Journal of Statistical Software

March 2013, Volume 52, Issue 10. *http://www.jstatsoft.org/*

Recursive Numerical Evaluation of the Cumulative Bivariate Normal Distribution

Christian Meyer
DZ BANK AG

Abstract

We propose an algorithm for evaluation of the cumulative bivariate normal distribution, building upon Marsaglia's ideas for evaluation of the cumulative univariate normal distribution. The algorithm delivers competitive performance and can easily be extended to arbitrary precision.

Keywords: cumulative bivariate normal distribution, numerical evaluation, high precision.

1. Introduction

The cumulative normal distribution, be it univariate or multivariate, has to be evaluated numerically. There are numerous algorithms available, many of these having been fine-tuned, leading to faster evaluation and higher precision but also to lack of flexibility (e.g., it may be difficult to extend a specific algorithm from double precision to quad-double precision, maybe due to constants derived from some optimization).

For the univariate case, Marsaglia (2004) has proposed a very simple and intuitive but powerful alternative that is based on Taylor expansion of Mills' ratio or similar functions. In this note we will extend Marsaglia's approach to the bivariate case. This will require two steps: reduction of the evaluation of the cumulative bivariate normal distribution to evaluation(s) of a univariate function, i.e., to the cumulative bivariate normal distribution on the diagonal, and Taylor expansion of that function. Note that a similar approach, but with reduction to the axes instead of the diagonals, has been proposed by Vogt (2008).

The resulting algorithm has to be compared with existing approaches. For overview on and discussion of the latter, see Genz and Bretz (2009), Ağca and Chance (2003), Terza and Welland (1991), and Wang and Kennedy (1990). Most implementations today will rely on variants of the approaches of Divgi (1979) or of Drezner and Wesolowsky (1990). Improve-

ments of the latter method have been provided by Genz (2004) and West (2005). The method of Drezner (1978), although less reliable, is also very common, mainly because it is featured in Hull (2008) and other prevalent books.

It will turn out that the algorithm proposed in this paper is able to deliver near double precision (in terms of absolute error) using double arithmetic. Furthermore, implementation of the algorithm using high-precision libraries is straightforward; indeed, a quad-double implementation has been applied for testing purposes. Performance is competitive, and trade-offs between speed and precision may be implemented with little effort.

2. Theory

In this section we are going to develop the algorithm. In order to keep the presentation lean we will often refer to the author's recent survey (Meyer 2013). For further background on normal distributions the reader is also referred to the text books by Balakrishnan and Lai (2009), Kotz, Balakrishnan, Johnson, and Lloyd (2000), and Patel and Read (1996).

The main idea will be evaluation of the cumulative bivariate normal distribution on the diagonal using Taylor expansion of a function that is analogous to Mills' ratio in the univariate case. This will be discussed in Section 2.1. We will then treat the general case in Section 2.2 by reduction to the diagonal using well-known transformation formulas.

2.1. Evaluation on the diagonal

Denote by

$$\varphi(x) := \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), \qquad \Phi(x) := \int_{-\infty}^{x} \varphi(t)\, dt$$

the density and distribution function of the standard normal distribution. Mills' ratio is then defined as

$$R(x) := \frac{1-\Phi(x)}{\varphi(x)} = \frac{\Phi(-x)}{\varphi(-x)}.$$

Furthermore, denote by

$$\varphi_2(x, y; \varrho) := \frac{1}{2\pi\sqrt{1-\varrho^2}} \exp\left(-\frac{x^2 - 2\varrho xy + y^2}{2(1-\varrho^2)}\right),$$
$$\Phi_2(x, y; \varrho) := \int_{-\infty}^{x}\int_{-\infty}^{y} \varphi_2(s, t; \varrho)\, dt\, ds,$$

the density and distribution function of the bivariate standard normal distribution with correlation parameter $\varrho \in (-1, 1)$. We will also write

$$\varphi_2(x; \varrho) := \varphi_2(x, x; \varrho) = \frac{1}{2\pi\sqrt{1-\varrho^2}} \exp\left(-\frac{x^2}{1+\varrho}\right),$$
$$\Phi_2(x; \varrho) := \Phi_2(x, x; \varrho),$$
$$\lambda(\varrho) := \sqrt{\frac{1-\varrho}{1+\varrho}}.$$

We are going to use the following properties:

$$\frac{d}{dx}\varphi_2(x;\varrho) = -\frac{2x}{1+\varrho}\cdot\varphi_2(x;\varrho),$$
$$\frac{d}{dx}\Phi_2(x;\varrho) = 2\cdot\varphi(x)\cdot\Phi(\lambda(\varrho)\cdot x),$$
$$\varphi_2(x;\varrho)\cdot\sqrt{1-\varrho^2} = \varphi(x)\cdot\varphi(\lambda(\varrho)\cdot x)$$

In the following we will assume that $x \leq 0$, $\varrho \geq 0$. In this case the following bounds apply Meyer (2013, Theorem 5.2):

$$1 + \frac{2}{\pi}\arcsin(\varrho) \leq \frac{\Phi_2(x;\varrho)}{\Phi(x)\cdot\Phi(\lambda(\varrho)\cdot x)} \leq 1+\varrho \tag{1}$$

