A simple and efficient algorithm to compute average Probability of False Accept for measurement decision rules

Author/Speaker: Collin J. Delker
Sandia National Laboratories
P.O. Box 5800, MS 0665
Albuquerque, New Mexico, 87185-0665
Phone: 505-845-7431 Email: cjdelke@sandia.gov

Abstract

A simple and efficient algorithm for computing average Probability of False Accept (PFA) to use in a measurement decision rule is presented. The algorithm is an analytical expression that does not require iterative or numerical methods to solve, making it straightforward to implement in general-purpose computing languages. It may be applied in spreadsheets without macros, allowing for easy distribution and adoption while avoiding the security challenges that come with distributing compiled software. The algorithm is based on converting the parameters used in the PFA calculation, which are derived from the measurement uncertainty and probability distribution of Devices Under Test (DUT), into parameters used by the textbook bivariate normal distribution. Then, a fast approximation to integrating the standard bivariate normal is used to calculate the PFA. While it assumes normal probability distributions, the algorithm is not limited to two-sided tolerances or a centered distribution of DUTs. A Microsoft Excel implementation of the algorithm is included, along with a comparison of execution speed and accuracy with respect to numerical calculation methods. The algorithm may also be combined with a bracketing solver algorithm to compute the acceptance limits that result in a desired PFA. This algorithm lowers the barrier to computing PFA for use in measurement decision rules, and will help reduce reliance on outdated and limited metrics like Test Uncertainty Ratios for ensuring measurement adequacy.

Learning Objectives:

1. Understand how the average PFA of a measurement is related to the bivariate normal probability distribution
2. Recognize different general approaches to integrating the bivariate normal distribution
3. Apply spreadsheet LAMBDA functions for implementing reusable algorithms such as PFA and integration of the bivariate normal

1. Introduction

Historically, the Test Uncertainty Ratio (TUR) has been used as a simplified metric for estimating the level of risk when making a statement of conformity to a tolerance based on a measurement. TUR-based criteria are commonly used as part of "decision rules" required by ISO/IEC 17025:2017 [1], where a TUR evaluation is used to account for the level of risk as required by the ISO standard. TUR is related to the risk of false acceptance in a conformity assessment. For example, these two statements may nominally be equivalent:

1. This measurement has a TUR of 4.
2. This measurement will, on average, result in 2 % of all calibrations being incorrectly accepted.

The second statement, written directly in terms of the average Probability of False Accept (PFA), is arguably more useful for informing a risk evaluation. While TUR is related to PFA, the PFA is also affected by the distribution of Devices Under Test (DUTs), often characterized by End-Of-Period Reliability (EOPR, also known as in-tolerance probability), so TUR does not always provide a one-to-one correlation with PFA. While a TUR of 4 results in a worst-case PFA of about 2 %, in other situations a TUR of 4 may indicate a PFA less than 1 % [2], [3]; for example, in a calibration lab that can demonstrate an EOPR of 95 %. Additionally, TUR comes with often unstated limitations and may be used with unjustified assumptions. It only works in certain measurement situations, such as those with symmetric two-sided tolerances, centered distribution of DUTs, and normal probability distributions [4], [5]. Typical TUR rules generally require TUR ≥ 4, or guardbanding applied when TUR < 4, but 4 is an arbitrary threshold that assumes an acceptable or appropriate level of risk for the given measurement [6]. The Test Accuracy Ratio (TAR), which does not account for all uncertainties in a measurement, is still sometimes encountered, and suffers from the same limitations and assumptions as TUR.

For these reasons, instead of always relying on TUR and its assumptions, the metrology industry is shifting toward calculating average PFA, also referred to as global PFA, for many of its decision rules [7], [8]. Unfortunately, PFA is also more difficult to calculate, requiring the double integration of a joint Probability Density Function (PDF) compared to the simple division used to determine the TUR.

The difficulty in calculating PFA led to the introduction of TAR in the 1950s, although its originators intended its use to be "temporary until better computing power became available" [9]. Obviously, computing power is slightly better than it was 70 years ago. With modern computing options and an array of statistical computing languages available, PFA calculations may readily be implemented, yet remain more complicated and slower to compute than TAR or TUR, leaving many measurement assurance plans still relying on a 70-year-old metric and its assumptions.

For the widest adoption of PFA as an alternative to TUR, calculations implemented entirely in spreadsheets are often preferred over more capable statistical libraries. While custom

software such as Suncal [10] are capable of PFA calculation, as niche software, it can be challenging for corporate and government users to gain approval and install these tools on their networks. Spreadsheet software, however, is commonly available, and spreadsheet files are typically distributed as "documents" rather than "software" in terms of information technology policy, so they do not come with the security restrictions or distribution challenges of compiled executable code. Spreadsheets are easily auditable, with all calculations and results visible in one document. However, spreadsheet macros should be avoided as they are often disabled by corporate security policies.

This paper introduces a simple and efficient algorithm for calculating PFA based on fast approximations of the standard bivariate normal distribution integral. While the algorithm is necessarily more complex than TUR, it remains a simple analytical expression; it does not require iterative methods or vectorizing the probability distributions before being integrated numerically, making it much faster to compute than numeric methods. This allows the algorithm to be implemented in a spreadsheet such as Microsoft Excel without use of macros or thousands of cells to enumerate an integral. The calculation is much quicker than other approaches using numerical methods. While the speed difference for a single PFA computation may not be noticeable, measurement assurance spreadsheets often involve hundreds of test points, all with their own PFA calculations. Furthermore, a common use case is to solve for the guardbanded acceptance limits that result in a desired PFA, a calculation that requires iterative methods computing PFA a few dozen times before settling on a solution. In these applications, speed improvements are compounded and become significant and noticeable to the user. In a measurement assurance spreadsheet implemented at Sandia National Laboratories, use of the new algorithm reduced the calculation time for finding a guardband from several seconds to milliseconds, improving the user experience while maintaining accurate results.

