

Precision Arithmetic: A New Floating-Point Arithmetic

Chengpu Wang Independent researcher 631-974-1078 Chengpu@gmail.com

1 Abstract

A new floating-point arithmetic called precision arithmetic is developed to track precision for arithmetic calculations. It uses a novel rounding scheme to avoid excessive rounding error propagation of conventional floating-point arithmetic. Unlike interval arithmetic, its uncertainty tracking is based on statistics and its bounding range is much tighter. Its stable rounding error distribution is approximated by truncated normal distribution. Generic standards and systematic methods for validating uncertainty-bearing arithmetics are discussed. The precision arithmetic is found to be better than interval arithmetic in uncertainty-tracking for linear algorithms.

The code of precision arithmetic is published at <http://precisionarithm.sourceforge.net>.

Keywords: Computer arithmetic, Error analysis, Interval arithmetic, Multi-precision arithmetic, Numerical algorithms.

2 Introduction

2.1 Measurement Precision

Except simplest counting, scientific and engineering measurement never gives completely precise results [1][2]. The precisions of measured values range from order-of-magnitude estimation of astronomical measurements, to 10^{-2} ~ 10^{-4} of common measurements, to 10^{-14} of state-of-art measurements of basic physics constants [3].

In scientific and engineering measurement the uncertainty of a measurement x is usually characterized by sample deviation δx [1][2][4], though in certain cases such as raw reading from an analog-to-digital converter (ADC) the uncertainty of a measurement x is given as bounding

range Δx ¹ [5]. If $[x-\Delta x, x+\Delta x]$ crosses 0, x is neither positive nor negative for certainty due to following two possibilities:

- Either Δx is too large to give a precise measurement of x ,
- Or x itself is a measurement of zero.

To distinguish which case it is, additional information is required so that the measurement $x \pm \Delta x$ itself is not significant if $[x-\Delta x, x+\Delta x]$ crosses 0. An insignificant value also has conceptual difficulty to participate in many mathematical operations, such as square root or acting as divisor.

$P \equiv \delta x/|x|$ is defined here as the (relative) precision of the measurement, and its inverse is commonly known as the significance² [1][2]. Precision represents reliable information content of a measurement. Finer precision means higher reliability thus better reproducibility of the measurement [1][2]. Precision in this paper does not mean the maximal significand bit count as in term “variable precision arithmetic”. It takes the traditional definition in measurement [1][2].

2.2 Problem of the Conventional Floating-point Arithmetic

The conventional floating-point arithmetic [6][7][8] assumes a constant and best-possible precision for each value all the time, and constantly generates artificial information during calculation [9]. For example, the following calculation is carried out precisely in integer format:

$$\begin{aligned} \text{Equation 2-1: } & (64919121 \times 205117922 - 159018721 \times 83739041) \\ & = 13316075197586562 - 13316075197586561 = 1; \end{aligned}$$

If calculation in Equation 2-1 is carried out using conventional floating-point arithmetic:

¹ x is normally an integer at output of ADC. In ideal case, Δx equals half bit of ADC. Δx can be larger if the settle time is not long enough or if the ADC is not ideal.

² To avoid confusion, significance is used instead of precision in this paper, because “large precision” or “high precision” in common language actually means that the value of precision is very small.

Precision Arithmetic: A New Floating-Point Arithmetic

$$\begin{aligned}\text{Equation 2-2: } & (64919121 \times 205117922 - 159018721 \times 83739041) \\ & = 13316075197586562 - 13316075197586560 = 2.000000000000000;\end{aligned}$$

1. The multiplication results exceed the maximal significance of the 64-bit IEEE floating-point representation, so they are rounded off, generating rounding errors;
2. The normalization of the subtraction result amplifies the rounding error to most significant bit (MSB) by padding zeros.

Equation 2-2 is a show case for the problem of conventional floating-point arithmetic. Because normalization happens after each arithmetic operation [6][7][8], such generation of rounding errors happens very frequently for addition and multiplication, and such amplification of rounding errors happens very frequently for subtraction and division. The accumulation of rounding errors is an intrinsic problem of conventional floating-point arithmetic [10], and in majority of cases such accumulation is almost invisible [9]. For example, even most input data have only $10^{-2} \sim 10^{-4}$ of precision, except for simplest calculations, the 32-bit IEEE floating-point format [6][7][8] which has constant 10^{-7} precision, is usually not fine enough, because the rounding error from lower digits quickly propagates to higher digits.

Self-censored rules are developed to avoid such rounding error propagation [10][11], such as avoiding subtracting results of large multiplication, as in Equation 2-1. However, these rules are not enforceable, and difficult to follow in many cases, e.g., even most carefully crafted algorithm can result in numerical instability. Because the propagation speed of rounding error depends on nature of calculation itself, e.g., generally faster in nonlinear algorithms than linear algorithms [12], propagation of rounding error in conventional floating-point arithmetic is very difficult to quantify generically [13] so it is difficult to tell if a calculation is improper or becomes excessive for a required result precision. In common practice, reasoning on individual theoretical base is used to estimate the error and validity of calculation results, such from the estimated transfer functions of the algorithms used in the calculation [10][13][15], but such analysis is rare, and generally very difficult to carry out in practice.

Today, most experimental data are collected by ADC [5]. The result from ADC is an integer with a fixed uncertainty, thus a smaller signal value has a coarser precision. When a waveform containing raw digitalized signals from ADC is converted into conventional floating-point representation, the information content of the digitalized waveform is distorted to favor small signals since all converted data now have same and best-possible precision. The effects of such distortion in signal processing are generally not clear.

So what needed is a floating-point arithmetic that tracks precision automatically. When the calculation is improper or becomes excessive, the results become insignificant. All existing uncertainty-bearing arithmetic is reviewed here.

2.3 Interval Arithmetic

Interval arithmetic [11][16][17] is currently a standard method to track calculation uncertainty. It assumes that a value x is absolutely bounded within its bounding range $[x\downarrow, x\uparrow]$, using the following arithmetic equations [11]:

$$\text{Equation 2-3: } [x_1\downarrow, x_1\uparrow] + [x_2\downarrow, x_2\uparrow] = [x_1\downarrow+x_2\downarrow, x_1\uparrow+x_2\uparrow];$$

$$\text{Equation 2-4: } [x_1\downarrow, x_1\uparrow] - [x_2\downarrow, x_2\uparrow] = [x_1\downarrow-x_2\uparrow, x_1\uparrow-x_2\downarrow];$$

$$\text{Equation 2-5: } [x_1\downarrow, x_1\uparrow] * [x_2\downarrow, x_2\uparrow] \equiv [x\downarrow, x\uparrow];$$

$$x\downarrow = \min(x_1\downarrow * x_2\downarrow, x_1\uparrow * x_2\downarrow, x_1\downarrow * x_2\uparrow, x_1\uparrow * x_2\uparrow);$$

$$x\uparrow = \max(x_1\downarrow * x_2\downarrow, x_1\uparrow * x_2\downarrow, x_1\downarrow * x_2\uparrow, x_1\uparrow * x_2\uparrow);$$

$$\text{Equation 2-6: } [x_1\downarrow, x_1\uparrow] / [x_2\downarrow, x_2\uparrow] \equiv [x\downarrow, x\uparrow];$$

$$x\downarrow = \min(x_1\downarrow / x_2\downarrow, x_1\uparrow / x_2\downarrow, x_1\downarrow / x_2\uparrow, x_1\uparrow / x_2\uparrow);$$

$$x\uparrow = \max(x_1\downarrow / x_2\downarrow, x_1\uparrow / x_2\downarrow, x_1\downarrow / x_2\uparrow, x_1\uparrow / x_2\uparrow);$$

When multiplying x with itself, the result $x\downarrow$ is floored at 0 so that $x\downarrow \geq 0$.

If it is implemented using a floating-point representation with limited resolution, the result bounding range is widened further [17].

A basic problem is that the bounding range used by interval arithmetic is not compatible with usual scientific and engineering measurements, which instead use statistical mean and deviations to characterize uncertainty. Most measured values are well approximated by normal distribution [1][2][4], which has no limited bounding range. Let bounding leakage be defined as the possibility of the true value to be outside a bounding range. If a bounding range is defined using a statistical rule on bounding leakage, such as the 6σ - 10^{-9} rule for normal distribution [4] (which says the bounding leakage is about 10^{-9} for a bounding range of mean ± 6 -fold of standard deviations), there is no guarantee that the calculation result will also obey the 6σ - 10^{-9} rule using interval arithmetic, since interval arithmetic has no statistical ground.

Another problem is that interval arithmetic only gives worst case of uncertainty propagation and thus tends to over-estimate uncertainty in reality. For example, in addition and subtraction, it gives the result when the two operands are +1 and -1 correlated respectively [19]. However, if the two operands are -1 and +1 correlated respectively instead, the actually bounding range after addition and subtraction reduces, which is called the best-case interval in random interval arithmetic [18]. The vast overestimation of bounding ranges in these two cases prompts the developments of affine arithmetic [19][20], which traces error sources using a first-order model. Because of its expense in execution, and dependence on approximate modeling for operations even as basic as multiplication and division, affine arithmetic has not been widely used. In another approach, random interval arithmetic [18] reduces the uncertainty over-estimation of standard interval arithmetic by randomly choosing between the best-case and the worst-case intervals.

A third problem is that the result of interval arithmetic may depend strongly on the actual expression of an analytic function $f(x)$. For an example, Equation 2-7, Equation 2-8 and Equation 2-9 are different expressions of a same $f(x)$, but only through Equation 2-7 the result is correct, and uncertainty may be exaggerated in the other two forms, e.g., by 67-fold at input

range [0.44, 0.56] using Equation 2-8. This is called the dependence problem of interval arithmetic [21]. A practical implementation of interval arithmetic [21] needs to be free from the dependence problem thus much more complicated than using Equation 2-3, Equation 2-4, Equation 2-5 and Equation 2-6 directly.

Equation 2-7: $f(x) = (x - 1/2)^2 - 1/4;$

Equation 2-8: $f(x) = x^2 - x;$

Equation 2-9: $f(x) = (x - 1) x;$

Interval arithmetic has very coarse and algorithm-specific precision but constant zero bounding leakage. It represents the other extreme from conventional floating-point arithmetic. To meet with practice needs, a better uncertainty-bearing arithmetic should be based on statistical propagation of rounding error and it should allow reasonable bounding leakage for normal usages.