Furthermore, as is proven implicitly in Meyer (2013, Appendix A.2),

$$\lim_{x\longrightarrow-\infty}\frac{\Phi_2(x;\varrho)}{\Phi(x)\cdot\Phi(\lambda(\varrho)\cdot x)} = 1+\varrho.$$

Now we define

$$D(x) := \frac{(1+\varrho)\cdot\Phi(x)\cdot\Phi(\lambda(\varrho)\cdot x) - \Phi_2(x;\varrho)}{\varphi_2(x;\varrho)}.$$

Starting with

$$D'(x) = (\varrho-1)\cdot\sqrt{1-\varrho^2}\cdot R(-\lambda(\varrho)\cdot x) + (1-\varrho^2)\cdot R(-x) + \frac{2x}{1+\varrho}\cdot D(x)$$

we find the recursion

$$\begin{aligned} D^{(k)}(x) = {} & (\varrho-1)\cdot\sqrt{1-\varrho^2}\cdot(-\lambda(\varrho))^{k-1}\cdot R^{(k-1)}(-\lambda(\varrho)\cdot x) \\ & + (1-\varrho^2)\cdot(-1)^{k-1}\cdot R^{(k-1)}(-x) \\ & + \frac{2(k-1)}{1+\varrho}\cdot D^{(k-2)}(x) + \frac{2x}{1+\varrho}\cdot D^{(k-1)}(x) \end{aligned}$$

which we can use to recursively evaluate the Taylor expansion of D around zero. Dividing by $\sqrt{1-\varrho^2}$ for convenience, we define

$$a_k := \frac{x^{k+1}}{k!}\cdot(\varrho-1)\cdot(-\lambda(\varrho))^k\cdot R^{(k)}(0),$$
$$b_k := \frac{x^{k+1}}{k!}\cdot\sqrt{1-\varrho^2}\cdot(-1)^k\cdot R^{(k)}(0),$$
$$d_k := \frac{x^k}{k!}\cdot\frac{D^{(k)}(0)}{\sqrt{1-\varrho^2}}.$$

Using

$$R^{(k)}(x) = (k-1)\cdot R^{(k-2)}(x) + x\cdot R^{(k-1)}(x)$$

we derive the following recursion scheme:

$$a_k = \frac{\alpha}{k}\cdot a_{k-2}, \tag{2}$$
$$b_k = \frac{\beta}{k}\cdot b_{k-2}, \tag{3}$$
$$d_k = \frac{1}{k}\cdot\left(a_{k-1} + b_{k-1} + \delta\cdot d_{k-2}\right), \tag{4}$$

with initial values

$$a_0 = (\varrho - 1) \cdot \sqrt{\frac{\pi}{2}} \cdot x, \tag{5}$$

$$a_1 = \lambda(\varrho) \cdot (\varrho - 1) \cdot x^2, \tag{6}$$

$$b_0 = \sqrt{1 - \varrho^2} \cdot \sqrt{\frac{\pi}{2}} \cdot x, \tag{7}$$

$$b_1 = \sqrt{1 - \varrho^2} \cdot x^2, \tag{8}$$

$$d_0 = \frac{\varrho \cdot \pi}{2} - \arcsin(\varrho), \tag{9}$$

$$d_{-1} = 0 \tag{10}$$

and coefficients

$$\alpha := x^2 \cdot \frac{1 - \varrho}{1 + \varrho} \geq 0, \qquad \beta := x^2 \geq \alpha \geq 0, \qquad \delta := \frac{2x^2}{1 + \varrho} = \alpha + \beta \geq 0.$$

Here we have used that

$$R(0) = \sqrt{\frac{\pi}{2}}, \qquad \Phi_2(0; \varrho) = \frac{1}{4} + \frac{1}{2\pi} \cdot \arcsin(\varrho).$$

We can now compute $\Phi_2(x; \varrho)$ numerically via

$$\Phi_2(x; \varrho) = (1 + \varrho) \cdot \Phi(x) \cdot \Phi(\lambda(\varrho) \cdot x) - \frac{1}{2\pi} \cdot \exp\left(-\frac{x^2}{1 + \varrho}\right) \cdot \left(\sum_{k=0}^{\infty} d_k\right).$$

Note that it would also have been possible to work with, e.g., one of the functions

$$D_2(x) := \frac{1 - \Phi_2(x; \varrho)}{\varphi_2(x; \varrho)},$$

$$D_3(x) := \frac{\Phi_2(0; \varrho) - \Phi_2(x; \varrho)}{\varphi_2(x; \varrho)},$$

$$D_4(x) := \frac{2 \cdot \Phi(x) \cdot \Phi(\lambda(\varrho) \cdot x) - \Phi_2(x; \varrho)}{\varphi_2(x; \varrho)}$$

instead. The resulting recursion schemes are in fact easier (two summands instead of three) but will be running into numerical problems (cancellation, or lower precision for $x \longrightarrow -\infty$).