An example spreadsheet demonstrating the proposed PFA implementation, along with a corresponding Probability of False Reject (PFR) function, is found at [10].

2. Probability of False Accept

The average PFA combines the probability of encountering a DUT having a given attribute value with the probability distribution associated with the uncertainty of a measurement on that attribute. In this paper, the DUTs are either products being manufactured or measuring and test equipment being calibrated.

Assuming normal probability distributions, the distribution of DUT attributes is

$$p_p(x) = \frac{1}{\sqrt{2\pi}\sigma_p} \exp\left(\frac{-\left(x-\mu_p\right)^2}{2\sigma_p^2}\right) \qquad (1)$$

where σ_p is the standard deviation and μ_p the mean value of the DUT attribute population, parameters typically estimated from historical measurement data or EOPR.

The probability distribution resulting from measuring a DUT with true attribute value x is

$$p_m(y,x) = \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(\frac{-(y-x)^2}{2\sigma_m^2}\right) \qquad (2)$$

where σ_m is the standard measurement uncertainty.

The product of these two probability distributions, illustrated in Figure 1, is a combination of possible true attribute values with a measurement result centered on the true attribute value [3]:

$$p(y,x) = \frac{1}{2\pi\sigma_m\sigma_p} \exp\left(\frac{-(y-x)^2}{2\sigma_m^2}\right) \exp\left(\frac{-\left(x-\mu_p\right)^2}{2\sigma_p^2}\right) \qquad (3)$$

False accept decisions occur when the DUT's true attribute value falls above or below the tolerance, but the measurement indicates the attribute falls within the limits. This may occur for attributes above the upper limit or below the lower limit. PFA may be calculated by integrating the joint PDF over these two regions:

$$PFA = \int_{-\infty}^{T_L}\int_{A_L}^{A_U} p\,(x,y)dydx + \int_{T_U}^{\infty}\int_{A_L}^{A_U} p\,(x,y)dydx \qquad (4)$$

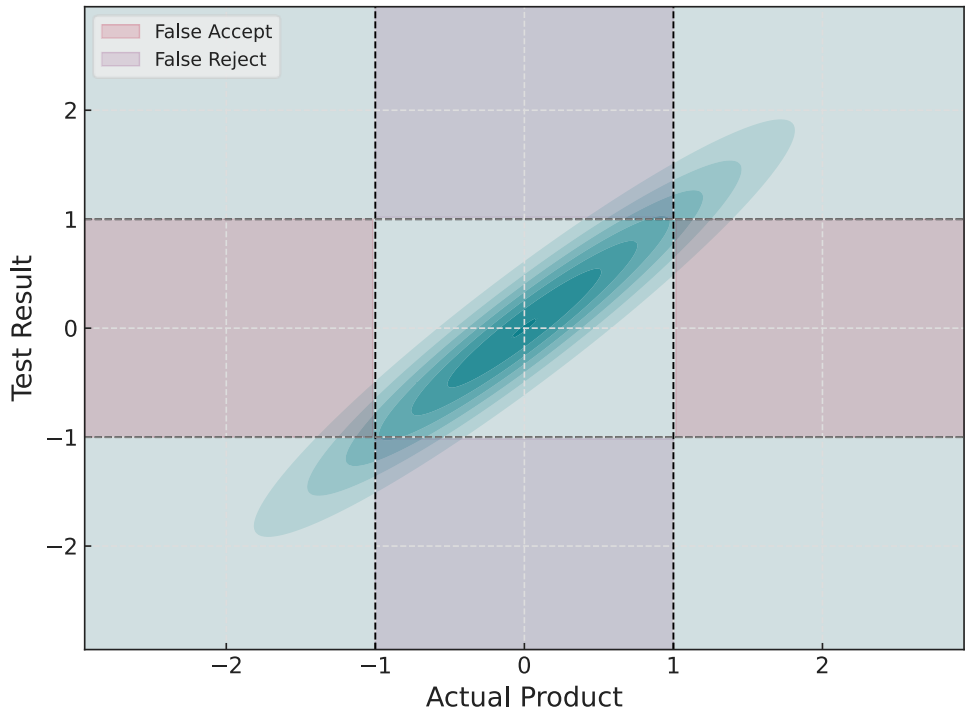

Figure 1: Joint PDF of measurements showing false accept and false reject regions.

Here, T_L and T_U are the lower and upper tolerance limits, and A_L and A_U are the lower and upper acceptance (guardbanded) limits. Equation 4 cannot be solved analytically, which traditionally makes PFA calculation difficult to implement.

2.1 Integrating the PFA Probabilities

Before turning to the proposed PFA algorithm, two numerical integration approaches are presented to illustrate the traditional methods of computing PFA.

2.1.1 Direct Double Integration

Most statistical computation libraries include a function for numerically performing double integrations. For example, using Python's SciPy, PFA may be computed using the `scipy.integrate.dblquad` function:

```
def pxy(y, x, sigm, sigp, mup):
    # Calculate joint probability p(x, y)
    return exp(-(y-x)**2 / (2*sigm**2)) * exp(-(x-mup)**2 / (2*sigp**2))

# Integrate p(x, y) over the two false accept regions
pfa = 1/(2*pi*sigm*sigp) * (
    dblquad(pxy, -inf, TL, AL, AU) +
    dblquad(pxy, TU, inf, AL, AU)
)
```

It uses an adaptive Gaussian Quadrature algorithm to compute the integration in both axes [11]. While producing highly accurate results, it relies on iterative (slow) methods and requires installing Python and SciPy (or equivalent statistical libraries).