2.4 Statistical Propagation of Uncertainty

If each operand is regarded as a random variable, and the statistical correlation between the two operands is known, the result uncertainty is given by statistical propagation of uncertainty [22], with the following arithmetic equations, in which σ is the deviation of a measured value x , P is its precision, and γ is correlation between the two operands x_1 and x_2 :

Equation 2-10: $(x_1 \pm \sigma_1) + (x_2 \pm \sigma_2) = (x_1 + x_2) \pm \sqrt{\sigma_1^2 + \sigma_2^2 + 2\sigma_1\sigma_2\gamma};$

Equation 2-11: $(x_1 \pm \sigma_1) - (x_2 \pm \sigma_2) = (x_1 - x_2) \pm \sqrt{\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\gamma};$

Equation 2-12: $(x_1 \pm \sigma_1) * (x_2 \pm \sigma_2) = (x_1 * x_2) \pm |x_1 * x_2| \sqrt{P_1^2 + P_2^2 + 2P_1P_2\gamma};$

Equation 2-13: $\frac{x_1 \pm \sigma_1}{x_2 \pm \sigma_2} = \frac{x_1}{x_2} \pm \left| \frac{x_1}{x_2} \right| \sqrt{P_1^2 + P_2^2 - 2P_1P_2\gamma};$

Tracking uncertainty propagation statistically seems an ideal solution. However in practice the correlation between two operands is generally not precisely known, so direct use of

statistical propagation of uncertainty is very limited. In this paper, as a proxy for statistical propagation of uncertainty, an independence arithmetic always assumes no correlation between any two operands, whose arithmetic equations are Equation 2-10, Equation 2-11, Equation 2-12 and Equation 2-13 but with $\gamma=0$. Independence arithmetic is actually de facto arithmetic in engineering data processing [1][2], such as in the common believe that uncertainty after averaging reduces by the square root of number of measurements.

2.5 Significance Arithmetic

Significance arithmetic [23] tries to track reliable bits in an imprecise value during calculation. Except early attempts [24][25], significance arithmetic has not yet been implemented digitally. In these attempts, the implementations of significance arithmetic are based on simple operating rules upon reliable bit counts, rather than on formal statistical approaches. They both treat the reliable bit counts as integers when applying their rules, while in reality a reliable bit count could be a fractional number, so they both can cause artificially quantum reduction of significance. Significance arithmetic is not widely practiced in scientific and engineering calculations [23].

Also can be categorized as significance arithmetic, a stochastic arithmetic [13] randomizes least significant bits (LSB) of each of input floating-point values, repeats a same calculation multiple times, and then uses statistics to seek invariant digits among the calculation results as significant digits. This approach may require too much calculation since the number of necessary repeats for each input is specific to each algorithm, especially when the algorithm contains branches. It is based on modeling of rounding errors in conventional floating-point arithmetic, which is quite complicated. A better approach may be to define arithmetic rules that make error tracking by probability easier.

2.6 An Overview of This Paper

In this paper, a new floating-point arithmetic called precision arithmetic [26] is developed to track uncertainty during floating-point calculations, as described in Section 3. Generic standards and systematic methods for validating uncertainty-bearing arithmetics are discussed in Section 4. The precision arithmetic is compared with other uncertainty-bearing arithmetics in Section 5 to Section 10. A brief discussion is provided in Section 11.

3 *The Precision Arithmetic*

3.1 Basic Assumptions for Precision Arithmetic

Precision arithmetic tracks uncertainty distribution during calculation using specially designed arithmetic rules. It has the independent uncertainty assumption as its basic assumption, which assumes that the uncertainties of any two different identities can be regarded as independent of each other. This assumption can be turned into a realistic statistical requirement on input data to precision arithmetic.

Because it is not realistic to track the actual uncertainty distributions which may vary according to each specific algorithm, the objectives of precision arithmetic is to enclose the actual uncertainty distribution with a bounding distribution with the following two goals, which needs only be approximately satisfied:

- The bounding distribution is symmetric around an expected value which is the value given by mathematics when there is no uncertainty.
- Outside the range of mean \pm deviation of the bounding distribution, the probability density function of the bounding distribution is generally larger than that of the actual uncertainty distribution. Precision arithmetic may occasionally extend uncertainty range of the bounding distribution at one or both sides to keep this goal valid.

In addition, precision arithmetic heavily uses the scaling principle, which says that the result precision should not change when an imprecise value is either multiplied or divided with a non-zero constant. It is concluded from Equation 2-12 and Equation 2-13 for statistical propagation of uncertainty.

3.2 The Independent Uncertainty Assumption

When there is good estimation of source of uncertainty, the independent uncertainty assumption can be judged directly, e.g., if noise [1][2] is the major source of uncertainty, the independent uncertainty assumption is probably true. Repeated measurements of a same signal needs to be judged by this criterion. Otherwise, the independent uncertainty assumption can be judged by the correlation and respective precisions of two measurements.

Assume X , Y , and Z are three mutually independent random variables with variance $\sigma^2(X)$, $\sigma^2(Y)$ and $\sigma^2(Z)$ respectively. Let α be a constant, $\text{Cov}()$ be covariance function and γ be correlation between $(X + Y)$ and $(\alpha X + Y)$:

$$\text{Equation 3-1: } \gamma = \frac{\text{Cov}(X+Y, \alpha X+Z)}{\sqrt{\sigma^2(X+Y) \cdot \sigma^2(\alpha X+Z)}} = \frac{\alpha \sigma^2(X)}{\sqrt{\sigma^2(X) + \sigma^2(Y)} \sqrt{\alpha^2 \sigma^2(X) + \sigma^2(Z)}};$$

$$\text{Let } \eta_1^2 \equiv \frac{\sigma^2(Y)}{\sigma^2(X)} \text{ and } \eta_2^2 \equiv \frac{\sigma^2(Z)}{\sigma^2(\alpha X)} = \frac{\sigma^2(Z)}{\alpha^2 \sigma^2(X)}:$$

$$\text{Equation 3-2: } \gamma = \frac{\alpha / |\alpha|}{\sqrt{1 + \eta_1^2} \sqrt{1 + \eta_2^2}} \equiv \frac{\alpha / |\alpha|}{1 + \eta^2};$$

Equation 3-2 gives the correlation γ between two identities, each of which contains a completely uncorrelated part and a completely correlated part, with η being the average ratio between these two parts. Equation 3-2 can also be interpreted reversely: if two identities are correlated by γ , each of them can be viewed as containing a completely uncorrelated part and a completely correlated part, with η being the average ratio between these two parts.

One special application of Equation 3-2 is the correlation between a measured signal and its true signal in which noise is the uncorrelated part between the two. Figure 1 shows the effect of noise on most significant two bits of a 4-bit measured signal when $\eta=1/4$. Its top chart shows a triangular waveform between 0 and 16 in black line, and a white noise between -2 and +2 using grey area. The measured signal is the sum of the triangle waveform and the noise. The middle chart of Figure 1 shows the values of the 3rd digit of the true signal in black line, and the mean values of the 3rd bit of the measurement in grey line. The 3rd bit is affected by the noise during its transition between 0 and 1. For example, when the signal is slightly below 8, only a small positive noise can turn the 3rd digit from 0 to 1. The bottom chart of Figure 1 shows the values of the 2nd digit of the signal and the measurement in black line and grey line respectively. Figure 1 clearly shows that the correlation between the measurement and the true signal is less at the 2nd digit than at the 3rd digit. Quantitatively, according to Equation 3-2:

- The 3rd digit of the measurement is 94% correlated to the signal with $\eta=1/4$;
- The 2nd digit of the measurement is 80% correlated to the signal with $\eta=1/2$;
- The 1st digit of the measurement is 50% correlated to the signal with $\eta=1$;
- The 0th digit of the measurement is 20% correlated to the signal with $\eta=2$;

The above conclusion agrees with the common experiences that below noise level of measured signals, noises rather than true signals dominate each digit.

Similarly, while the correlated portion between two identities has exactly same value at each bit of the two identities, the ratio of uncorrelated portion to the correlated portion increases by 2-fold for each bit down from MSB of the two identities, regardless the nature of the uncorrelated portion. Quantitatively, assume ϕ is the smaller significance of the two identities, and η_ϕ is the ratio of uncorrelated portion to the correlated portion at level of uncertainty:

Equation 3-3: $\eta_\phi = \eta \cdot \phi$, $\phi > 1$;

According to Equation 3-2 and Equation 3-3, if two significant identities are overall correlated by γ , at level of uncertainty the correlation between the two identities decreases to γ_ϕ :

$$\text{Equation 3-4: } \left(\frac{1}{\gamma_\phi} - 1\right) = \left(\frac{1}{\gamma} - 1\right) \cdot \phi^2, \quad \phi > 1;$$

Figure 2 plots the relation of γ vs. ϕ for each given γ_ϕ in Equation 3-4. If when γ_ϕ is less than a maximal threshold (e.g., 2%, 5% or 10%), the two identities can be deemed virtually independent of each other at the level of uncertainty, then there is a maximal allowed correlation between any two identities for each significance ϕ , as shown in Figure 2. If two identities are independent of each other at their uncertainty levels, their uncertainties are independent of each other. For a given γ_ϕ requirement, below the maximal allowed correlation threshold, the independent uncertainty assumption is true for the two identities. Figure 2 shows that for two precisely measured identities, their correlation γ is allowed to be quite high. While to be acceptable in precision arithmetic, each of low-resolution identities should contain enough true noise in its uncertainty, so that they have not much correction through systematic error [1][2]. Thus, the independence uncertainty assumption of precision arithmetic is much weaker than requiring the two identities to be independent of each other, which is the assumption for independence arithmetic.

It is tempting to add true noise to otherwise unqualified identities to make their uncertainties independent of each other. As an extreme case of this approach, if two identities are constructed by adding true noise to a same signal, they are at least 50% correlated at uncertainty level and hence will not satisfy the independent uncertainty assumption. The 50% curve in Figure 2 thus defines maximal possible correlations between any two measured signals. This other conclusion of Equation 3-4 makes sense because the measurable correlation between two identities should be limited by the precisions of their measurements.

3.3 Precision Representation and Precision Round up Rule

Let the content of a floating-point number be denoted as $S@E$, in which S is the significand³ and E is the exponent of 2 of the floating-point number. In addition, precision representation contains a carry \sim to indicate its rounding error, which can be:

- $+$: The rounding error is positive;
- $-$: The rounding error is negative;
- $?$: The sign of rounding error is unknown;
- $\#$: The precision value contains an error code. Each error code is generated due to a specific illegal arithmetic operation such as dividing by zero. An operand error code is directly transferred to the operation result. In this way, illegal operations can be traced back to the source.

A round up happens when E is incremented by 1 according to the following round up rule:

- A value of $2S\sim@E$ is rounded up to $S\sim@E+1$.
- A value of $(2S+1)+@E$ is rounded up to $(S+1)-@E+1$.
- A value of $(2S+1)-@E$ is rounded up to $S+@E+1$.
- A value of $(2S+1)?@E$ is rounded up randomly to either $(S+1)-@E+1$ or $S+@E+1$ with equal probability.

Thus, unlike in conventional floating-point representation, precision arithmetic no longer has polymorphism [6][7][8] in its representation,.

After each round up, the original rounding error is reduced by half for the new significand. If the original significand is odd, the round up generates a new rounding error of $1/2$, which is

³ While “significand” is the official word [8] to describe “The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.”, “mantissa” is often unofficially used instead.

added to the existing rounding error. Since the newly generated rounding error always cancels the existing rounding error, the rounding error range is limited to half bit of significand for the precision round up rule.

If initial \sim is wrong, it will be reduced and corrected quickly during rounding up. So the precision round up process is stable and self-correcting.

The precision arithmetic also tracks rounding error bounding range R , so that the precision representation becomes $S \sim R @ E$. After each rounding up, R is reduced by half. If the LSB of significand is odd, R is incremented by $1/2$.