2.2. Reduction to the diagonal

In order to apply the results from Section 2.1 to the numerical evaluation of $\Phi_2(x, y; \varrho)$ for general x, y and ϱ, we start with the symmetric formula (Meyer 2013, Equation 3.16)

$$\Phi_2(x, y; \varrho) = \Phi_2(x, 0; \varrho_x) - \delta_x + \Phi_2(0, y; \varrho_y) - \delta_y, \tag{11}$$

where

$$\delta_x = \begin{cases} \frac{1}{2}, & x < 0 \quad \text{and} \quad y \geq 0, \\ 0, & \text{else}, \end{cases}, \qquad \delta_y = \begin{cases} \frac{1}{2}, & y < 0 \quad \text{and} \quad x \geq 0, \\ 0, & \text{else}, \end{cases}$$

and

$$\varrho_x = -\frac{\alpha_x}{\sqrt{1+\alpha_x^2}}, \qquad \alpha_x = \frac{1}{\sqrt{1-\varrho^2}}\left(\frac{y}{x} - \varrho\right),$$
$$\varrho_y = -\frac{\alpha_y}{\sqrt{1+\alpha_y^2}}, \qquad \alpha_y = \frac{1}{\sqrt{1-\varrho^2}}\left(\frac{x}{y} - \varrho\right).$$

From the axis $y = 0$ to the diagonal $x = y$ we get by applying the formula Meyer (2013, Equation 3.18)

$$\Phi_2(x, 0; \varrho) = \begin{cases} \frac{1}{2} \cdot \Phi_2(x, x; 1 - 2\varrho^2), & \varrho < 0, \\ \Phi(x) - \frac{1}{2} \cdot \Phi_2(x, x; 1 - 2\varrho^2), & \varrho \geq 0. \end{cases} \tag{12}$$

Specifically, we obtain

$$1 - 2\varrho_x^2 = 1 - \frac{2 \cdot (\varrho x - y)^2}{x^2 + y^2 - 2\varrho xy} = 1 - \frac{2 \cdot a_x}{1 + a_x} \tag{13}$$

with

$$a_x = \left(\frac{\varrho x - y}{x \cdot \sqrt{1-\varrho^2}}\right)^2 \tag{14}$$
$$= \left(\frac{x - y}{x \cdot \sqrt{1-\varrho^2}} - \sqrt{\frac{1-\varrho}{1+\varrho}}\right)^2 \tag{15}$$
$$= \left(\frac{x + y}{x \cdot \sqrt{1-\varrho^2}} - \sqrt{\frac{1+\varrho}{1-\varrho}}\right)^2 \tag{16}$$

where, in an implementation, Equation 15 should be used for $\varrho \longrightarrow 1$, and Equation 16 for $\varrho \longrightarrow -1$, in order to avoid catastrophic cancellation. Note also that

$$\varrho_x < 0 \quad \Longleftrightarrow \quad \alpha_x > 0 \quad \Longleftrightarrow \quad \frac{y}{x} > \varrho.$$

In a last step, if necessary to ensure $x \leq 0$ and $\varrho \geq 0$, we apply the formulas Meyer (2013, Equations 2.15 and 3.27)

$$\Phi_2(x, x; \varrho) = 2 \cdot \Phi(x) - 1 + \Phi_2(-x, -x; \varrho), \tag{17}$$
$$\Phi_2(x, x; \varrho) = 2 \cdot \Phi(x) \cdot \Phi\left(\lambda(\varrho) \cdot x\right) - \Phi_2\left(\lambda(\varrho) \cdot x, \lambda(\varrho) \cdot x, -\varrho\right). \tag{18}$$

Specifically, we obtain

$$|\lambda(1 - 2\varrho_x^2) \cdot x| = |x| \cdot \sqrt{\frac{\varrho_x^2}{1 - \varrho_x^2}} = \frac{|\varrho x - y|}{\sqrt{1-\varrho^2}} \tag{19}$$
$$= \left|\frac{x - y}{\sqrt{1-\varrho^2}} - x \cdot \sqrt{\frac{1-\varrho}{1+\varrho}}\right| \tag{20}$$
$$= \left|\frac{x + y}{\sqrt{1-\varrho^2}} - x \cdot \sqrt{\frac{1+\varrho}{1-\varrho}}\right|, \tag{21}$$

where, in an implementation, Equation 20 should be used for $\varrho \longrightarrow 1$, and Equation 21 for $\varrho \longrightarrow -1$, in order to avoid catastrophic cancellation.

It will be favorable to work with

$$2\varrho_x^2 = \frac{2 \cdot a_x}{1 + a_x}$$

instead of $1 - 2\varrho_x^2$. If Equation 18 has to be applied (i.e., if $1 - 2\varrho_x^2 < 0$, which is equivalent with $a_x > 1$), correspondingly we will work with

$$1 - (-(1 - 2\varrho_x^2)) = 2 - 2\varrho_x^2 = \frac{2}{1 + a_x}. \tag{22}$$

3. Implementation

In the following we will discuss an implementation of the algorithm derived in Section 2. The C++ language has been chosen because it is the market standard in quantitative finance, one of the fields frequently requiring evaluation of normal distributions.