2.1.2 Simplification Using Single Integration

The PFA integration may be simplified by recognizing that a single integral over a normal distribution is by definition a normal Cumulative Distribution Function (CDF). Most computing languages and common spreadsheet applications include a direct implementation of the normal CDF (or the underlying error function) that does not rely on numerical techniques. Using the standard normal CDF, $\Phi(x)$, the PFA may be reduced to a single integration:

$$PFA = \int_{-\infty}^{T_L} [\Phi(A_U/\sigma_m) - \Phi(A_L/\sigma_m)]\, e^{\frac{-(x-\mu_p)^2}{2\sigma_p^2}}\, dx + \int_{T_U}^{\infty} [\Phi(A_U/\sigma_m) - \Phi(A_L/\sigma_m)]\, e^{\frac{-(x-\mu_p)^2}{2\sigma_p^2}}\, dx \qquad (5)$$

In Python, the `scipy.integrate.quad` function may be used:

```
def pxy(y, x, sigm, sigp, mup):
    # Calculate joint probability p(x, y)
    return (cdf(AU/sigm) - cdf(AL/sigm)) * exp(-(x-mup)**2 / (2*sigp**2))

# Integrate p(x, y) over the two false accept regions
pfa = 1/(2*pi*sigm*sigp) * (
    quad(integrand, -inf, TL) +
    quad(integrand, TU, inf)
)
```

This is much faster, but still requires numerical integration along one dimension. Note that further simplification may also be applied if the limits are symmetric about the nominal value μ_p.

2.1.3 Spreadsheet Implementation

The PFA calculation in Equation 5 was implemented in a spreadsheet as a first attempt to incorporate PFA into a spreadsheet measurement assurance plan. This method requires a numerical integration function, but this is not typically a built-in spreadsheet function. An integration function may, however, be implemented using LAMBDA functions[1].

A LAMBDA function defines a named function with generic input parameters and an output parameter available anywhere in the spreadsheet. It provides a reusable function, working identically to a built-in spreadsheet function, and may be added to any cell in the spreadsheet and reference any other cells as inputs.

For example, a simple Root-Sum-of-Squares (RSS) function may be defined using LAMBDA:

```
=LAMBDA(a, b, SQRT(a^2+b^2))
```

The first two parameters define the names of input variables. The last parameter defines the output of the function in terms of the input variables. In Excel, this function would be added to the Name Manager dialog and given a name, such as `RSS`. It could then be used in a spreadsheet cell, for example `=RSS(A1, B1)` to RSS the values of cells `A1` and `B1`. While this example may not be highly useful, as an RSS may be done easily without LAMBDA functions, the power and flexibility of LAMBDA functions becomes apparent when realizing that LAMBDA functions may be called recursively and the function itself may be passed as a parameter to other functions. This enables LAMBDAs to be used in more complicated algorithms, such as a trapezoidal integration function.

A trapezoidal integration that operates on an *arbitrary* function `f` may be defined:

```
=LAMBDA(lower, upper, dx, f,
    LET(seq, SEQUENCE(0+(upper-lower)/dx, 1, lower+dx, dx),
        REDUCE(0, seq,
            LAMBDA(acc, b, acc + (dx / 2) * (f(b) + f(b-dx)))
        )
    )
)
```

[1] In Microsoft Excel, LAMBDA functions were added in Office 365 and 2024 versions [12]. Google Sheets implements LAMBDA as Named Functions. At the time of publication, Libreoffice has not implemented LAMBDA, but there is an open feature request.

Here, the input parameters `lower` and `upper` are the limits of integration, `dx` the spacing between points, and `f` another LAMBDA function to integrate. The function sets up a sequence of points spaced `dx` apart over the integration interval, then totals (using `REDUCE`) the area of the trapezoids as computed by `f` at each point.

This user-defined function, assigned the name `TRAPZ`, can integrate arbitrary functions in a single cell. For example, placing `=TRAPZ(0, PI(), .001, LAMBDA(x, SIN(x^2)))` in a cell will numerically calculate $\int_0^{\pi} \sin(x^2)dx$.

Now `TRAPZ` may be used to create a function for calculating PFA:

```
=LAMBDA(tlimit, alimit, sigm, sigp,
    LET(upperinf, sigp*6,
        pfint, TRAPZ(tlimit, upperinf, ABS(upperinf-tlimit)/500,
        LAMBDA(x, (NORM.DIST(alimit-x, 0, sigm, TRUE)
                   -NORM.DIST(-alimit-x, 0, sigm, TRUE)
                   )*EXP(-(x^2)/(2*sigp^2)))),
        2 * pfint / (SQRT(2*PI())*sigp)
        )
)
```

The inner LAMBDA defines a function for computing the integrand of Equation 5 over which `TRAPZ` will integrate. `NORM.DIST` is a built-in function that computes the standard normal CDF. Since `TRAPZ` cannot integrate over infinite limits, the upper integration limit was set to 6 standard deviations of the product distribution. Somewhat arbitrarily, 500 points along the interval were selected to result in reasonable tradeoff between accuracy calculation time. However, this implementation remains slow to calculate and requires an arbitrary upper limit and number of datapoints, affecting accuracy.