3.4 Precise Value

When R is zero, the precision value is precise, otherwise it is imprecise. A precise value has no rounding error. After a precise value with odd significand is rounded up once, it becomes an imprecise value of $S ? 1/2 @ E$.

An imprecise value can be decomposed as a precise value plus an imprecise zero:

$$\text{Equation 3-5: } S \sim R @ E = S @ E + 0 \sim R @ E;$$

In Equation 3-5, $S @ E$ carries mathematically expected value while $0 \sim R @ E$ carries uncertainty.

3.5 Probability Distribution of Rounding Errors

An ideal floating-point calculation is carried out to infinitesimal precision before it is rounded up to representation precision [8][11][13]. Thus, rounding up should be an independent process from any calculation, and be evaluated separately. To find out rounding error distribution within its bounding range $[-1/2, +1/2]$, a large count of positive random integers are converted into precision values and then rounded up until each of them has significand larger than a minimal significand threshold. Each rounded up precision value is compared with the original value for rounding error. Figure 3 shows the result histogram of rounding errors for the minimal significand thresholds 0, 1, 4 and 16 respectively. When each bit of significand has equal

chance to be either 0 or 1, the result distribution is expected to be uniformly distributed within range $(-1/2, +1/2)$ [27]. However, the precision round up rule changes such equal chance for few lowest digits of significand. So when the minimal significand threshold is smaller, the bias in rounding error distribution is larger, as shown in Figure 3, and the result distribution is close to uniform only when the minimal significand threshold is 4 and above.

3.6 Result Bounding For Addition and Subtraction

To a floating-point arithmetic, rounding errors are uncertainties [8][11][13]. The precision round up rule incorporates all randomness of an imprecise value into its carry and bounding range, so that it preserves the independent uncertainty assumption between any two identities. The independent uncertainty assumption suggests that the result error distribution of addition is the convolution of the two operand error distributions, while the result error distribution of subtraction is the convolution of the first operand error distribution and the mirror image of the second operand error distribution [4]. Thus, when the exponents of two operands are equal, the results of addition and subtraction are:

$$\text{Equation 3-6: } S_1 \sim R_1 @ E \pm S_2 \sim R_2 @ E = (S_1 \pm S_2) \sim (R_1 + R_2) @ E;$$

By assuming that both operands are immediate result of applying precision round up rule, Table 1 shows the result \sim for addition while Table 2 shows the result \sim for subtraction. It will be shown that the \sim immediately after a calculation is actually not important, because precision round up rule is frequently applied after each calculation in precision arithmetic as its normalization process.

Assume the rounding errors are uniformly distributed between $[-1/2, +1/2]$ with a density function $P_{1/2}(x)$:

$$\text{Equation 3-7: } P_{1/2}(x) \equiv 1, \quad -1/2 \leq x \leq +1/2;$$

Then the rounding error distribution density function $P_{n/2}(x)$ is obtained by convolution:

$$\text{Equation 3-8: } P_{n/2}(x) \equiv \int_{-\infty}^{+\infty} P_{1/2}(y) P_{(n-1)/2}(x-y) dy = \int_{-1/2}^{+1/2} P_{(n-1)/2}(x-y) dy, \quad n=2, 3, 4, \dots;$$

Equation 3-8 shows that $P_{n/2}(x)$ has a bounding range of $R \equiv n/2$. It is easy to prove that the deviation σ of $P_{n/2}(x)$ is determined by its bounding range R :

$$\text{Equation 3-9: } \sigma^2 = R/6;$$

Also, a same bounding range can be reached in any combination:

$$\text{Equation 3-10: } P_{(m+n)/2}(x) = \int_{-\infty}^{+\infty} P_{m/2}(y) P_{n/2}(x-y) dy ;$$

In reality, $P_{1/2}(x)$ is not strictly uniformly distributed in its bounding range $[-1/2, +1/2]$. As the worst case, let $P_{1/2}(x)$ be the error distribution with a minimal significand threshold of 0 in Figure 3, Figure 4 shows the rounding error distribution after addition and subtraction once and twice, in which:

- $R=1/2$: “1” for no addition and subtraction.
- $R=2/2$: “1+1” for addition once, and “1-1” for subtraction once.
- $R=3/2$: “1+1+1” for addition twice, “1-1-1” for subtraction twice, “1+1-1” for addition once then subtraction once, and “1-1+1” for subtraction once then addition once.

Figure 4 shows that rounding error distributions for same bounding range largely repeat each other, confirming Equation 3-10. Addition and subtraction have slightly different result distribution due to uneven $P_{1/2}(x)$. In all cases, the distributions quickly approach Gaussian with increase bounding ranges. Figure 5 shows that for all distributions in Figure 3, the deviation of the rounding error distributions increases with the bounding range according to a square root relation to a very high degree with a factor coefficient near $1/\sqrt{6}$, confirming Equation 3-9. Any mean of the distributions is always 10-fold smaller than its deviation thus can be ignored.

The probability density function $D_Y(y)$ after linear transformation ($y = \alpha x + \beta$) of a generic probability density function $D_X(x)$ is [4]:

Equation 3-11: $D_Y(y) = D_X((y - \beta)/\alpha)/\alpha;$

Let $N(x)$ be standard normal distribution. Since $P_{n/2}(x)$ is sum of $P_{1/2}(x)$ by n -times, according to central limit theorem [4], Equation 3-9 and Equation 3-11, $P_{n/2}(x)$ converges in distribution to a Gaussian distribution with mean 0 and deviation σ with increased n :

Equation 3-12: $P_{n/2}(y) \xrightarrow{d} N(y/\sigma)/\sigma, \quad y \in [-R, +R];$

3.7 Result Bounding for Rounding Up

According to Equation 3-11, when an imprecise value is rounded up once, both its original bounding range R and its original deviation σ are reduced to half for the new significand. According to Equation 3-9, when R is reduced to 1/2-fold, σ is reduced to $1/\sqrt{2}$ -fold, while when σ is reduced to 1/2-fold, R is reduced to 1/4-fold. The former is called round up by range, while the latter is called round up by deviation. Figure 6 compares these two ways of rounding up when the original error range is $R=8$, in which $R=4$ is rounded up by range, while $R=2$ is rounded up by deviation. It clearly shows that round up by deviation results in a much similar rounding error distribution. After rounded up, the stable distribution in Equation 3-12 becomes $N(y/\frac{\sigma}{2})/\frac{\sigma}{2}, y \in [-\frac{R}{2}, +\frac{R}{2}]$, which is better approximated as $N(y/\frac{\sigma}{2})/\frac{\sigma}{2}, y \in [-\frac{R}{4}, +\frac{R}{4}]$ when R is sufficiently large. Round up by deviation is also required by the scaling principle so it is used universally in precision arithmetic.

Round up by deviation also introduces bounding leakage called roundup leakage. In Figure 6, the $8/2$ distribution of rounding error outside the range $[-2, +2]$ contributes to a round-up leakage of 0.05%. Figure 7 shows that the round-up leakage decreases exponentially with increased bounding range R . Smaller round-up leakage means that the actual rounding error distribution is more similar to the rounding error distribution chosen by rounding up by deviation.

When R is above a threshold R_{\max} , round-up leakage is small enough, so that rounding up by deviation can be applied repeatedly. This is the normalization process in precision arithmetic.

An imprecise value $S \sim R @ E$ is in normalized format if its R is in the range of $[R_{\max}/4, R_{\max})$. Otherwise it is called a nearly precise value. According to Figure 7, when $R_{\max}=16$, the maximal normalization leakage is 10^{-6} , which is small enough for most applications. One function of normalization is to enforce the correctness of carry sign \sim in precision representation $S \sim R @ E$.

Because of normalization, $P_{n/2}(x)$ is extended to $P_R(x)$ for 2's fractional $R \in [1/2, R_{\max})$, so that the deviation δx and bounding range Δx of $S \sim R @ E$ is:

$$\text{Equation 3-13: } \delta x = \sigma 2^E, \quad \sigma < \sqrt{R_{\max}}/6;$$

$$\text{Equation 3-14: } \Delta x = R 2^E, \quad R < R_{\max};$$

$$\text{Equation 3-15: } \Delta x / \delta x = R / \sigma = \sqrt{6R};$$

The stable probability density function in Equation 3-12 becomes probability density function $\rho(y)$ in Equation 3-16.

$$\text{Equation 3-16: } \rho(y) = N(y/\delta x) / \delta x, \quad y \in [-\Delta x, \Delta x];$$

All characteristics of a distribution are decided by its moments [4]. A normal distribution decreases very fast at its two tails so that its tails do not contribute significantly to moment calculation [4][28]. Thus, according to Equation 3-15, when R is sufficiently large, $\rho(y)$ has identical moments of the corresponding normal distribution, and it is truly the normal distribution but with truncated range. For example, in Figure 7, bounding leakage actually indicates the difference of 0th moment from 1 with increased R . It is assumed that when $R \geq R_{\max}/4$, R is sufficiently large. Equation 3-5 can be reinterpreted as Equation 3-17, in which x is the mathematically expected value and y is a ρ -distributed random variable measuring uncertainty of x .

$$\text{Equation 3-17: } x \pm \delta x = x + y, \quad y \in \rho(y);$$

3.8 Precision Round Down Rule

As opposite operation for rounding up by deviation, the precision round down rule is defined using the scaling principle. After round down once, $S \sim R @ E$ becomes $2S \sim 4R @ E - 1$. Round-down reduces bounding leakage.

To add or subtract two operands with different exponents, the operand with larger exponent is first round down to the other exponent, and the result of addition or subtraction using Equation 3-6 is normalized afterward. If the operand with larger exponent is normalized, the result exponent is at least as large as the larger operand exponent.

Adding an operand to itself is same as multiplying the operand by 2. Subtracting an operand from itself results in precise 0.

3.9 Bounding Initiation

An integer S is initialized as a precise value $S @ 0$.

A conventional 64-bit floating-point value $S @ E$ is usually initialized as a nearly precise value $S ? 1/2 @ E$, because the IEEE floating-point standard [8] guarantees accuracy to half bit of significand.

A mean-deviation pair $(x \pm \delta x)$ of 64-bit conventional floating-point values is initialized as an imprecise value $S \sim R @ E$ by (1) rounding up δx until Equation 3-13 is satisfied; (2) obtaining R and E from final δx ; (3) rounding up x to E ; (4) obtaining S and \sim from final x . Immediately after initialization, \sim is accurate due to rounding up. If the precision of the measured value is worse than 10^{-16} , the result value is normalized. Practically all measured values are normalized.

Precision arithmetic has limited calculation inside uncertainty, and the resolution of its significand is determined by its precision. In addition to other characteristics, Table 3 shows bits calculation inside uncertainty for different R_{\max} using precision arithmetic. Although a larger R_{\max} has smaller normalization leakage, it achieves this by having larger bounding range, thus it greatly increase the chance for a value to be insignificant. According to Table 3, only when

$R_{\max}=16$ the precision arithmetic has a comparable bounding range as de facto $6\sigma\cdot 10^{-9}$ rule. Thus, $R_{\max}=16$ is used in this paper. While calculating many bits inside uncertainty does not seem meaningful, not calculating at all inside uncertainty may not be an optimal approach either. Thus, Equation 3-13 is modified as Equation 3-18, in which χ is a small constant positive integer, to introduce χ -bit calculation inside uncertainty by interpreting the precision for $S\sim R@E$.