3.1. Evaluation on the diagonal

Source code (in C++) for evaluation of $\Phi_2(x; \varrho)$ as in Section 2.1, for $x \leq 0$ and $\varrho \geq 0$, will be provided at the end of this section. In the following we will comment on some details of the implementation.

Equations 2–10 show that it is reasonable to provide $\texttt{a} = 1 - \varrho$, instead of ϱ, as input for the evaluation of $\Phi_2(x; \varrho)$. Moreover, cf. Equations 13 and 22, double inversion (i.e., computation of $1 - (1 - z)$ instead of z) is to be avoided in the reduction algorithm.

Values for $\texttt{px} = \Phi(x)$ and for $\texttt{pxs} = \Phi(\lambda(\varrho) \cdot x)$ are also expected as input parameters. This makes sense because the values are needed by the reduction algorithm as well (and hence should not be computed twice).

Evaluation of $\arcsin(\varrho)$ is to be avoided for $\varrho \longrightarrow 1$ and has been replaced (without optimization of the cutoff point) by $\arcsin(\varrho) = \arccos\left(\sqrt{1 - \varrho^2}\right)$.

Constants ($\texttt{c1} = 2/\pi$, $\texttt{c2} = \sqrt{\pi/2}$, $\texttt{c3} = \pi/2$, $\texttt{c4} = 1/(2\pi)$) have been pre-computed in double precision. The recursion stops if a new term does not change the computed sum. If the a priori bound for the absolute error, given by Equation 1, is less than $5 \cdot 10^{-17}$, the upper bound is returned (relative accuracy on the diagonal may be increased by dropping this condition but overall relative accuracy will still be determined by the reduction to the diagonal, cf. Section 3.2), and by the accuracy of the implementation of Φ. The final result is always checked against the upper and lower bound.

In order to avoid cancellation, d_{2k} and d_{2k+1} will be bracketed before summation (cf. Section 4.2).

```
double Phi2diag( double x, double a, double px, double pxs )
{
    if( a <= 0.0 ) return px;
    if( a >= 1.0 ) return px * px;
```

```
    double b = 2.0 - a, sqrt_ab = sqrt( a * b );

    double c1 = 6.36619772367581343e-001;
    double c2 = 1.25331413731550025;
    double c3 = 1.57079632679489662;
    double c4 = 1.59154943091895358e-001;

    double asr = ( a > 0.1 ? asin( 1.0 - a ) : acos( sqrt_ab ) );
    double comp = px * pxs;
    if( comp * ( 1.0 - a - c1 * asr ) < 5e-17 )
        return b * comp;

    double tmp    = c2 * x;
    double alpha  = a * x * x / b;
    double a_even = -tmp * a;
    double a_odd  = -sqrt_ab * alpha;
    double beta   = x * x;
    double b_even = tmp * sqrt_ab;
    double b_odd  = sqrt_ab * beta;
    double delta  = 2.0 * x * x / b;
    double d_even = ( 1.0 - a ) * c3 - asr;
    double d_odd  = tmp * ( sqrt_ab - a );

    double res = 0.0, res_new = d_even + d_odd;
    int k = 2;
    while( res != res_new )
    {
        d_even = ( a_odd + b_odd + delta * d_even ) / k;
        a_even *= alpha / k;
        b_even *= beta / k;
        k++;
        a_odd *= alpha / k;
        b_odd *= beta / k;
        d_odd = ( a_even + b_even + delta * d_odd ) / k;
        k++;
        res = res_new;
        res_new += d_even + d_odd;
    }

    res *= exp( -x * x / b ) * c4;
    return max( ( 1.0 + c1 * asr ) * comp, b * comp - max( 0.0, res ) );
}
```

3.2. Reduction to the diagonal

The following C++ function `Phi2()` evaluates $\Phi_2(x, y; \varrho)$ using Equation 11. It is assumed that `Phi(x)` evaluates $\Phi(x)$. The special cases $|\varrho| = 1$ and $x = y = 0$ are dealt with separately.

```
double Phi2( double x, double y, double rho )
{
    if( ( 1.0 - rho ) * ( 1.0 + rho ) <= 0.0 )
        if( rho > 0.0 )
            return Phi( min( x, y ) );
        else
            return max( 0.0, min( 1.0, Phi( x ) + Phi( y ) - 1.0 ) );

    if( x == 0.0 && y == 0.0 )
        if( rho > 0.0 )
            return Phi2diag( 0.0, 1.0 - rho, 0.5, 0.5 );
        else
            return 0.5 - Phi2diag( 0.0, 1.0 + rho, 0.5, 0.5 );

    return max( 0.0,
           min( 1.0,
           Phi2help( x, y, rho ) + Phi2help( y, x, rho ) ) );
}
```