While this implementation allows PFA calculation in a spreadsheet, calculation speed becomes an issue, especially when adding iterative methods to solve for guardbanding limits, which requires a few dozen PFA calculations to settle on a solution.

3. Alternative PFA Calculation

A more efficient PFA calculation can be obtained, eliminating the need to vectorize the PFA integrand into discrete points for numerical integration. Consider the integrand of PFA (repeating Equation 3):

$$p(y,x) = \frac{1}{2\pi\sigma_m\sigma_p}\exp\left(\frac{-(y-x)^2}{2\sigma_m^2}\right)\exp\left(\frac{-\left(x-\mu_p\right)^2}{2\sigma_p^2}\right) \qquad (6)$$

Notice this equation defines a bivariate normal probability distribution based on parameters known from the measurement: the measurement standard uncertainty σ_m and the standard deviation of the product distribution σ_p. The ellipse in Figure 1 illustrates the distribution, where measurement results, along the vertical direction, are clearly correlated with the true value of the DUT attribute being measured in the horizontal direction.

A textbook bivariate normal distribution is often defined in a slightly different form [13]:

$$p(y,x) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[\frac{-1}{2(1-\rho^2)}\left(\frac{(x-\mu_x)^2}{\sigma_1^2} + \frac{(y-\mu_y)^2}{\sigma_2^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_1\sigma_2}\right)\right] \quad (7)$$

where ρ is the correlation coefficient between the two related normal distributions. The PFA distribution clearly exhibits correlation, but the correlation coefficient is unknown. By rearranging both forms of $p(y,x)$, and letting $\mu_x = \mu_y = 0$, the two forms of bivariate normal distributions may be written as:

$$p_{BVN}(y,x) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_1^2(1-\rho^2)} + \frac{y^2}{\sigma_2^2(1-\rho)^2} - \frac{2\rho xy}{\sigma_1\sigma_2(1-\rho^2)}\right)\right] \quad (8)$$

$$p_{PFA}(y,x) = \frac{1}{2\pi\sigma_m\sigma_p} \exp\left[-\frac{1}{2}\left(x^2\frac{\sigma_m^2+\sigma_p^2}{\sigma_m^2\sigma_p^2} + \frac{y^2}{\sigma_m^2} - \frac{2xy}{\sigma_m^2}\right)\right] \quad (9)$$

The PFA integrand now appears in the same form as the textbook bivariate normal. Matching the coefficients results in a system of equations relating the PFA standard deviations to the textbook bivariate standard deviations and correlation coefficient.

$$\sigma_1\sigma_2\sqrt{1-\rho^2} = \sigma_m\sigma_p \quad (10)$$

$$\sigma_1^2(1-\rho^2) = \left(\sigma_m^2\sigma_p^2\right)/\left(\sigma_m^2+\sigma_p^2\right) \quad (11)$$

$$\sigma_2^2(1-\rho^2) = \sigma_m^2 \quad (12)$$

$$\sigma_1\sigma_2(1-\rho^2)/\rho = \sigma_m^2 \quad (13)$$

Solving this system of equations produces a translation between the known measurement parameters σ_m and σ_p and the textbook parameters σ_1, σ_2, and ρ.

$$\sigma_1 = \sigma_p \quad (14)$$

$$\sigma_2 = \sigma_m^2 + \sigma_p^2 \qquad (15)$$

$$\rho = \sqrt{\frac{\sigma_p^2}{\sigma_p^2 + \sigma_m^2}} \qquad (16)$$

Now, with the PFA integrand converted to textbook form, it may be integrated using any number of published fast approximations for integrating the bivariate normal, such as [14], [15], or [16].

3.1 PFA in Terms of the Standard Bivariate Normal Integral

Most algorithms integrate the *standard* bivariate normal distribution, where $\sigma_1 = \sigma_2 = 1$, from negative infinity to a given point (p, q):

$$\Phi(p, q, \rho) = \int_{-\infty}^{p} \int_{-\infty}^{q} \phi\,(x, y, \rho) dx dy$$

To use $\Phi(p, q, \rho)$ to find PFA, the parameters need to be scaled to standard form, and Φ must be calculated twice to obtain the integral over finite limits. Starting with case of symmetric limits, where $T = (T_U - \mu_p) = (\mu_p - T_L)$ and $A = (A_U - \mu_p) = (\mu_p - A_L)$, the PFA due to false accepts below the lower tolerance is:

$$PFA_L = \Phi\left(\frac{A}{\sigma_2}, \frac{-T}{\sigma_1}, \rho\right) - \Phi\left(\frac{-A}{\sigma_2}, \frac{-T}{\sigma_1}, \rho\right) \qquad (17)$$

The formula is illustrated in Figure 2, where the first term of Equation 17 integrates the probability of falling below the lower limit, shown by the dashed rectangle, while the second term subtracts out the probability of correctly rejecting the item, shown by the dotted rectangle.