Equation 3-18: $\delta x = \sqrt{R/6 \cdot 2^{-\chi}} 2^{E+\chi};$

Table 4 shows examples of precision arithmetic with different χ for $R_{\max}=16$. χ will be found empirically to be 2 later in this paper.

Limited resolution of significands does not necessarily mean that precision arithmetic has larger calculation error. The following example shows that symmetry of precision representation cancels out rounding errors:

- $1/3 \pm 0.001 + 2/3 \pm 0.001 = 341 + 6.29@-10 + 683 - 6.29@-10 = 1024 \pm 12.6@-12 = 1 \pm 0.0014;$

In the above equation, the result expected value is precise 1 even the expected values of the two operands for addition are not precise $1/3$ and $2/3$. It can be proved that such cancellation of rounding errors due to symmetry is a generic property of the precision round up rule in calculating rational values.

3.10 Result Bounding For Multiplication

After $S\sim R@E$ is multiplied by 2, both its range and deviation increase by 2-fold. If the scaling principle is applied, the result is $2S\sim 4R@E$. When $S\sim R@E$ is normalized, the result is then normalized as $S\sim R@E+1$, and there is neither bound widening nor bounding leakage.

Generally, the direct result of multiplying $S_1\sim_1R_1@E_1$ by $S_2@E_2$ is:

Equation 3-19: $S_1\sim_1R_1@E_1 \times S_2@E_2 = S_1S_2\sim_1S_2^2R_1@ (E_1+E_2);$

Equation 3-19 obeys scaling principle, and the result uncertainty is still ρ -distributed.

According to independent uncertainty assumption, the product bounding range of multiplying $0 \sim_1 R_1 @ E_1$ by $0 \sim_2 R_2 @ E_2$ is $R_1 R_2 @ (E_1 + E_2)$. Thus:

$$\text{Equation 3-20: } S_1 \sim_1 R_1 @ E_1 \times S_2 \sim_2 R_2 @ E_2 = (S_1 S_2) @ (E_1 + E_2) + \\ 0 \sim_1 R_1 S_2^2 @ (E_1 + E_2) + 0 \sim_2 R_2 S_1^2 @ (E_1 + E_2) + 0 (\sim_1 \sim_2) R_1 R_2 @ (E_1 + E_2);$$

In Equation 3-20, $0 \sim_1 R_1 S_2^2 @ (E_1 + E_2)$, $0 \sim_2 R_2 S_1^2 @ (E_1 + E_2)$ and $0 (\sim_1 \sim_2) R_1 R_2 @ (E_1 + E_2)$ are still ρ -distributed, and the addition further converges the result uncertainty toward ρ -distributed.

The result precision P of multiplication is:

$$\text{Equation 3-21: } P^2 = \frac{1}{6} \frac{(S_2^2 R_1 + S_1^2 R_2 + R_1 R_2)}{(S_1 S_2)^2} = P_1^2 + P_2^2 + \frac{1}{6} P_1^2 P_2^2;$$

Equation 3-21 shows that precision cannot be improved during multiplication. It is identical to Equation 2-12 except their cross term, representing difference in their statistical requirements.

A special case of multiplication is between an operand and itself. If x is $N(x)$ distributed, x^2 is χ^2 -distributed with freedom 1 [4], which has mean 1 and variance of 2. $N(x)$ and $\chi^2(x)$ have quite different characteristics, e.g., $\chi^2(x)$ only roughly resembles half of $N(x)$. According to Equation 3-11, the square of $0 \pm \delta x$ has mean of δx and variance of $2 (\delta x)^4$. The bounding goal of precision arithmetic extends x^2 distribution to the other side of the mathematically expected value, and absorb the square of mean $(\delta x)^2$ into final variance:

$$\text{Equation 3-22: } (S \sim R @ E)^2 \equiv S^2 @ 2E + 0 \sim 4 R S^2 @ 2E + 0 \sim 3 R^2 @ 2E;$$

3.11 Result Bounding For Division

The reverse of Equation 3-19 defines:

$$\text{Equation 3-23: } S_1 \sim_1 R_1 @ E_1 / S_2 @ E_2 = (S_1 / S_2) \sim_1 (R_1 / S_2^2) @ (E_1 - E_2);$$

In Equation 3-23, the rounding error decreases by S_2 -fold, but the bounding range decreases by S_2^2 -fold, so there is bounding leakage. To limit the bounding leakage to acceptable level, the result is rounded down until normalized. In this case, rounding down the direct result is equivalent to round down the dividend $S_1 \sim R_1 @ E_1$ by the same amount before the division.

The scaling principle is used to define the result of dividing $S_1 @ E_1$ by $S_2 \pm R_2 @ E_2$:

$$\text{Equation 3-24: } S_1 @ E_1 / S_2 \sim_2 R_2 @ E_2 \equiv (S_1 / S_2) (-\sim_2) (R_2 S_1^2 / S_2^4) @ (E_1 - E_2) ;$$

The result also satisfies the reciprocal relation—when $S_1 @ E_1$ is divided by it, the result is exactly $S_2 \sim_2 R_2 @ E_2$. To limit the bounding leakage to acceptable level, the result is rounded down until normalized.

The direct result of division between two operands $S_1 \sim_1 R_1 @ E_1$ and $S_2 \sim_2 R_2 @ E_2$ is calculated as $(S_1 \sim_1 R_1 @ E_1) \times 1 / (S_2 \sim_2 R_2 @ E_2)$:

$$\text{Equation 3-25: } S_1 \sim_1 R_1 @ E_1 / S_2 \sim_2 R_2 @ E_2 = (S_1 / S_2) @ (E_1 - E_2) + \\ 0 \sim_1 R_1 / S_2^2 @ (E_1 - E_2) - 0 \sim_2 R_2 S_1^2 / S_2^4 @ (E_1 - E_2) - 0 (\sim_1 \sim_2) R_1 R_2 / S_2^4 @ (E_1 - E_2) ;$$

The result precision P of division is:

$$\text{Equation 3-26: } P^2 = \frac{1}{6} \frac{(R_1 / S_2^2 + R_2 S_1^2 / S_2^4 + R_1 R_2 / S_2^4)}{(S_1 / S_2)^2} = P_1^2 + P_2^2 + \frac{1}{6} P_1^2 P_2^2 ;$$

Equation 3-26 is identical to Equation 3-21, because $1 / (S_2 \sim_2 R_2 @ E_2)$ has same precision as $S_2 \sim_2 R_2 @ E_2$.

Dividing an operand by itself results in precise 1.

3.12 Function Evaluation

The uncertainty of function $f(x)$ at $x \pm \delta x$ is evaluated by set $\{f(x+y)-f(x)\}$, in which y is a random variable defined by Equation 3-16. Because the bounding goal of precision arithmetic centers on mathematically expected value, the mean of the set is assumed to be zero, and the variance $(\delta f)^2$ of the set is calculated as:

$$\text{Equation 3-27: } (\delta f)^2 \equiv \int (f(x+y) - f(x))^2 \rho(y) dy ;$$

An analytical expression of $(\delta f)^2$ solely in term of x^2 and $(\delta x)^2$ may be turned into an arithmetic operation executed directly in precision representation.

Denote $M(n)$ as n th moment of y . If y is symmetrically distributed around $y=0$, then all odd momentum of y is zero, as described by Equation 3-28, in which j is a natural number.

Equation 3-28: $M(2j+1) = 0;$

If y is normal distributed, then its non-zero momentums are given by Equation 3-29:

Equation 3-29: $M(2j+2) = \prod_{k=0}^j (2k+1);$

If function $f(x)$ is Taylor expandable at x , $f(x+y)-f(x)$ is calculated according to Equation 3-30, in which $f^{(n)}(x)$ is n th deviation of $f(x)$ at x . Using Equation 3-30, the definition of uncertainty of $f(x)$ at x can be extended to the cases when range $[x-\Delta x, x+\Delta x]$ is partially outside the definition range of $f(x)$. $(\delta f)^2$ is given by Equation 3-31.

Equation 3-30: $\Delta f \equiv f(x+y) - f(x) = \sum_{n=1}^{\infty} \frac{f^{(n)}(x)}{n!} y^n;$

Equation 3-31: $(\delta f)^2 = \sum_{n=1}^{\infty} \sum_{j=1}^{\infty} \frac{f^{(n)}(x)}{n!} \frac{f^{(j)}(x)}{j!} (\delta x)^{n+j} M(n+j);$

According to Equation 3-31:

- $f(x) = \alpha x$: $(\delta f)^2 = \alpha^2 (\delta x)^2$, confirming Equation 3-19.
- $f(x) = x^2$: $(\delta f)^2 = 3 (\delta x)^4 + 4 x^2 (\delta x)^2$, confirming Equation 3-22.
- $f(x) = \sqrt{x}$: When $x \pm \delta x$ is significant, $|x| > R$, $|x| \gg \delta x$, $(\delta f)^2 \approx (\delta x)^2 / (4|x|)$, which is reverse to the result for $f(x) = x^2$ when $|x| \gg \delta x$.
- $f(x) = 1/x$: When $x \pm \delta x$ is significant, $(\delta f)^2 \approx (\delta x)^2 / x^4$, confirming Equation 3-24.

Taylor expansion can also be used to find the result $(\delta f)^2$ of function $f(x_1, x_2)$, in which $f^{(m,n)}(x_1, x_2)$ is the m th and n th partial deviation of x_1 and x_2 respectively, and the independent uncertainty assumption between x_1 and x_2 leads to independence between random variables y_1 and y_2 in Equation 3-32:

Equation 3-32: $\Delta f \equiv f(x_1 + y_1, x_2 + y_2) - f(x_1, x_2) = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} \frac{f^{(m,n)}(x_1, x_2)}{m!n!} y_1^m y_2^n;$

Equation 3-33:
$$(\delta f)^2 = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \frac{f^{(m,n)}(x_1, x_2)}{m!n!} \frac{f^{(i,j)}(x_1, x_2)}{i!j!} (\delta x_1)^{m+i} (\delta x_2)^{n+j} M(m+i)M(n+j);$$

Such approach can be extended to a function of arbitrary number of input variables. According to Equation 3-33:

- $f(x_1, x_2) = x_1 \pm x_2$: $(\delta f)^2 = (\delta x_1)^2 + (\delta x_2)^2$, confirming Equation 3-6.
- $f(x_1, x_2) = x_1 x_2$: $(\delta f)^2 = x_2^2 (\delta x_1)^2 + x_1^2 (\delta x_2)^2 + (\delta x_1)^2 (\delta x_2)^2$, confirming Equation 3-16.
- $f(x_1, x_2) = x_1 / x_2$: $(\delta f)^2 \approx (\delta x_1)^2 / x_2^2 + (\delta x_2)^2 x_1^2 / x_2^4 + (\delta x_1)^2 (\delta x_2)^2 / x_2^4$, confirming Equation 3-25.