The auxiliary function `Phi2help()` evaluates $\Phi_2(x, 0; \varrho_x) - \delta_x$. Source code is provided in the following. The special cases $|\varrho| = 1$ and $x = y = 0$ are dealt with in `Phi2()`. Therefore, in `Phi2help()` there is no check against `1.0 - rho == 0.0`, `1.0 + rho == 0.0` or `s == 0.0`. It is assumed that `Phi(x)` evaluates $\Phi(x)$, and that `sqr(x)` evaluates `x*x`. The cutoff points $|\varrho| = 0.99$ in `Phi2help()` have been set by visual inspection and might be optimized.

```
double Phi2help( double x, double y, double rho )
{
    double s = sqrt( ( 1.0 - rho ) * ( 1.0 + rho ) );
    double a = 0.0, b1 = -fabs( x ), b2 = 0.0;

    if( rho > 0.99 )
    {
        double tmp = sqrt( ( 1.0 - rho ) / ( 1.0 + rho ) );
        b2 = -fabs( ( x - y ) / s - x * tmp );
        a = sqr( ( x - y ) / x / s - tmp );
    }
    else if( rho < -0.99 )
    {
        double tmp = sqrt( ( 1.0 + rho ) / ( 1.0 - rho ) );
        b2 = -fabs( ( x + y ) / s - x * tmp );
        a = sqr( ( x + y ) / x / s - tmp );
    }
    else
    {
        b2 = -fabs( rho * x - y ) / s;
        a = sqr( b2 / x );
    }
```

```
    double p1 = Phi( b1 ), p2 = Phi( b2 ), q = 0.0;
    if( a <= 1.0 )
        q = 0.5 * Phi2diag( b1, 2.0 * a / ( 1.0 + a ), p1, p2 );
    else
        q = p1 * p2 - 0.5 * Phi2diag( b2, 2.0 / ( 1.0 + a ), p2, p1 );

    int c1 = ( y / x >= rho ), c2 = ( x < 0.0 ), c3 = c2 && ( y >= 0.0 );
    return ( c1 && c3 ? q - 0.5
                      : c1 && c2 ? q
                      : c1 ? 0.5 - p1 + q
                      : c3 ? p1 - q - 0.5
                      : c2 ? p1 - q
                      : 0.5 - q );
}
```

4. Discussion

We will now discuss convergence of the algorithm in theory (Section 4.1), the possibility of cancellation (Section 4.2), and performance of the algorithm in practice (Section 4.3).

4.1. Convergence

In this section we will discuss the convergence of the recursion for evaluation of the function D (cf. Section 2.1). We will show that convergence is at least linear, and we will derive both a priori and a posteriori error bounds.

Throughout the section we will assume that $x \leq 0$, $\varrho \geq 0$. We start by noting that

$$\begin{aligned} \text{for } k \geq 0 \text{ even}: \quad & a_{k+1} \leq 0, \quad b_{k+1} \geq 0, \quad a_{k+1} + b_{k+1} \geq 0, \quad d_k \geq 0; \\ \text{for } k \geq -1 \text{ odd}: \quad & a_{k+1} \geq 0, \quad b_{k+1} \leq 0, \quad a_{k+1} + b_{k+1} \leq 0, \quad d_k \leq 0. \end{aligned}$$

Proofs are easy by induction. In particular, observe that

$$a_1 + b_1 = x^2 \cdot \sqrt{1 - \varrho^2} \cdot \frac{2\varrho}{1 + \varrho} \geq 0$$

and that $d_0 \geq 0$ due to the convexity of arcsin on $[0, 1]$. Moreover, for $k \geq 0$ even, we have

$$a_{k+3} + b_{k+3} = \frac{\alpha \cdot a_{k+1} + \beta \cdot b_{k+1}}{k+3} \geq \frac{\alpha \cdot (a_{k+1} + b_{k+1})}{k+3} \geq 0.$$

Analogously, we find

$$a_0 + b_0 = \left(\sqrt{1 - \varrho} \cdot \left(\sqrt{1 + \varrho} - \sqrt{1 - \varrho}\right)\right) \cdot \sqrt{\frac{\pi}{2}} \cdot x \leq 0$$

and, for $k \geq -1$ odd,

$$a_{k+3} + b_{k+3} = \frac{\alpha \cdot a_{k+1} + \beta \cdot b_{k+1}}{k+3} \leq \frac{\beta \cdot (a_{k+1} + b_{k+1})}{k+3} \leq 0.$$

