0007	0.0006	0.0013	0.0009	0.0216
5	0.0004	0.0008	0.0008	0.0023	0.0012	0.0278
6	0.0005	0.0011	0.0011	0.0038	0.0016	0.0387
7	0.0007	0.0012	0.0015	0.0057	0.0020	0.0412
8	0.0008	0.0015	0.0019	0.0083	0.0025	0.0479
9	0.0009	0.0017	0.0023	0.0116	0.0029	0.0609
10	0.0011	0.0020	0.0028	0.0158	0.0036	0.0582
11	0.0013	0.0022	0.0033	0.0207	0.0041	0.0658
12	0.0014	0.0025	0.0039	0.0267	0.0048	0.0843
13	0.0016	0.0027	0.0045	0.0336	0.0053	0.0763
14	0.0018	0.0030	0.0052	0.0417	0.0061	0.0821
15	0.0020	0.0033	0.0059	0.0511	0.0068	0.0923
16	0.0022	0.0037	0.0066	0.0618	0.0077	0.0964
17	0.0025	0.0039	0.0075	0.0739	0.0085	0.1020
18	0.0027	0.0043	0.0083	0.0876	0.0094	0.1291
19	0.0030	0.0046	0.0092	0.1026	0.0103	0.1195
20	0.0032	0.0050	0.0102	0.1195	0.0113	0.1274

uses $O(n^3)$ bivariate sorting, and the relatively small increase in accuracy does not justify the use of this algorithm compared to Algorithm 3.4. The times given for **mvncdf** were average times with the absolute accuracy tolerance level set at 10^{-3}. At this accuracy level, the **mvncdf** function typically requires more than 10–20 times more computation time when compared to Algorithm 3.4.

The algorithms described here will all be exact (to within whatever the UVN computer accuracy is) for problems where the input covariance matrix is diagonal, and the bivariate conditioned algorithms will be exact if the input covariance matrix is 2×2 block diagonal. But, as the dependence between the the variables increases, we expect the algorithms to become less accurate, so we also completed some (more limited) tests to check the accuracy of the algorithms for problems where there is stronger dependence between the variables. For these tests, we used $\Sigma = QRQ^t$, determined from a randomly generated $n \times n$ orthogonal matrix Q, and $R = I_n + \rho(L_n + L_n^t)$, with L_n a strictly (with zeros on the diagonal) lower triangular matrix with uniform $[0, 1]$ random entries, and $0 < \rho < 1$. Increasing ρ increases the overall dependence between the variables. As in the previous tests, the $\mathbf{b}$ vectors had $b_i = nv_i$, with v_i uniform random from $[0, 1]$, and we used $a_i = -\infty$ for all i for all tests.

Tables 4, 5 and 6 show results with $\rho = .1$, $\rho = 0.7$ and $\rho = 0.9$ (with average "exact" values ≈ 0.7 for all cases). For these tables, the average errors for each n were also computed

Table 4 Ave. errors for $\rho = 0.1$ dependence problems

n	Algorithm average absolute errors $\rho = 0.1$				
	2.1	3.2	2.2	3.3	3.4
6	0.00189	0.00154	0.00045	0.00027	0.00027
7	0.00236	0.00200	0.00047	0.00026	0.00027
8	0.00275	0.00246	0.00052	0.00031	0.00031
9	0.00264	0.00227	0.00047	0.00026	0.00026
10	0.00345	0.00304	0.00056	0.00031	0.00031

Table 5 Ave. errors for $\rho = 0.7$ dependence problems

n	Algorithm average absolute errors $\rho = 0.7$				
	2.1	3.2	2.2	3.3	3.4
6	0.05688	0.04410	0.01512	0.00498	0.00548
7	0.06483	0.05546	0.01299	0.00369	0.00423
8	0.06844	0.05888	0.01584	0.00521	0.00629
9	0.06658	0.05927	0.01446	0.00443	0.00547
10	0.07515	0.06823	0.01535	0.00442	0.00501

Table 6 Ave. errors for $\rho = 0.9$ dependence problems

n	Algorithm average absolute errors, $\rho = 0.9$				
	2.1	3.2	2.2	3.3	3.4
6	0.11624	0.08971	0.02503	0.00368	0.00664
7	0.11374	0.09839	0.01944	0.00317	0.00420
8	0.10807	0.09276	0.01937	0.00216	0.00422
9	0.12177	0.10634	0.02359	0.00380	0.00571
10	0.11970	0.10510	0.02066	0.00358	0.00476

using 250 random Σ's and $\mathbf{b}$'s. As was expected, for the set of problems with weak ($\rho = 0.1$) dependence all of the algorithms were more accurate, but the problems with stronger dependence were more difficult for the algorithms. Overall the reordered variable algorithms are still more accurate compared to the unsorted variable algorithms, and the bivariate conditioned algorithms are still significantly more accurate than the univariate conditioned algorithms, with more significant error reductions as the dependence becomes stronger.