Because of symmetry, PFA_R, the PFA due to false accepts above the upper tolerace, will be the same, so:

$$PFA = 2PFA_L = 2\left[\Phi\left(\frac{A}{\sigma_2}, \frac{-T}{\sigma_1}, \rho\right) - \Phi\left(\frac{-A}{\sigma_2}, \frac{-T}{\sigma_1}, \rho\right)\right] \qquad (18)$$

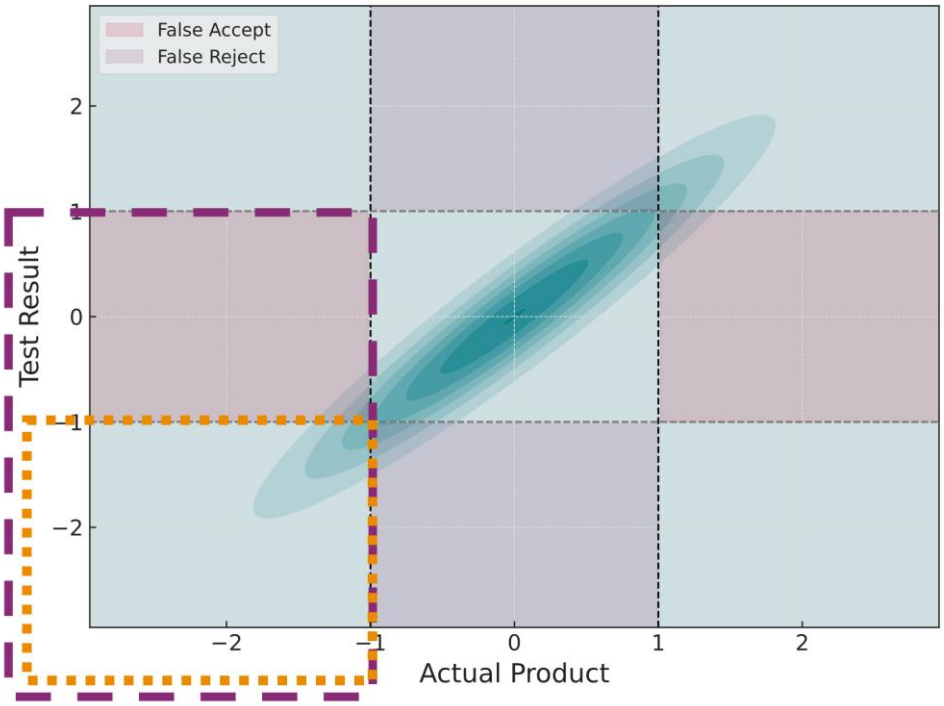

Figure 2: PFA calculated using two integrations of the bivariate normal distribution.

In the asymmetric case, there are different false accept probabilities on each side of the tolerance zone, so both sides must be calculated separately (inverting signs for the right side):

$$PFA_L = \left[\Phi\left(\frac{A_U - \mu_p}{\sigma_2}, \frac{T_L - \mu_p}{\sigma_1}, \rho\right) - \Phi\left(\frac{A_L - \mu_p}{\sigma_2}, \frac{T_L - \mu_p}{\sigma_1}, \rho\right)\right] \qquad (19)$$

$$PFA_R = \left[\Phi\left(\frac{-A_L + \mu_p}{\sigma_2}, \frac{-T_U + \mu_p}{\sigma_1}, \rho\right) - \Phi\left(\frac{-A_U + \mu_p}{\sigma_2}, \frac{-T_U + \mu_p}{\sigma_1}, \rho\right)\right] \qquad (20)$$

$$PFA = PFA_L + PFA_R \qquad (21)$$

3.2 Integrating the Standard Bivariate Normal Distribution

To calculate $\Phi(p, q, \rho)$, the algorithm described by *Tsay and Ke* [14] was selected for its speed and accuracy. The algorithm is split into five cases depending on the magnitudes of p, q, and ρ. In the PFA problem, $\rho > 0$, so only two of the five cases need to be implemented.

From the Tsay-Ke algorithm, if $aq + b \geq 0$,

$$\Phi(p,q,\rho) \approx \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{q}{\sqrt{2}}\right) - \frac{1}{4\sqrt{1-a^2c_2}} \exp\left(\frac{a^2c_1^2 + 2\sqrt{2}bc_1 + 2b^2c_2}{4(1-a^2c_2)}\right) \times \left[1 + \operatorname{erf}\left(\frac{\sqrt{2}q - \sqrt{2}a^2c_2q - \sqrt{2}abc_2 - ac_1}{2\sqrt{1-a^2c_2}}\right)\right] \quad (22)$$

and if $aq + b < 0$,

$$\Phi(p,q,\rho) \approx \frac{1}{2} - \frac{1}{2}\operatorname{erf}\left(\frac{b}{\sqrt{2}a}\right) - \frac{1}{4\sqrt{1-a^2c_2}} \exp\left(\frac{a^2c_1^2 + 2\sqrt{2}bc_1 + 2b^2c_2}{4(1-a^2c_2)}\right) \times \left[1 - \operatorname{erf}\left(\frac{\sqrt{2}b + a^2c_1}{2a\sqrt{1-a^2c_2}}\right)\right] + \frac{1}{4\sqrt{1-a^2c_2}} \exp\left(\frac{a^2c_1^2 - 2\sqrt{2}bc_1 + 2b^2c_2}{4(1-a^2c_2)}\right) \times \left[\operatorname{erf}\left(\frac{\sqrt{2}q - \sqrt{2}a^2c_2q - \sqrt{2}abc_2 + ac_1}{2\sqrt{1-a^2c_2}}\right) + \operatorname{erf}\left(\frac{-a^2c_1 + \sqrt{2}b}{2a\sqrt{1-a^2c_2}}\right)\right] \quad (23)$$

where

$$a = \frac{-\rho}{(1-\rho^2)^{1/2}}, \; b = \frac{p}{(1-\rho^2)^{1/2}} \quad (24)$$

and $c_1 = -1.0950081470333$, $c_2 = -0.75651138383854$, and $erf(z)$ is the error function of z.