Now, for $k \geq 0$ even, we conclude that

$$\begin{aligned}\frac{1+\delta}{k+2} \cdot (d_k + b_{k+1}) &\geq \frac{b_{k+1} + \delta \cdot d_k}{k+2} + \frac{\delta \cdot b_{k+1}}{k+2} \\ &\geq \frac{a_{k+1} + b_{k+1} + \delta \cdot d_k}{k+2} + \frac{\delta}{\beta} \cdot \frac{k+3}{k+2} \cdot b_{k+3} \geq d_{k+2} + b_{k+3}\end{aligned}$$

and, by induction,

$$\begin{aligned}\sum_{i \geq 2m, i \text{ even}} d_i &\leq (d_{2m} + b_{2m+1}) \cdot \left(1 + \frac{1+\delta}{2m+2} + \frac{(1+\delta)^2}{(2m+2)(2m+4)} + \dots\right) \\ &= (d_{2m} + b_{2m+1}) \cdot \sum_{k=0}^{\infty} \frac{m! \cdot (1+\delta)^k}{2^k \cdot (m+k)!} \\ &\leq (d_{2m} + b_{2m+1}) \cdot \sum_{k=0}^{\infty} \frac{\left(\frac{1+\delta}{2}\right)^k}{k!} = (d_{2m} + b_{2m+1}) \cdot \exp\left(\frac{1+\delta}{2}\right).\end{aligned}$$

Analogously, for $k \geq -1$ odd, we find that

$$\frac{1+\delta}{k+2} \cdot (d_k + b_{k+1}) \leq d_{k+2} + b_{k+3}$$

and, by induction,

$$\begin{aligned}\sum_{i \geq 2m, i \text{ odd}} d_i &\geq (d_{2m+1} + b_{2m+2}) \cdot \left(1 + \frac{1+\delta}{2m+3} + \frac{(1+\delta)^2}{(2m+3)(2m+5)} + \dots\right) \\ &\geq (d_{2m+1} + b_{2m+2}) \cdot \left(1 + \frac{1+\delta}{2m+2} + \frac{(1+\delta)^2}{(2m+2)(2m+4)} + \dots\right) \\ &\geq (d_{2m+1} + b_{2m+2}) \cdot \exp\left(\frac{1+\delta}{2}\right).\end{aligned}$$

Combining the above inequalities, we find the following a posteriori error bound:

$$\begin{aligned}\left|\sum_{i \geq 2m} d_i\right| &\leq \max\left(\sum_{i \geq 2m, i \text{ even}} |d_i|, \sum_{i \geq 2m, i \text{ odd}} |d_i|\right) \\ &\leq \max\left(|d_{2m} + b_{2m+1}|, |d_{2m+1} + b_{2m+2}|\right) \cdot \exp\left(\frac{1+\delta}{2}\right)\end{aligned}$$

Applying the inequalities again, an a priori error bound can be derived as well:

$$\left|\sum_{i \geq 2m} d_i\right| \leq \max\left(|d_0 + b_1|, |d_1 + b_2|\right) \cdot \frac{\left(\frac{1+\delta}{2}\right)^m}{m!} \cdot \exp\left(\frac{1+\delta}{2}\right)$$

Consequently, the order of convergence of the recursion is at least that of an exponential series. That is, convergence is at least linear. The above error bounds, of course, are rather weak. In particular, they are ignoring the alternating signs of the d_k. We will therefore complement the analysis by investigating the possibility of cancellation (Section 4.2), and the performance of the algorithm in practice (Section 4.3).

4.2. Cancellation

Since the signs of the d_k are alternating, cancellation may become an issue. However, it can be avoided by choosing an appropriate order of summation. Numerical evidence leads to the following conjecture:

Conjecture: The sequence $(d_k)_{k\geq -1}$ is log-concave, i.e., for all $k \geq -1$ the following Turán-type inequality holds:

$$d_{k+2} \cdot d_k \leq d_{k+1}^2 \tag{23}$$

A proof of this conjecture is outside the scope of this paper. Note that related statements have been proved for Mills' ratio, cf. Baricz (2008).

If the conjecture holds and if there is $m \geq 0$ with $s_m := d_{2m} + d_{2m+1} \geq 0$ then

$$-1 \geq \frac{d_{2m}}{d_{2m+1}} \geq \frac{d_{2m+2}}{d_{2m+3}}$$

and therefore $s_{m+1} = d_{2m+2} + d_{2m+3} \geq 0$ as well. That is, the sequence $(s_m)_{m\geq 0}$ changes signs not more than once. More precisely, we have

$$s_0 \leq 0 \quad \Longleftrightarrow \quad x \geq x(\varrho) := \sqrt{\frac{2}{\pi}} \cdot \frac{\arcsin(\varrho) - \varrho \cdot \pi/2}{\varrho - 1 + \sqrt{1-\varrho^2}}.$$

The function $x(\varrho)$ is decreasing in ϱ with

$$\lim_{\varrho \longrightarrow 0} x(\varrho) = \sqrt{\frac{2}{\pi}} - \sqrt{\frac{\pi}{2}} \approx -0.455, \qquad \lim_{\varrho \longrightarrow 1} x(\varrho) = -\sqrt{\frac{2}{\pi}} \approx -0.798.$$

In order to avoid cancellation, it is therefore reasonable to group the d_k into pairs.