If $f(x)$ is a black-box function, because $N(x)$ is well-known, standard methods exist to divide range $[x-\Delta x, x+\Delta x]$ into equal-probability quantiles [4], and δf can be found numerically by sampling. For example, a κ -point monotonic sampling requires that (1) each numerically monotonic region contains at least κ consecutive sampling points; and (2) the whole range $[x-\Delta x, x+\Delta x]$ has been divided into monotonic regions only. When κ is sufficiently large, the chance of missing peak and valley is small enough, so that the sampling is fair enough. Range $[x-\Delta x, x+\Delta x]$ is first divided into κ equal-probability quantiles, and at each additional step each quantile is further divided into equal number of sub quantiles, until both sampling requirements are met. Then $(\delta f)^2$ is calculated as sample variance plus the square of sample mean.

3.13 Dependence Problem

Equation 3-31 accurately accounts for all correlation within $f(x)$, and it provides a clean solution for δf , e.g., it gives the same result for Equation 2-7, Equation 2-8 and Equation 2-9. It is tempting to applying basic arithmetic operations to $f(x)$ directly. However, such approach may apply the independent uncertainty assumption wrongly between an identity and its constant mathematical expression, such as between x and x^2 , so that it may result in dependence

problem similar to that of interval arithmetic [21]. For example, same as using interval arithmetic, only Equation 2-7 gives correct result for such use of precision arithmetic.

Conventional arithmetic has no uncertainty and it can decompose an analytic expression $f(x_1, x_2, \dots)$ into many simpler form $e(g(x_1, x_2, \dots), h(x_1, x_2, \dots))$, until each simpler form contains a constant mathematical expression of one identity. Equation 2-7, Equation 2-8 and Equation 2-9 are three examples of different decompositions of a same function $f(x)$. The flexibility of such decomposition makes conventional arithmetic very powerful. Using precision arithmetic, the uncertainty independence between x_1 and x_2 does not necessarily mean the uncertainty independence between $g(x_1, x_2)$ and $h(x_1, x_2)$. Thus, if $g(x_1, x_2, \dots)$ and $h(x_1, x_2, \dots)$ has any identity x_j in common, Equation 3-31 and Equation 3-33 can be safely applied only on final expression $f(x_1, x_2, \dots)$, but not necessarily through the intermediate expressions $g(x_1, x_2, \dots)$, $h(x_1, x_2, \dots)$ and $e(g(x_1, x_2, \dots), h(x_1, x_2, \dots))$. This rigid requirement may limit both capability and convenience of use for precision arithmetic. It may even be a generic requirement because an uncertainty-bearing arithmetic may need to work on the final expression to avoid counting uncertainty excessively, e.g., an uncertainty-bearing arithmetic no longer has freedom to add mutually canceling terms to an expression. To obtain the final expression $f(x_1, x_2, \dots)$, symbolic calculation similar to that used in affine arithmetic [29] may be required, so that implementation of algorithms using an uncertainty-bearing arithmetic may be more complicated than its counterpart using conventional floating-point arithmetic.

3.14 Conditional Execution

Conditional execution based on the comparison relation between two values is frequently used in practical algorithms [10]. When each value has associated uncertainty, the comparison relation between two values becomes quite different. This is particularly true for interval arithmetic, in which a value can be anywhere inside its bounding range [16]. In precision arithmetic, each value has a mathematically expected value plus a well-defined bounding

distribution for uncertainty. The comparison relation between two imprecise values in precision representation can either be defined by their mathematically expected values, or by their statistical comparison relations based on confidence [4]. However, the usage of condition execution in a traditional algorithm needs to be reevaluated conceptually with uncertainty statistics in mind when upgrading an algorithm to use precision arithmetic.

3.15 Implementation

The conventional 64-bit floating-point standard IEEE-754 [5][7][8] has:

- 11 bits for storing exponent E;
- Including a hidden MSB, 53 bits for storing significand S.
- 1 bit for storing sign;

To be a super set of conventional 64-bit floating-point standard, an 80-bit implementation of precision arithmetic has:

- 11 bits for storing exponent E;
- 53 bits for storing significand S (without using the hidden MSB);
- 1 bit for storing sign;
- 2 bit for storing carry \sim ;
- 13 bits to store bounding range R as a fixed-point value.

The precision arithmetic is implemented in C++. With heavy additional code to count for statistics and to detect implementation errors, it runs now about 7-fold slower than the implementation of interval arithmetic using Equation 2-3, Equation 2-4, Equation 2-5 and Equation 2-6. It is probably comparable in speed to the implementation of interval arithmetic without dependence problem [21]. With code weight-trimming and optimization, its speed is expected to be improved by at least 3-fold. Unlike conventional floating-point arithmetic, it only calculates limited number of significand bits, e.g., 12 bits instead of all 53 bits when the precision is 10^{-3} and χ is 2. Its slowest but very frequent operation is to find the position of

highest non-zero significand digit, which can be found instantly with a decoder [3]. So hardware optimization can also improve the speed of precision arithmetic by another few folds.

Appendix provides more implementation details for precision arithmetic.

4 Standards and Methods for Validating Uncertainty-Bearing Arithmetic

4.1 Validating Standards and Methods

Algorithms with known analytic result are used to validate an uncertainty-bearing arithmetic. The difference between the arithmetic result and the analytic result is defined as the value error. The question is whether the uncertainty bounding range or uncertainty deviation is enough to cover value error with increased amount of calculation for any input. Corresponding to two different goals for uncertainty-bearing, there are actual two different sets of measurements:

- The ratio of absolute value error to uncertainty deviation is defined as error significand for each output value. An ideal uncertainty-tracking arithmetic should have an average error significands close to 1.
- The ratio of absolute value error to uncertainty bounding range is defined as bounding ratio for each output value. An ideal uncertainty-bounding arithmetic should have a maximal bounding ratio less than but close to 1. If the maximal bounding ratio is more than 1, bounding leakage measures the probability for errors to be outside uncertainty bounding ranges, and it should be a constant.

In both cases, all measurements should be stable for an algorithm—they should not change significantly for different input deviation, input data, or the amount of calculation. For example, if different branches of conditional executions contain very different amount of calculations, such stability is crucial to obtain valid estimation of result precision.

4.2 Comparing Uncertainty-Bearing Arithmetics

Precision arithmetic tracks both uncertainty bounding range and uncertainty deviation, so it can be evaluated for both goals. Independence arithmetic has no uncertainty bounding range, while interval arithmetic has no uncertainty deviation. To be able to compare all the three arithmetics, $[x-6\delta x, x+6\delta x]$ is used artificially as the bounding range for an average value x with deviation δx for independence arithmetic, and vice versa for interval arithmetic.

The statistical assumption of precision arithmetic is weaker than that of independence arithmetic but stronger than that of interval arithmetic, so after executing a same algorithm on a same input data, the output deviation and bounding range of precision arithmetic are expected to be more than those of independence arithmetic but less than those of interval arithmetic. According to Equation 3-9 and Equation 3-6, when the exponents of the two operands are same, for addition and subtraction using precision arithmetic, the result deviation propagates in the same way as that of independence arithmetic, while the result bounding range propagates in the same way as that of interval arithmetic. According to Equation 3-21 and Equation 3-26, the result precision of multiplication and division by precision arithmetic is always larger than those by independence arithmetic. However, if both operands have precisions much less than 1, the result precision of multiplication and division is very close to that of independence arithmetic. Thus, the result of precision arithmetic should be much closer to that of independence arithmetic.

4.3 Comparing Algorithms for Tests

Algorithms of complete different nature with each representative for its category are needed to test the generic applicability of uncertainty-bearing arithmetic.

An algorithm can be categorized by comparing the amount of its input and output data:

- Transforming: A transforming algorithm has about equal amount of input and output data.

The information contained in the data remains about the same after transforming.

Discrete Fourier transformation is a typical transforming algorithm, which contains

exactly same amount of input and output data, and its output data can be transformed back to input data using essentially the same algorithm. Matrix reversion is another such reversible algorithm. For reversible transformations, a unique requirement for uncertainty-bearing arithmetic is to introduce least amount of additional uncertainty after forward and reverse transformation. A test of uncertainty-bearing arithmetics using FFT algorithms is provided in Section 5, and a test of matrix inversion is provided in Section 6.

- **Generating:** A generating algorithm has much more output data than input data. Solving differential equations numerically and generating a numerical table of a specific function are typical generating algorithms. During generating algorithm, mathematical knowledge is coded into data, so there is an increase of information in the output data. In this sense, calculating function values by Taylor expansion belongs to generating algorithms even though its input data may be more than its output data. In generating algorithms, input uncertainty should also be considered when deciding if the result is good enough so that the calculation can stop. This consideration is demonstrated by Taylor expansion in Section 8. Some generating algorithms are theoretical calculations which involve no imprecise input, so that all result uncertainty is due to rounding errors. Section 6 demonstrated such an algorithm, which calculates a table of sine function using trigonometric relations and one precise input data, $\sin(\pi/2)=1$.
- **Reducing:** A reducing algorithm has much less output data than input data, such as numerical integration, regression, and statistical characterization of a data set. Some information of the data is generally lost during reducing. Conventional wisdom is that reducing algorithm generally benefic from larger input data set [4]. Such notion needs to be reevaluated when input data contain uncertainty, or when uncertainty accumulates during calculation. A test of uncertainty-bearing arithmetics using numerical integration is provided in Section 9, and a test of linear regression is provided in Section 10.

Other relations between the input and output of can also be used to categorize an algorithm.

- In an expressive algorithm, each output is implemented as an analytic mathematical expression of inputs. Equation 3-31 and Equation 3-33 of precision arithmetic are powerful tools to solve expressive algorithms using precision arithmetic.
- In a progressive algorithm, each output is based on partial inputs and previously generated outputs. If an output depends on state which is defined by previous inputs and outputs, the algorithm is also progressive. Most practical algorithms are progressive. Even there may be an expected analytic mathematical expression between its input and output, an algorithm may not be expressive due to the way of progressive execution. Dependence problem usually presents in a progressive algorithm, such as dependence on the usage history. Section 8 demonstrates a progressive moving-window algorithm.

4.4 Input Data to Use

To test input data of any precision, a precise input value can be cast to any specific input deviation using precision representation. There are actually two ways of implementing such casting:

- A clean signal is obtained by directly casting a perfect signal to a specific precision. Such casting may contain systematic rounding errors. For an example, if a perfect sine signal repeat 2^n times in 2^{n+2} samples, the signal contains only values 0, ± 1 and $\pm 1/\sqrt{2}$, with each value repeated multiple times in the signal. Such value repeat may translate to artificial repeat of rounding errors. On the other hand, the symmetry of an arithmetic may be tested by the output symmetry of clean input signals.
- A noisy signal is obtained by adding Gaussian noise of same deviation as the input deviation to a perfect signal before casting. It represents realistic signal, and it should be used in validating arithmetic on uncertainty propagation.