Using the Tsay-Ke approximation to Φ with Equation 18 or Equation 21 leads to a simple and efficient calculation of PFA when leveraging a highly accurate built-in error function available in most programming languages. Not requiring any numerical methods, it may readily be implemented in spreadsheet LAMBDA functions, and its execution time and accuracy are independent of any parameters chosen.

3.3 Fast PFA Algorithm

A summary of the proposed PFA algorithm follows.

1. Convert PFA parameters σ_m and σ_p into σ_1, σ_2, and ρ using Equation 14, Equation 15, and Equation 16.
2. Write PFA in terms of the standard $\Phi(p,q,\rho)$ using Equation 18 (symmetric limits) or Equation 21 (asymmetric limits).
3. Use the $\Phi(p,q,\rho)$ approximation Equation 22 and Equation 23 to calculate PFA.

Appendix A provides complete LAMBDA functions to implement the algorithm in a spreadsheet. Functions for PFR are also provided, following the same method but with modified integration limits. The appendix also includes an additional function for calculating acceptance limits A that results in a desired PFA.

3.3.1 Speed and Accuracy Comparison

The execution speed of the proposed PFA algorithm was compared to alternatives using a Python implementation. (Python was used over a spreadsheet because it provides a `%%timeit` tool for easily estimating execution times that is not readily available in Excel. While timing in a spreadsheet will not be identical, general trends are expected to be similar.) Four methods were compared: numerical integration of the full PFA integrand using `dblquad`, numerical integration `quad` with the CDF simplification, an alternate algorithm described in [15] for integrating the bivariate normal, and the Tsay-Ke algorithm presented above. Results for both a single calculation of the bivariate integral and for a calculation of an asymmetric PFA are presented in Table 1.

Table 1: Execution times for integrating the bivariate normal and computing PFA using different integration algorithms.

Method	Bivariate Integral	PFA
dblquad	84 000 µs	483 000 µs
quad	N/A	31 000 µs
Drezner	447 µs	689 µs
Tsay-Ke	36 µs	48 µs

A common use case for PFA calculation is to find the guardbanded limits (A) that result in a desired PFA. This calculation requires bracketing the limits and calculating PFA a few dozen times. For the slower algorithms, this may require tens of seconds, where the faster algorithm completes it in milliseconds.

Algorithm accuracy was also compared over a complete range of input parameters, with results checked against those from Suncal (which applies the `quad` method). For example, results of a PFA calculation with $T = 1$, $A = 0.9$, $\sigma_p = 0.51$, $\sigma_m = 0.125$ are compared in Table 2. Note that all methods are numerical approximations to PFA, with the `dblquad` and `quad` producing the most accurate results. However, with estimated uncertainty and historical standard deviations usually considered good to two significant figures, any differences in the accuracy of the algorithms seen here is insignificant.

Table 2: Accuracy comparison of PFA calculated using different integration algorithms.

Method	PFA
dblquad	0.275 560 508 607 280 %
quad	0.275 560 434 646 414 %
Drezner	0.275 463 105 374 944 %
Tsay-Ke	0.275 751 381 275 329 %

In Excel, rough timing can be estimated using a `TIMER` function developed as a LAMBDA available from Vertex42 [17]. Assuming symmetric tolerances and solving for guardbands, the time required to calculate a measurement assurance plan containing 100 testpoints was estimated as listed in Table 3 using the trapezoidal integration with two different resolutions and the Tsay-Ke approximation for integration.

Table 3: Execution time for calculating PFA in Excel using different integration algorithms.

Method	Time to compute 100 rows
trapz (n=5000)	63.0 seconds
trapz (n=500)	12.3 seconds
Tsay-Ke	0.12 seconds

4. Summary and Future Work

The algorithms for PFA and PFR provided in this paper may be readily implemented in code, including spreadsheets, without use of macros, reducing the burden of using PFA analysis in common measurement assurance documentation. It provides fast calculation and accuracy sufficient for evaluating PFA in measurement decision rules. This algorithm may be used to reduce reliance on outdated metrics such as TUR or TAR when documenting decision rules for ISO/IEC 17025:2017 compliance. It can assist in calculating optimal guardband limits and assessing the tradeoffs between false acceptance and false rejection in calibrations and the acceptance of products.

While the algorithm is limited to normal probability distributions, it is not limited to symmetric tolerances or two-sided tolerances as commonly encountered using metrics like TUR.

Future work could employ similar approaches to enable spreadsheet calculation of other statistical metrology calculations. For example, some of the calculations used in an End-to-end Measurement Quality Assurance model, as described in the draft NCSLI Recommended Practice 19 (RP-19) [18], could be efficiently evaluated using the $\Phi(p, q, \rho)$ algorithm with appropriate parameters. Other RP-19 calculations would need a trivariate normal approximation, a possible extension to the algorithm mentioned by Tsay and Ke, but not provided in their article.

Acknowledgements

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan. SAND2026-21045C.

5. References

[1] "ISO/IEC 17025:2017. General requirements for the competence of testing and calibration laboratories." 2017.

[2] S. M. Mimbs, "Using reliability to meet Z540. 3's 2 percent rule," in *2011 NCSL international symposium and workshop*, 2011.

[3] S. Crowder, C. Delker, E. Forrest, and N. Martin, *Introduction to statistics in metrology*. Springer Nature, 2020.

[4] C. J. Delker, "Evaluating risk in an abnormal world: How arbitrary probability distributions affect false accept and reject evaluation." 2023.

[5] C. J. Delker, "Guardbanding one-sided maximum and minimum tolerances." 2020.

[6] C. J. Delker, "Whose risk is it anyway? Considerations for risk-based decision rules." 2024.