4.3. Performance

Evaluation of $\Phi_2(x, y; \varrho)$ as in Section 3 will require (at most) four calls to an implementation of the cumulative standard normal distribution Φ (`Phi()` in the code). The actual choice may well determine both precision and running time of the algorithm. For testing purposes I have been using a hybrid method, calling the algorithm from West (2005, Figur 2) (a C++ implementation is available from West (2006)) for absolute value of the argument larger than 0.5, and `Phi()` from Marsaglia (2004) else. Besides `Phi()`, `exp()` will be called two times, `asin()` or `acos()` two times, and `sqrt()` six times. Everything else is elementary arithmetic.

Due to the reduction algorithm, the final result will be a sum. Therefore, very high precision in terms of relative error can not be expected. Consequently, evaluation of the diagonal aims at absolute error as well.

The `Phi2diag()` function is behaving as it may be expected from an approximation by a Taylor series around zero: (absolute) error increases with decreasing x. For $x < -7$ (or $\varrho \longrightarrow 0$ or $\varrho \longrightarrow 1$) the error bounds from Equation 1 are taking over, and absolute error decreases again. The maximum absolute error is obtained for $x \approx -7$, $\varrho \approx 0.8$ (maximum error of the upper bound is obtained for $\varrho = \sqrt{1 - 4/\pi^2} \approx 0.7712$, cf. Meyer (2013, Theorem 5.2)).

In general, assuming that all numerical fallacies in the reduction algorithm have been taken care of, the diagonal is expected to provide a worst case because the errors of the two calls

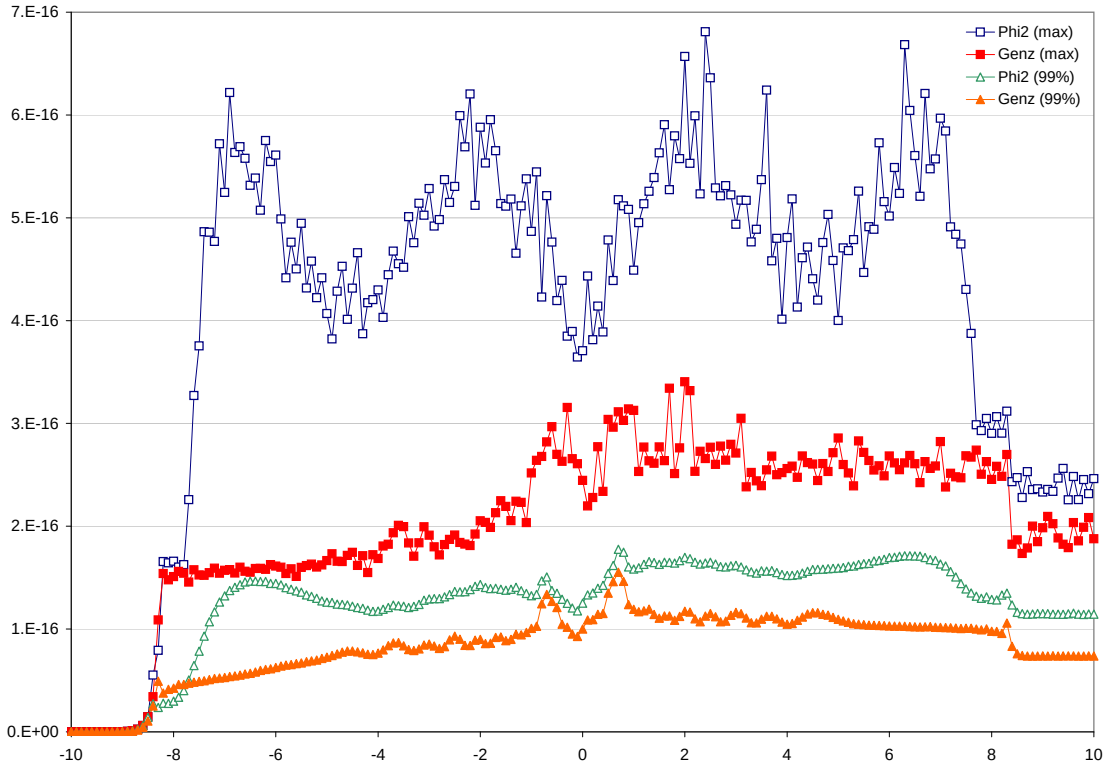

Figure 1: Absolute error of implementations of Φ_2 in a simulation study.

to `Phi2diag()` will not cancel. With respect to the reduction algorithm, the case $\varrho x \approx y$, $\varrho y \approx x$, implying $|\varrho| \approx 1$, is most critical.

In order to give an impression of the algorithm's behaviour, we will discuss the results of a simulation study. For each $n \in \{0, \ldots, 200\}$, $m \in \{1, \ldots, 10^6\}$, the value of $\Phi_2(x_m^n, y_m^n; \varrho_m^n)$ has been computed via the `Phi2()` function from Section 3.2 where x_m^n has been drawn from a uniform distribution on $[x^n - 0.05, x^n + 0.05]$ with $x^n := n/10 - 10$, y_m^n has been drawn from a uniform distribution on $[-10, 10]$, and $\varrho_m^n := 2\Phi(r_m^n) - 0.5$ where r_m^n has been drawn from a uniform distribution on $[-10, 10]$ as well.