5 Validation Using FFT

5.1 Frequency Response of Discrete Fourier Transformation

Each testing algorithm needs to be scrutinized carefully. One important issue is whether the digital implementation of the algorithm is faithful for the original analytic algorithm. For an example, discrete Fourier transformation is only faithful for continuous Fourier transformation at certain frequencies, and it has different degree of faithfulness for other frequencies. This is called the frequency response of discrete Fourier transformation in this paper.

For each signal sequence $h[k]$, $k = 0, 1 \dots N-1$, in which N is a positive integer, the discrete Fourier transformation $H[n]$, $n = 0, 1 \dots N-1$ is [10]:

$$\text{Equation 5-1: } H[n] = \sum_{k=0}^{N-1} h[k] \cdot e^{i2\pi \cdot n \cdot \frac{k}{N}} ;$$

The corresponding reverse transformation is [10]:

$$\text{Equation 5-2: } h[k] = \frac{1}{N} \sum_{n=0}^{N-1} H[n] \cdot e^{-i2\pi \cdot k \cdot \frac{n}{N}} ;$$

The $H[n]$ of a pure sine signal $h[k] = \sin(2\pi f k/N)$ is calculated as:

$$\text{Equation 5-3: } H[n] = \frac{\sum_{k=0}^{N-1} e^{i2\pi \cdot (n+f) \cdot \frac{k}{N}} - \sum_{k=0}^{N-1} e^{i2\pi \cdot (n-f) \cdot \frac{k}{N}}}{2i} ;$$

When f is an integer F from a set $0 \leq F < N/2$ which is defined as the index frequency for $H[n]$:

$$\text{Equation 5-4: } H[n] = i\delta_{n,F} N/2 ;$$

Otherwise:

$$\text{Equation 5-5: } H[n] = \frac{1}{2} \frac{\sin(2\pi f - 2\pi \frac{f}{N}) + \sin(2\pi \frac{f}{N}) - \sin(2\pi f) \cdot e^{-i2\pi \frac{n}{N}}}{\cos(2\pi \frac{n}{N}) - \cos(2\pi \frac{f}{N})} ;$$

Let F be the closest integer to f , and define fractional frequency $\Delta f \equiv f - F$:

$$\text{Equation 5-6: } |\Delta f| \rightarrow 0: \quad H[n] \rightarrow i\delta_{n,F} N/2;$$

$$|\Delta f| \rightarrow 1/2: \quad H[n] \rightarrow N/\pi;$$

Discrete Fourier transformation $H[n]$ of a signal $h[k]$ is the digital implementation of continuous Fourier transformation $H(s)$ of signal $h(t)$ [10], in which $H(s)=i\delta(s-f)$ for $h[k]=\sin(2\pi f)$. From Equation 5-6, when a signal frequency of the original signal falls between two index frequencies of the transformation, the peak is lower and wider with a wrong phase, depending on the fractional frequency. Thus, a discrete Fourier transformation is only faithful for signal components with exactly one of the index frequencies of the transform, and it suppresses and widens unfaithful signal components, each of which has different phase than its closest faithful representation, with the phase of a sine wave distorted toward that of a cosine wave, and vice versa. Examples of unfaithful representation of fractional frequency by discrete Fourier transformation are shown in Figure 8.

Due to its width, an unfaithful transformation may interact with other frequency components of the discrete Fourier spectrum, thus sabotages the whole idea of using Fourier transformation to decompose a signal into independent frequency components. Because the reverse Fourier transformation mathematically restores the original $\{h[k]\}$ for any $\{H[n]\}$, it exaggerates and narrows all unfaithful signal components correspondingly. This means that the common method of signal processing in Fourier space [10][13][15] may generate artifacts due to its uniform treatment of faithful and unfaithful signal components, which probably coexist in reality. Unlike aliasing [10][13][15], unfaithful representation of discrete Fourier transformation has equal effect in whole frequency range, so that it cannot be avoided by different samplings.

Unfaithful representation arises from implied assumption of discrete Fourier transformation. The continuous Fourier transformation has infinitive signal range, so that:

$$\text{Equation 5-7: } h(t) \Leftrightarrow H(s): \quad h(t-\tau) \Leftrightarrow H(s) \exp(i2\pi s \tau);$$

As an analog, the discrete Fourier transformation $G[n]$ of signal $h[k]$, $k = 1 \dots N$ can be calculated mathematically from the discrete Fourier transformation $H[n]$ of $h[k]$, $k = 0 \dots N-1$:

$$\text{Equation 5-8: } G[n] = (H[n] + h[N] - h[0]) \exp(i2\pi n/N);$$

So for $G[n]$ to be equivalent to $H[n]$, $h[N] = h[0]$, or discrete Fourier transformation has an implied assumption that the signal $h[k]$ repeats itself outside the region of $[0, N-1]$. For a unfaithful frequency, $h[N-1]$ and $h[N]$ are discontinuous in regard to signal periodicity, resulting in larger peak width, lower peak height, and wrong phase.

The most convenient signals to test uncertainty-bearing arithmetics are sine or cosine signals with index frequencies. A linear signal with slope λ , $h[k] = \lambda k$, provides a generic test for input frequencies other than index frequencies, whose Fourier spectrum is:

$$\text{Equation 5-9: } H[n] = -\lambda \frac{N}{2} \left(1 + \frac{i}{\tan(\pi \cdot n / N)}\right);$$

5.2 FFT (Fast Fourier Transformation)

When $N = 2^L$, in which L is a positive integer, the generalized Danielson-Lanczos lemma [10] can be applied in Fourier transformation as FFT (Fast Fourier Transformation) [10]:

$$\text{Equation 5-10: } H\left[n, \frac{k}{2^L}\right] = h[j], \quad k, n = 0, 1 \dots N-1, \quad j \text{ is bit-reverse of } n;$$

$$\text{Equation 5-11: } H\left[n, \frac{k}{2^m}\right] = H\left[n, \frac{k}{2^{m+1}}\right] + H\left[n, \frac{k}{2^{m+1}}\right] \cdot e^{+i2\pi \cdot n \cdot \frac{1}{2^{L-m}}}, \quad m = 0, 1 \dots L-1$$

$$\text{Equation 5-12: } H[n] = H\left[n, \frac{k}{2^0}\right];$$

Thus, each output value is obtained after applying Equation 5-11 L times. L is called FFT order in this paper.

The calculation of term $e^{i2\pi \cdot n \cdot \frac{1}{2^{L-m}}}$ in Equation 5-11 can be simplified. Let \ll denotes bit left-shift operation and let $\&$ denote bitwise AND operation:

Equation 5-13: $\phi[n] \equiv e^{i2\pi \cdot n \frac{1}{2^L}} ; \quad e^{i2\pi \cdot n \frac{1}{2^{L-m}}} = \phi[(n \ll m) \& ((1 \ll L) - 1)] ;$

It is important to have an accurate phase factor array $\phi[n]$ when tracking the FFT calculation error. The accuracy of $\phi[n]$ can be checked rigidly within itself by trigonometric relations, so that no significant error is introduced from trigonometric functions.

Equation 5-11 sums up two mutually independent operands, so the error propagation in a FFT algorithm is precisely tracked by independence arithmetic, and dependency problem should not be a concern for interval arithmetic and precision arithmetic.

FFT is one of the most widely used algorithms [10]. By providing a balanced usage of addition, subtraction and multiplication involving trigonometric functions, it services as one of the most important benchmarks in testing processors for overall mathematical performance [34]. Since all three uncertainty-bearing arithmetics are generic in nature, e.g., no special optimization for FFT algorithms, the testing result using FFT algorithms can be generic for expressive algorithms. FFT algorithms provide a good linear platform to test any uncertainty-bearing arithmetic, with (1) a clearly defined value measuring the amount of calculation, (2) a known error propagation mechanism, (3) no conditional execution in the algorithm, and (4) usage of basic arithmetic operations only without dependence problem.

5.3 Evaluating Calculation inside Uncertainty

Figure 9 shows the output deviations and value errors for a noisy sine signal after forward FFT. It shows that the output deviations using precision arithmetic are slightly larger than those using independence arithmetic, but much less than those using interval arithmetic. For a fixed input deviation, the output deviation using independence arithmetic is a constant for each FFT. Because value and uncertainty interact with each other through normalization in precision arithmetic, output deviations of Equation 5-11 are no longer a constant. One interesting

consequence is that only in precision arithmetic the output deviations for a noisy input signal are larger than those for a corresponding clean input signal.

Figure 9 shows that the value errors calculated using precision arithmetic are comparable to those using conventional floating-point arithmetic, and they are both comparable to the output deviations using either precision arithmetic or independence arithmetic. In another word, the result of calculating 2-bit or 53-bit into uncertainty are quite comparable, so that limited calculation inside uncertainty is reasonable.

Figure 10 compares the output value errors of precision arithmetics calculating different bit inside uncertainty. With no calculation inside uncertainty, the output value errors are only on four levels. Such quantum distribution is reduced noticeable by the 2-bit calculation inside uncertainty, and further reduced by the 4-bit calculation inside uncertainty. Compared with Figure 9, Figure 10 shows that the result using precision arithmetic with 2-bit calculation inside uncertainty approaches that of using independence arithmetic, so that 2-bit calculation inside uncertainty seems sufficient. Precision arithmetic with 2-bit calculation inside uncertainty is used for further tests.

5.4 Evaluating Uncertainty Distribution

Precision arithmetic tracks all increases of rounding errors, but it can not track decreases of rounding error due to mutual cancellations during arithmetic operations. So the uncertainty distribution provided by precision arithmetic serves as bounding distribution for value errors, and the actual distribution could be narrower than the bounding distribution. FFT provides a good test for such probability bounding. Its forward and reverse algorithms are identical except a constant, so that they result in exactly the same bounding probability distributions. On the other hand, forward FFT condenses a sine signal into only two non-zero imaginary values by mutual cancellation of signal components, while reverse FFT spreads only two non-zero imaginary

values to construct a sine signal. Thus forward FFT is more sensitive to calculation errors than reverse FFT, and should have a broader actual uncertainty distribution.

Because value errors are only observable as errors of result significand, the theoretical rounding error distributions obtained from Equation 3-16 are integrated around each integer value of rounding error, to result in theoretical probability densities of value error significand for different bounding ranges. The occurrence frequencies of output value error significand are counted for different bounding ranges and normalized as measured probability densities. Figure 11 and Figure 12 respectively shows the theoretical and empirical distribution of significand errors for different bounding ranges on noisy sine signals for forward and reverse FFT respectively using precision arithmetic with 2-bit calculated inside uncertainty. They shows that the uncertainty distribution of forward FFT is very close to the bounding distribution, while the uncertainty distribution of reverse FFT is indeed narrower. In contrast, for linear input signal, forward and reverse FFT have almost identical value error significands distribution. Thus, the previous hypothesis on the distribution width of value error significands is validated empirically in this case.

Another way to measure empirical probability distribution is to normalize output value error with the corresponding output uncertainty deviation. If output value errors are Gaussian distributed with deviation given precisely by the corresponding output uncertainty deviation, then the normalized histogram should be normal distributed. Figure 13 shows that the normalized histograms using independence arithmetic are best fit by a Gaussian distribution with a deviation of 0.98 and a mean of 0.06. The reason for this small difference between normal distribution and the measured probability distribution is not clear at this moment. If there were no rounding errors, all the result should have zero value errors. Indeed in Figure 13, the measured histograms have much larger population at where the value errors are zero. Figure 14 shows that the measured normalized histograms using precision arithmetic are very similar to those using independence arithmetic, and they are all well fitted by a Gaussian distribution.