[7] H. Zumbrun, G. Cenker, and D. Shah, "Decision rule guidance, 1st edition," https://mhforce.com/support-item/decision-rule-guidance-1st-edition-2024/, 2024.

[8] J. Harben and P. Reese, "Implementing strategies for risk mitigation in the modern calibration laboratory," in *NCSLI workshop and symposium*, 2011.

[9] S. M. Mimbs, "Measurement decision risk-the importance of definitions," in *2007 NCSL International Workshop and Symposium*, 2007.

[10] Sandia National Laboratories, "Suncal software for metrology statistics." https://sandiapsl.github.io.

[11] The SciPy Community, "Scipy documentation." https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.dblquad.html.

[12] Microsoft, "LAMBDA function." https://support.microsoft.com/en-us/office/lambda-function-bd212d27-1cd1-4321-a34a-ccbf254b8b67.

[13] Y. L. Tong, *The multivariate normal distribution*. Springer Science & Business Media, 2012.

[14] W. Tsay and P. Ke, "A simple approximation for the bivariate normal integral," *Communications in Statistics-Simulation and Computation*, vol. 52, no. 4, pp. 1462–1475, 2023.

[15] Z. Drezner, "Computation of the bivariate normal integral," *Mathematics of Computation*, vol. 32, no. 141, 1978.

[16] D. R. Cox and N. Wermuth, "A simple approximation for bivariate and trivariate normal integrals," *International Statistical Review/Revue Internationale de Statistique*, pp. 263–269, 1991.

[17] Vertex42, "The LAMBDA library." https://www.vertex42.com/lambda/.

[18] NCSLI, "NCSLI recommended practice 19: End-to-end measurement quality assurance," (Draft).

Appendix A. Spreadsheet Implementation

A.1 Bivariate Normal Cumulative Distribution Function

Using Cases 4 and 5 of [14], define `BIVARIATE_NORMCDF`. Note the other cases are not implemented (will result in `NA`), but could readily be added.

```
=LAMBDA(p,q,rho,
    LET(cx, -1.0950081470333,
        cy, -0.75651138383854,
        d, SQRT(1-rho^2),
        a, -rho/d,
        b, p/d,
        aqplusb, a*q + b,
        z, 1-a^2*cy,
        sqrtz, SQRT(z),
        sqrt2, SQRT(2),
        IF(a>=0, NA(),
            IF(aqplusb>=0,
                0.5 + 0.5 * ERF(q/sqrt2)
                - 1/(4*sqrtz) * EXP((a^2*cx^2+2*sqrt2*b*cx+2*b^2*cy)/(4*z))
                * (1 + ERF((sqrt2*q - sqrt2*a^2*cy*q - sqrt2*a*b*cy -
a*cx)/(2*sqrtz))),

                0.5 - 0.5 * ERF(b/(sqrt2*a))
                - 1/(4*sqrtz) * EXP((a^2*cx^2+2*sqrt2*b*cx+2*b^2*cy)/(4*z))
                * (1 - ERF((sqrt2*b + a^2*cx)/(2*a*sqrtz)))
                + 1/(4*sqrtz) * EXP((a^2*cx^2-2*sqrt2*b*cx+2*b^2*cy)/(4*z))
                * (ERF((sqrt2*q-sqrt2*a^2*cy*q - sqrt2*a*b*cy + a*cx)/(2*sqrtz))
                + ERF((-(a^2)*cx + sqrt2*b)/(2*a*sqrtz)))
            )
        )
    )
)
```

A.2 Probability of False Accept

For symmetric limits, define Probability of False Accept (`PFA`):

```
=LAMBDA(t,a,sigm,sigp,
    LET(sp, sigp,
        st, SQRT(sigm^2 + sigp^2),
        rho, SQRT(sigp^2 / (sigp^2+sigm^2)),
        2*(BIVARIATE_NORMCDF(a/st, -t/sp, rho) - BIVARIATE_NORMCDF(-a/st, -t/sp,
rho))
    )
)
```

where input `sigm` is the measurement uncertainty standard deviation, `sigp` the product distribution standard deviation, and `t` and `a` are the plus-minus tolerance and acceptance limits, respectively.

For asymmetric limits, including single-sided limits ($\mu_p \neq 0$, or $T_U = \infty$ or $T_L = -\infty$), define `PFA_ASYMMETRIC`:

```
=LAMBDA(tl,tu,al,au,mup,sigm,sigp,
    LET(sp, sigp,
        st, SQRT(sigm^2 + sigp^2),
        rho, SQRT(sigp^2 / (sigp^2+sigm^2)),
        ttl, IF(ISNUMBER(tl), tl, -1E+99),
        ttu, IF(ISNUMBER(tu), tu, 1E+99),
        aal, IF(ISNUMBER(al), al, ttl),
        aau, IF(ISNUMBER(au), au, ttu),
        pfaleft, BIVARIATE_NORMCDF((aau-mup)/st, (ttl-mup)/sp, rho) -
BIVARIATE_NORMCDF((aal-mup)/st, (ttl-mup)/sp, rho),
        pfarght, BIVARIATE_NORMCDF((-aal+mup)/st, (-ttu+mup)/sp, rho) -
BIVARIATE_NORMCDF((-aau+mup)/st, (-ttu+mup)/sp, rho),
    pfaleft+pfarght
    )
)
```

with `mup` the mean of the product distribution, `tl`, `tu` the lower and upper tolerance limits, and `al`, `au` the lower and upper acceptance limits.