The C++ implementation available from West (2006) of the Algorithm of Genz (2004) has been serving as a competitor. Both functions have been evaluated against a quad-double precision version of `Phi2()`, implemented using the **QD** library of Bailey, Hida, and Li (2010) and quad-double precision constants.

The diagram in Figure 1 is displaying, for $n \in \{0, \ldots, 200\}$, the 99% quantile and the maximum of the absolute difference between the double precision algorithms (`Phi2` and `Genz`) and the quad-double precision algorithm.

Apart from a shift due to subtractions for positive x, errors of `Phi2` are rather symmetric around zero. The peaks at $|x^n| \approx 7$ are due to the Taylor expansion around zero; the peaks at $|x^n| \approx 2$ are due to Taylor expansion after transformation of the argument. The characteristics of the 99% quantile, in particular the little peaks at $|x^n| \approx 0.7$, are already visible in the error of the Φ function used. The maximum error of `Genz` almost always stays below the one of

`Phi2`. Note that the maximum error of `Genz` is determined by the case $\varrho \longrightarrow -1$ and might be reduced by careful consideration of that case.

In the simulation study, `Phi2` was a little slower than `Genz`: it took approximately five minutes and four minutes to perform the $201 \cdot 10^6$ evaluations on a fairly standard office PC (and it took two days to perform the corresponding quad-double precision evaluations). The number of recursion steps used by `Phi2diag` is increasing with $|x|$. It should be easy to find an appropriate trade-off between speed and precision (maybe even equal precision with less running time) by replacing the condition terminating the recursion.

Acknowledgments

The author wishes to thank Axel Vogt for helpful discussion.

References

Ağca Ş, Chance DM (2003). "Speed and Accuracy Comparison of Bivariate Normal Distribution Approximations for Option Pricing." *Journal of Computational Finance*, **6**(4), 61–96.

Bailey DH, Hida Y, Li XS (2010). "QD (C++/Fortran-90 Double-Double and Quad-Double Package)." URL http://crd.lbl.gov/~dhbailey/mpdist/.

Balakrishnan N, Lai CD (2009). *Continuous Bivariate Distributions*. 2nd edition. Springer-Verlag, New York.

Baricz Á (2008). "Mills' Ratio: Monotonicity Patterns and Functional Inequalities." *Journal of Mathematical Analysis and Applications*, **340**, 1362–1370.

Divgi DR (1979). "Calculation of Univariate and Bivariate Normal Probability Functions." *The Annals of Statistics*, **7**(4), 903–910.

Drezner Z (1978). "Computation of the Bivariate Normal Integral." *Mathematics of Computation*, **32**(141), 277–279.

Drezner Z, Wesolowsky GO (1990). "On the Computation of the Bivariate Normal Integral." *Journal of Statistical Computation and Simulation*, **35**, 101–107.

Genz A (2004). "Numerical Computation of Rectangular Bivariate and Trivariate Normal and t Probabilities." *Statistics and Computing*, **14**(3), 151–160.

Genz A, Bretz F (2009). *Computation of Multivariate Normal and t Probabilities*. Springer-Verlag, New York.

Hull J (2008). *Options, Futures, and Other Derivatives*. 6th edition. Prentice Hall.

Kotz S, Balakrishnan N, Johnson, Lloyd N (2000). *Continuous Multivariate Distributions*, volume 1: Models and Applications. 2nd edition. John Wiley & Sons.

Marsaglia G (2004). "Evaluating the Normal Distribution." *Journal of Statistical Software*, **11**(4), 1–11. URL http://www.jstatsoft.org/v11/i04/.

Meyer C (2013). "The Bivariate Normal Copula." *Communications in Statistics – Theory and Methods*. Forthcoming, preprint available at http://arxiv.org/abs/0912.2816.

Patel JK, Read CB (1996). *Handbook of the Normal Distribution*. 2nd edition. Dekker.

Terza JV, Welland U (1991). "A Comparison of Bivariate Normal Algorithms." *Journal of Statistical Computation and Simulation*, **39**(1-2), 115–127.

Vogt A (2008). "Computing the Cumulative Bivariate Normal Distribution." URL http://www.axelvogt.de/axalom/.

Wang MC, Kennedy WJ (1990). "Comparison of Algorithms for Bivariate Normal Probability over a Rectangle Based on Self-Validated Results from Interval Analysis." *Journal of Statistical Computation and Simulation*, **37**(1-2), 13–25.

West G (2005). "Better Approximations to Cumulative Normal Functions." *Wilmott Magazine*, pp. 70–76.

West G (2006). "Code for Univariate, Bivariate and Trivariate Cumulative Normal Functions." URL http://www.finmod.co.za/research.htm.

Affiliation:

Christian Meyer
DZ BANK AG
Platz der Republik
60265 Frankfurt, Germany
E-mail: christian.meyer@dzbank.de

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 52, Issue 10 *Submitted:* 2010-04-28
March 2013 *Accepted:* 2012-12-18