5.5 Evaluating Uncertainty-Tracking

Figure 15 shows that for a same input deviation, the output deviations of forward FFT increases exponentially with FFT order using all three arithmetics. Figure 16 shows that for a same FFT order, the output deviations of forward FFT increases linearly with input deviation using all three arithmetics. The output deviation does not change with input frequency, so that all data of a same input deviation and FFT order but different input frequencies can be pooled together during analysis. The trends in Figure 15 and Figure 16 is modeled by Equation 5-14, in which L is FFT order, δx is input deviation, δy is output average deviation, α and β are empirical fitting constants:

$$\text{Equation 5-14: } \delta y = \alpha \beta^L \delta x;$$

β measures propagation speed of deviation with increased amount of calculation in Equation 5-14. It is called propagation base rate. Unless β is close to 1, β dominates α in fitting thus determines characteristics of Equation 5-14.

It turns out that Equation 5-14 is very good fits for both average output deviations and value errors for all three arithmetics, such as demonstrated in Figure 17. Because uncertainty-tracking is a competition between error propagation and uncertainty propagation, the average output error significand for forward FFT is expected to fit Equation 5-15 and Equation 5-16, in which z is average output error significand, L is FFT order, $(\alpha_{\text{dev}}, \beta_{\text{dev}})$ and $(\alpha_{\text{err}}, \beta_{\text{err}})$ are fitting parameters of Equation 5-14 for average output deviations and value errors respectively:

$$\text{Equation 5-15: } z = \alpha \beta^L;$$

$$\text{Equation 5-16: } \alpha = \alpha_{\text{err}}/\alpha_{\text{dev}}; \quad \beta = \beta_{\text{err}}/\beta_{\text{dev}};$$

The estimated average output error significand can then be compared with the measured ones to evaluate the predictability of uncertainty-tracking mechanism. One example of measured average output error significands is shown in Figure 18, which shows that the average output error significands using precision arithmetic are a constant despite that both average output

uncertainty deviations and value errors increases linearly with input deviation and exponentially with FFT order. Equation 5-14 and Equation 5-15 are found empirically to be good fit for any FFT algorithm with any input signal using any arithmetic.

Reverse FFT algorithm is identical to Forward FFT algorithm, except:

- Reverse FFT algorithm uses constant $(-i)$ instead $(+i)$ in Equation 5-11.
- Reverse FFT algorithm divides the result further by 2^L .

Thus, the output average deviations and value errors of reverse FFT algorithm are expected to obey Equation 5-14 and Equation 5-17, in which $(\alpha_{for}, \beta_{for})$ are corresponding fitting parameters of Equation 5-14 for forward FFT, while the output average error significands is expected to obey Equation 5-15 with same α and β as those of forward FFT.

Equation 5-17: $\alpha = \alpha_{for}; \quad \beta = \beta_{for}/2;$

Roundtrip FFT is forward FFT followed by reverse FFT, with the output of forward FFT as input to reverse FFT. Thus, both its average output deviations and value errors are expected to fit Equation 5-14 and Equation 5-18, in which $(\alpha_{for}, \beta_{for})$ and $(\alpha_{rev}, \beta_{rev})$ are corresponding fitting parameters of Equation 5-14 for forward FFT and reverse FFT respectively. And its error significands are expected to fit Equation 5-15 and Equation 5-18, in which $(\alpha_{for}, \beta_{for})$ and $(\alpha_{rev}, \beta_{rev})$ are corresponding fitting parameters of Equation 5-15 for forward FFT and reverse FFT respectively.

Equation 5-18: $\alpha = \alpha_{for} \alpha_{rev}; \quad \beta = \beta_{for} \beta_{rev};$

Figure 19, Figure 20 and Figure 21 show the fitting of β for independent, precision and interval arithmetic for all the three algorithms respectively. These three figures show that all measured β makes no distinction between input signals for any algorithms using any arithmetic, e.g., no difference between real and imaginary part for a sine signal. The estimated β for average error significands is obtained from Equation 5-16. The estimated β for average uncertainty deviations and value errors for reverse FFT and roundtrip FFT are obtained from

Equation 5-17 and Equation 5-18 respectively. The estimated β for average uncertainty deviations for forward FFT is $\sqrt{2}$, which will be demonstrated later. The measured β and the estimated β agree well with each other in all cases. This confirms that uncertainty-tracking is a simple competition between error propagation and uncertainty propagation:

- Figure 19 confirms that independence arithmetic is ideal for uncertainty-tracking for FFT algorithms: (1) β for error significands is a constant 1; and (2) β for both the average output deviations and value errors are both 1 for roundtrip FFT because the result signal after roundtrip FFT should be restored as the original signal. Thus, the theoretical β for forward FFT and reverse FFT are $\sqrt{2}$ and $1/\sqrt{2}$ respectively.
- Precision arithmetic has β for average output deviations slightly larger than those of value errors, resulting in β for average output error significands to be a constant slightly less than 1. Its β for average output deviations is slightly larger than the corresponding β of independence arithmetic, so its average output deviations propagate slightly faster with increased FFT order than those of independent arithmetic. Such slightly faster increase with the amount calculation arithmetic is anticipated by the difference between Equation 3-21 and Equation 2-12 with $\gamma=0$.
- Interval arithmetic has β for average output deviations always much larger than β for average output value errors, resulting in β for average output error significands of about 0.62 for forward and reverse FFT, and about $0.39 \cong (0.62)^2$ for roundtrip FFT. So its average output deviations propagate much faster with the amount of calculations than its value error does. Such fast propagation of uncertainty ranges is intrinsic to interval arithmetic due to its worst-case assumption.

Figure 22 shows that for forward FFT the measured average output error significands using either precision arithmetic or independence arithmetic are both approximately constant of 0.8 regardless of FFT order. In contrast, it shows that using interval arithmetic the measured

average output error significands decrease exponentially with FFT order L . Such trends of average error significands hold for all three FFT algorithms and all input signals. Thus, in this case, the direct uncertainty tracking provided by precision arithmetic is better than the indirect uncertainty tracking provided by interval arithmetic.

Figure 23 shows that using precision arithmetic, each average output uncertainty deviation equals the corresponding input uncertainty deviation for all FFT orders. This is also true for the average output value errors. Thus after each round-trip operation, precision arithmetic restores the original signal for FFT. Such behavior seems ideal for a reversible algorithm. In contrast, Figure 24 shows that using interval arithmetic, the average output uncertainty deviations increase exponentially with FFT orders, which means undesirable broadening of uncertainty.

5.6 Evaluating Uncertainty-Bounding

While uncertainty-tracking is the result of propagation competition between average output deviations and average values errors with increased amount of calculations, uncertainty-bounding is the result of propagation competition between maximal output bounding ranges and maximal value errors, both of which still fits Equation 5-14 well experimentally using any arithmetic. Equation 5-15 and Equation 5-16 can be used to estimate maximal bounding ratio as well. For example, Figure 25 shows that the maximal output bounding ratios using precision arithmetic fits Equation 5-15 well. Unlike average output error significands in Figure 18, the maximal output bounding ratios increase slowly with FFT order using precision arithmetic.

To characterize trend of maximal output bounding ratios with FFT order, β for measured average output deviation vs. measured maximal output range, β for measured average vs. maximal output value errors, and β for measured average error significand vs. measured maximal bounding ratios are plotted side-by-side respectively in an arithmetic characterization plot for a particular input signal. Figure 26 shows the characterization plot for independence arithmetic while Figure 27 shows the characterization plot for precision arithmetic. In both

characterization plots, β for maximal value error exceeds β for maximal uncertainty bounding ranges, suggesting that maximal bound ratio should increase with FFT order, which corresponds well with β for maximal bounding ratio to be slightly larger than 1. Figure 29 shows that the output maximal bounding ratios indeed increase with FFT order of forward FFT for both arithmetics. As shown in Figure 28 and Figure 29, like its average error significands, interval arithmetic has its maximal bounding ratios decreasing exponentially with increased FFT order for all algorithms while keeps its bounding leakages at constant 0. Like independence arithmetic and precision arithmetic, its characterization plot is stable among different types of input signals. Whether precision arithmetic is better than interval arithmetic in uncertainty bounding depends on the statistical requirements for the uncertainty-bounding:

- In situation when absolute bounding is required, interval arithmetic is the only choice.
- In range estimation [1] involving low-resolution measurements whose sources of uncertainty are unclear, interval arithmetic is a better choice because the independence uncertainty assumption of precision arithmetic may not be satisfied.
- Otherwise if precision arithmetic can also be used, precision arithmetic should be more suitable for normal usages because most input data already contain bounding leakage.

5.7 Evaluating Deterministic Precision Round-up Rule

The result of current precision round-up rule is not deterministic for a value of $(2S+1)?@E$. If deterministic is a requirement, rounding $(2S+1)?@E$ always to $(S+1)-@E+1$ provides slightly higher significance than rounding it always to $S+@E+1$. Compared with the random precision round-up rule, both flavors of precision arithmetic have identical result in both uncertainty tracking and in uncertainty bounding, although both are not conceptually as good the random precision round-up rules.

6 Validation using Matrix Inversion

6.1 Uncertainty Propagation in Matrix Determinant

Assume vector $[p_1, p_2 \dots p_n]_n$ is a permutation of vector $(1, 2 \dots n)$, and $\text{Sign}[p_1, p_2 \dots p_n]_n$ is its permutation sign [30]. For a n -by- n square matrix M with element x_{ij} , $i, j=1, 2 \dots n$, define determinant as [10]:

$$\text{Equation 6-1: } |M| \equiv \sum_{[p_1 \dots p_n]_n} \text{Sign}[p_1 \dots p_n]_n \prod_k x_{k, p_k} ;$$

Define sub-determinant at index (i, j) as [31]:

$$\text{Equation 6-2: } |M_{i,j}| \equiv \sum_{[p_1 \dots p_n]_n}^{p_i=j} \text{Sign}[p_1 \dots p_n]_n \prod_{k \neq i} x_{k, p_k} ;$$

$(-1)^{i+j} |M_{(i,j)}|$ is the determinant of the $(n-1)$ -by- $(n-1)$ matrix that results from deleting row i and column j of M [10]. Equation 6-3 holds for arbitrary row index i or arbitrary column index j .

$$\text{Equation 6-3: } |M| = \sum_{j=1}^n |M_{i,j}| x_{i,j} = \sum_{i=1}^n |M_{i,j}| x_{i,j} = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^n |M_{i,j}| x_{i,j} ;$$

When $i_1 \neq i_2, j_1 \neq j_2$, define sub-determinant at index $(i_1, j_1) (i_2, j_2)$ as:

$$\text{Equation 6-4: } |M_{[i_1, i_2][j_1, j_2]}| \equiv \sum_{[p_1 \dots p_n]_n}^{p_{i_1}=j_1, p_{i_2}=j_2} \text{Sign}[p_1 \dots p_n]_n \prod_{k \neq i_1, k \neq i_2} x_{k, p_k} ;$$