A.3 Probability of False Reject

Probability of False Reject (PFR) is calculated by integrating over the two false reject regions. For symmetric limits:

$$PFR = 2\left[\Phi\left(\frac{-A}{\sigma_2}, \frac{T}{\sigma_1}, \rho\right) - \Phi\left(\frac{-A}{\sigma_2}, \frac{-T}{\sigma_1}, \rho\right)\right] \qquad (25)$$

And for asymmetric limits:

$$PFR_B = \left[\Phi\left(\frac{A_L - \mu_p}{\sigma_2}, \frac{T_U - \mu_p}{\sigma_1}, \rho\right) - \Phi\left(\frac{A_L - \mu_p}{\sigma_2}, \frac{T_L - \mu_p}{\sigma_1}, \rho\right)\right] \qquad (26)$$

$$PFR_T = \left[\Phi\left(\frac{-A_U+\mu_p}{\sigma_2}, \frac{-T_L+\mu_p}{\sigma_1}, \rho\right) - \Phi\left(\frac{-A_U+\mu_p}{\sigma_2}, \frac{-T_U+\mu_p}{\sigma_1}, \rho\right)\right] \qquad (27)$$

$$PFR = PFR_B + PFR_T \qquad (28)$$

A.3.1 PFR LAMBDA Function

Functions for PFR are similar to PFA, limits of integration modified to cover the appropriate regions.

Define `PFR`:

```
=LAMBDA(t,a,sigm,sigp,
    LET(sp, sigp,
        st, SQRT(sigm^2 + sigp^2),
        rho, SQRT(sigp^2 / (sigp^2+sigm^2)),
        2*(BIVARIATE_NORMCDF(-a/st, t/sp, rho) - BIVARIATE_NORMCDF(-a/st, -t/sp,
rho))
    )
)
```

and `PFR_ASYMMETRIC`:

```
=LAMBDA(tl,tu,al,au,mup,sigm,sigp,
    LET(sp, sigp,
        st, SQRT(sigm^2 + sigp^2),
        rho, SQRT(sigp^2 / (sigp^2+sigm^2)),
        ttl, IF(ISNUMBER(tl), tl, -1E+99),
        ttu, IF(ISNUMBER(tu), tu, 1E+99),
        aal, IF(ISNUMBER(al), al, ttl),
        aau, IF(ISNUMBER(au), au, ttu),
        pfrbot, BIVARIATE_NORMCDF((aal-mup)/st, (ttu-mup)/sp, rho) -
BIVARIATE_NORMCDF((aal-mup)/st, (ttl-mup)/sp, rho),
        pfrtop, BIVARIATE_NORMCDF((-aau+mup)/st, (-ttl+mup)/sp, rho) -
BIVARIATE_NORMCDF((-aau+mup)/st, (-ttu+mup)/sp, rho),
        pfrtop+pfrbot
    )
)
```

A.4 Solving for Guardband

A bracketing solver is used to find the guardbanded limits *A* that result in a desired PFA. The `SOLVE_NR` function from Vertex42 [17] (MIT Licensed) provides a LAMBDA function solution:

```
=LAMBDA(function,xstart,[y],[xstep],[epsilon],[iterations],[info],
    LET(doc,"https://www.vertex42.com/lambda/newton-raphson.html", version,"1.0.0",
        n,ROWS(xstart),
        y,IF(ISOMITTED(y),0,y),
        xstep,IF(ISOMITTED(xstep),0.1,xstep),
        epsilon,IF(ISOMITTED(epsilon),0.00000000000001,epsilon),
        iterations,IF(ISOMITTED(iterations),20,iterations),
```

```
        ret,REDUCE({"x","Fx","Iterations"},SEQUENCE(n),LAMBDA(acc_row,i,
            VSTACK(acc_row,
            REDUCE(
                HSTACK(INDEX(xstart,i),function(INDEX(xstart,i))-y,0),
                SEQUENCE(iterations),
                LAMBDA(acc,k,
                LET(cur_row,TAKE(acc,-1),
                    IF(ABS(INDEX(cur_row,1,2))<ABS(epsilon),acc,
                    LET(xn,INDEX(cur_row,1,1),
                        fn,INDEX(cur_row,1,2),
                        fprimen,(function(xn+xstep)-y-fn)/xstep,
                        xnew,xn-fn/fprimen,
                        fnew,function(xnew)-y,
                        new_row,HSTACK(xnew,fnew,k),
                        output,IF(info="verbose",VSTACK(acc,new_row),new_row),
                        output
                    )
                )
            ))
         ))
    )),
    IF(OR(info=TRUE,info="verbose"),ret,DROP(ret,1,-2))
))
```

To calculate the guardband to achieve the input `pfaval`,

```
=LAMBDA(tl,tu,mup,sigm,sigp,pfaval,
    IFS(AND(ISNUMBER(tl), ISNUMBER(tu)), LET(
        func, LAMBDA(x, PFA(tl,tu,tl+x,tu-x,mup,sigm,sigp)),
        SOLVE_NR(func, sigm/10, pfaval, sigm/100, sigm/1000, 25)
        ),
    ISNUMBER(tl), LET(
        func, LAMBDA(x, PFA_ASYMMETRIC(NA(),tu,NA(),tu-x,mup,sigm,sigp)),
        SOLVE_NR(func, sigm/10, pfaval, sigm/100, sigm/1000, 10)
        ),
    ISNUMBER(tu), LET(
        func, LAMBDA(x, PFA_ASYMMETRIC(tl,NA(),tl+x,NA(),mup,sigm,sigp)),
        SOLVE_NR(func, sigm/10, pfaval, sigm/100, sigm/1000, 10)
        )
    )
)
```