Another, more limited, set of tests was completed to investigate the behavior of the algorithms when the Φ_n values were smaller. For these tests we generated Σ's in the same way as we did for the Table 3 tests, but $b_i = nv_i/25$, with $v_i \sim U[0, 1]$. This choice for the b_i values produced average exact probabilities ≈ 0.15 at $n = 3$ and decreased to ≈ 0.008 at $n = 10$. The results are given in Table 7. Here it appears that the differences between the algorithms diminishes as the Φ_n values decrease. The bivariate algorithms are still a lit-

Table 7 Average errors for small probability problems

n	Algorithm average errors, $b_i \sim U(0, n/25)$				
	2.1	3.2	2.2	3.3	3.4
3	0.00598	0.00453	0.00487	0.00286	0.00337
4	0.00645	0.00427	0.00413	0.00211	0.00239
5	0.00545	0.00453	0.00398	0.00269	0.00278
6	0.00451	0.00323	0.00330	0.00210	0.00237
7	0.00290	0.00268	0.00258	0.00193	0.00201
8	0.00287	0.00229	0.00218	0.00176	0.00176
9	0.00196	0.00171	0.00132	0.00117	0.00115
10	0.00170	0.00152	0.00126	0.00104	0.00108

tle more accurate, but sorting the variables does not make a significant difference.

5 Conclusions and discussion

The kind of approximations described in this paper, which use bivariate conditioning are more accurate, than the univariate conditioning approximations. Variable reorderings using UVN values can significantly improve the accuracy of the approximations, without a significant increase in computational cost. However, bivariate variable reordering prioritized by considering optimal pairs of variables at each stage, which has an $O(n^3)$ time cost (vs. $O(n^2)$ for univariate reordering) does not significantly reduce absolute errors, compared to univariate prioritized reordering, for the bivariate approximation methods. Typical error reductions for the variable prioritized bivariate approximation methods compared to the variable prioritized univariate approximation method were 50–75 %, including problems with higher dependence between the variables.

A significant problem with the type of approximation methods discussed in this article is that, given a specific MVN problem, a particular method can quickly compute an approximation, but without an error estimate. Potential users of these methods will often have some idea about the type of problems where the approximations are needed. The kind of tests that have been used here could be used to assess the expected accuracy level of the approximations for their problems, and to help decide whether to use the approximations instead of much more computationally intensive simulation methods which include error estimates. Software for a MATLAB (also Octave) implementation of Algorithm 3.4 is available from the corresponding author's website (Genz 2013).

The authors of this paper are in the process of extending the work in this article to develop simulation methods which use bivariate conditioning, but this investigation is still not completed. It is also possible to use conditioning approximations similar to the ones described in this paper for the computation of multivariate t probabilities, using appropriate formulas for the truncated moments (see Ho et al. 2012), We would expect similar behaviors for the errors in these approximations, but truncated bivariate t probabilities are more difficult to compute than BVN probabilities, so the time differences between the univariate and bivariate algorithms could be more significant.

References

Connors, R.D., Hess, S., Daly, A.: Analytic approximations for computing probit choice probabilities. Transportmetrica **10**(2), 119–139 (2014)

Drezner, Z., Wesolowsky, G.O.: On the computation of the bivariate normal integral. J. Stat. Comput. Simul. **3**, 101–107 (1990)

Genz, A.: Numerical computation of rectangular bivariate and trivariate normal and t probabilities. Stat. Comput. **14**, 151–160 (2004)

Genz, A.: MVNXPB, a MATLAB/Octave function for the approximation of multivariate Normal probabilities. (2013), at www.math.wsu.edu/faculty/genz/software/matlab

Genz, A., Bretz, F.: Computation of Multivariate Normal and t Probabilities. Lecture Notes in Statistics 195, Springer, New York (2009)

Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore (2013)

Gibson, G.J., Glasbey, C.A., Elston, D.A.: Monte Carlo evaluation of multivariate normal integrals and sensitivity to variate ordering. In: Dimov, I.T., Sendov, B., Vassilevski, P.S. (eds.) Advances in Numerical Methods and Applications, pp. 120–126. World Scientific Publishing, River Edge (1994)

Ho, H.J., Lin, T.I., Chen, H.Y., Wang, W.L.: Some results on the truncated multivariate t distribution. J. Stat. Plan. Inference **142**, 25–40 (2012)

Joe, H.: Approximations to multivariate normal rectangle probabilities based on conditional expectations. J. Am. Stat. Assoc. **90**, 957–964 (1995)

Kamakura, W.A.: The estimation of multinomial probit models: a new calibration algorithm. Transp. Sci. **23**, 253–265 (1989)

McFadden, D., Train, K.: Mixed MNL models for discrete response. J. Appl. Econom. **15**, 447–470 (2000)

Mendell, N.R., Elston, R.C.: Multifactorial qualitative traits: genetic analysis and prediction of recurrence risks. Biometrics **30**, 41–57 (1974)

Muthén, B.: Moments of the censored and truncated bivariate normal distribution. Br J Math Stat Psychol **43**, 131–143 (1991)

Ochi, Y., Prentice, R.L.: Likelihood inference in a correlated probit regression model. Biometrika **71**, 531–543 (1984)

Rosenbaum, S.: Moments of a truncated bivariate normal distribution. J. R. Stat. Soc. **23**, 405–408 (1961)

Schervish, M.J.: Algorithm AS 195: multivariate normal probabilities with error bound. J. R. Stat. Soc. Ser. C **33**, 81–94 (1984), correction **34**, 103–104 (1985)

Stewart, G.W.: The efficient generation of random orthogonal matrices with an application to condition estimators. SIAM J. Numer. Anal. **17**(3), 403–409 (1980)