Similarity, $(-1)^{i_1+i_2+j_1+j_2} |M_{[i_1, i_2][j_1, j_2]}|$ is the determinant of the $(n-2)$ -by- $(n-2)$ matrix that results from deleting row i_1 and i_2 , column j_1 and j_2 of M . This leads to Equation 6-5.

$$\text{Equation 6-5: } |M_{[i_1, i_2][j_1, j_2]}| = |M_{[i_1, i_2][j_2, j_1]}| = |M_{[i_2, i_1][j_1, j_2]}| = |M_{[i_2, i_1][j_2, j_1]}| ;$$

Assuming $p_1, p_2 \in \{1, 2 \dots n\}$, define $[p_1, p_2]_n$ as length-2 unordered permutation which satisfies $p_1 \neq p_2$, and define $\langle p_1, p_2 \rangle_n$ as length-2 ordered permutation which satisfies $p_1 < p_2$. Ordered permutation is mathematically equivalent to combination. The four equivalent sub-determinants in Equation 6-5 can be uniquely identified as $|M_{\langle i_1, i_2 \rangle \langle j_1, j_2 \rangle}|$. Such definition of sub-determinant can be

extended to $|M_{<i_1 \dots i_k >_n <j_1 \dots j_k >_n}| = |M_{[i_1 \dots i_k]_n [j_1 \dots j_k]_n}|$, in which k is an arbitrary integer between 1 and n, and $i_1 \dots i_k$ are a same set of indexes. Define $|M_{<1 \dots n >_n <1 \dots n >_n}| \equiv 1$.

According to the goal and basic assumption of precision arithmetic, the uncertainty of each element $x_{i,j}$ is independently and symmetrically distributed. Assume $y_{i,j}$ is a random variable at index (i, j) symmetrically distributed with a deviation $\delta x_{i,j}$. Define $|\hat{M}|$ as the determinant of the matrix \hat{M} whose element is $(x_{i,j} + y_{i,j})$. Applying Equation 3-27 to calculate $\delta|M|^2$:

$$\text{Equation 6-6: } \delta|M|^2 = \int (|\hat{M}| - |M|)^2 \prod_m \prod_n \rho(y_{m,n}) dy_{m,n} ;$$

According to Equation 3-28:

$$\text{Equation 6-7: } \int |\hat{M}| \prod_m \prod_n \rho(y_{m,n}) dy_{m,n} = |M| ;$$

$$\text{Equation 6-8: } \delta|M|^2 = \sum_{[p_1 \dots p_n]_n} \sum_{[q_1 \dots q_n]_n} \text{Sign}[p_1 \dots p_n]_n \text{Sign}[q_1 \dots q_n]_n \prod_k (x_{k,p_k} x_{k,q_k} + \delta x_{k,p_k}^2 \delta_{p_k,q_k}) - |M|^2 ;$$

$$\text{Equation 6-9: } |M|^2 = \sum_{[p_1 \dots p_n]_n} \sum_{[q_1 \dots q_n]_n} \text{Sign}[p_1 \dots p_n]_n \text{Sign}[q_1 \dots q_n]_n \prod_k x_{k,p_k} x_{k,q_k} ;$$

Equation 6-8 is not ideal for implementation because it contains differences of large values. In Equation 6-8, each product term to be summed in $\delta|M|^2$ contains at least one and at most n deviation square of the elements. It can be expanded by Equation 6-10, in which k is the count of deviation square of the elements:

$$\text{Equation 6-10: } \delta|M|^2 \equiv \sum_{k=1}^n \delta|M|_{(k)}^2 ;$$

- The sum of all product terms in $\delta|M|_{(1)}^2$ containing $\delta x_{i,j}^2$ is given by Equation 6-11.

$$\text{Equation 6-11: } \delta x_{i,j}^2 \sum_{[p_1 \dots p_n]_n} \sum_{[q_1 \dots q_n]_n} \text{Sign}[p_1 \dots p_n]_n \text{Sign}[q_1 \dots q_n]_n \prod_{k \neq i} x_{k,p_k} x_{k,q_k} = \delta x_{i,j}^2 |M_{i,j}|^2 ;$$

- Similarly, the sum of all product terms in $\delta|M|_{(2)}^2$ containing $\delta x_{i_1, j_1}^2 \delta x_{i_2, j_2}^2$ is given by

$$\delta x_{i_1, j_1}^2 \delta x_{i_2, j_2}^2 |M_{<i_1, i_2> <j_1, j_2>}|^2. \text{ The exchange of } (i_1, j_1) \text{ and } (i_2, j_2) \text{ results in the same term}$$

so such repeat needs to be avoided. Thus $\delta|M|_{(2)}^2$ is calculated by Equation 6-12.

$$\text{Equation 6-12: } \delta|M|_{(2)}^2 = \sum_{<p_1, p_2>_n} \sum_{[q_1, q_2]_n} \delta x_{p_1, q_1}^2 \delta x_{p_2, q_2}^2 |M_{<p_1, p_2> [q_1, q_2]}|^2;$$

Equation 6-13 generalizes Equation 6-12. Equation 6-10 and Equation 6-13 can also be obtained by applying Equation 3-31 regressively to Equation 6-3. .

$$\text{Equation 6-13: } \delta|M|_{(k)}^2 = \sum_{<p_1 \dots p_k>_n} \sum_{[q_1 \dots q_k]_n} \delta x_{p_1, q_1}^2 \dots \delta x_{p_k, q_k}^2 |M_{<p_1 \dots p_k>_n [q_1 \dots q_k]_n}|^2;$$

Starting from $|M_{<1 \dots n>_n <1 \dots n>_n}| \equiv 1$, Equation 6-14 is a regressive implementation of Equation 6-10 and Equation 6-13. It provides a most efficient method to calculate matrix determinant.

$$\text{Equation 6-14: } \delta|M_{<p_1 \dots p_k>_n <q_1 \dots q_k>_n}|^2 = \sum_{p_i} \sum_{q_j} \delta x_{p_i, q_j}^2 (|M_{<p_1 \dots p_i \dots p_k>_n <q_1 \dots q_j \dots q_k>_n}|^2 + \delta|M_{<p_1 \dots p_i \dots p_k>_n <q_1 \dots q_j \dots q_k>_n}|^2);$$

The element $z_{i,j}$ at index (i,j) of the inverted matrix M^{-1} is calculated as [31]:

$$\text{Equation 6-15: } z_{i,j} = \frac{|M_{j,i}|}{|M|};$$

Equation 6-15 shows that the uncertainty of matrix determinant propagates to every element of the inverted matrix M^{-1} . Instead, the matrix which consists of element $|M_{j,i}|$ at index (i, j) is defined as adjugate matrix M^A [32], whose elements are not directly affected by $|M|$.

When using precision arithmetic to calculate adjugate matrix M^A according to Equation 6-8, the result is independent of the sequence of execution. This is quite different from traditional algorithms to calculate adjugate matrix M^A using conventional floating-point arithmetic such as LU decomposition [10], which carefully chooses sequence of execution to minimize rounding errors.

6.2 Matrix Testing Algorithm

A matrix \mathbf{M} is constructed using random integers between $[-16384, +16384]$. Its adjugate matrix \mathbf{M}^A and its determinant $|\mathbf{M}|$ are calculated precisely using integer arithmetic. \mathbf{M} , $|\mathbf{M}|$ and \mathbf{M}^A are all scaled down proportionally as M , $|\mathbf{M}|$ and M^A so that the elements of M are 2's fractional numbers randomly distributed between $[-1, +1]$. The scaled down matrix M is called a clean testing matrix. M^{-1} is calculated from $|\mathbf{M}|$ and \mathbf{M}^A using Equation 6-15, and it is deemed precise⁴. A floating-point arithmetic is used to calculate M^A and M^{-1} from M , and the results are compared with the corresponding precise results for value errors. Gaussian noises corresponding to different deviations between 10^{-17} and 10^{-1} may be added to each clean testing matrix, to result in noisy testing matrix. Each combination of matrix size and input deviation is tested by 32 different noisy matrixes.

6.3 Testing Matrix Stability

Each matrix has different stability [33], which means how stable the inverted matrix is in regard to small value changes of the original matrix elements. It is well known that more mutual cancellations in Equation 6-1 mean less stability of the matrix, and a matrix with determinant near zero is generally not stable [9][10]. Condition number has been defined to quantify the stability of a matrix [33]. Even though the definition of condition number excludes the effects of rounding errors, in reality most calculation are done numerically using conventional floating-point arithmetic, so that the combination effect of rounding errors and matrix instability cannot

⁴ Because $|M_{j,i}|$ and $|\mathbf{M}|$ are not independent of each other, M^{-1} calculated by Equation 6-15 using precision arithmetic contains dependency problem. Only introduced in the last step, this dependency problem does not affect M^{-1} severely, but it affects $(M^{-1})^{-1}$ noticeably. For example, the result error distribution of $(M^{-1})^{-1}$ is no longer Gaussian, and the result average error significands increases slowly with the matrix size.

be avoided. When a matrix is unstable, the result is more error-prone due to rounding errors of conventional floating-point arithmetic [9]. Unfortunately, conventional floating-point arithmetic has neither control nor characterization on rounding errors, so that there are no general means to avoid the mysterious and nasty “numerical instability” in numerical applications due to rounding errors [9]. For example, the numerical value of the calculated condition number of a matrix may already be a victim of “numerical instability” and there is no sure way to judge this suspicion, so this value may not be very useful in judging the stability of the matrix in practice. On the other hand, the rounding errors of conventional floating-point arithmetic can be used to test stability of a matrix. If inverted matrix and adjugate matrix are calculated using conventional floating point arithmetic, larger value errors indicate less stability of the matrix.

Precision arithmetic accounts for all rounding error with a stable and relatively narrow bounding range. The more mutual cancellations in Equation 6-1 will result in smaller absolute value related to the uncertainty deviation of the determinant. So the significance of the determinant $|M|$ of a matrix M calculated using precision arithmetic measures the amount of mutual cancellations, and it may measure the stability of the matrix. Particularly, if $|M|$ has smaller significance, then each element of M^{-1} should tend to have larger value error according to Equation 6-15. This hypothesis is confirmed by Figure 30, which shows a good linear relation between the significance of $|M|$ and maximal value error of its inverted matrix M^{-1} , regardless of the size of the matrix. Average output values errors are related to the significance of $|M|$ in the same fashion. In contrast, Figure 31 shows that the value errors of adjugate matrix M^A do not depend noticeably on the significance of $|M|$. Thus the decreased significance of denominator in Equation 6-15 determines overall stability in matrix inversion. Such observation confirms the validity of common advice to avoid matrix inversion operations in general [10].

Such linear relation between the significance and value error also extends to the calculation of adjugate matrix. Define relative value error as the ratio of value error divided by the expected value. Relative error is expected to correspond to result significance quantitatively. Figure 32

compares each significance of the sub-matrix determinant $|M_{j,i}|$ with the corresponding relative error of element at index (i, j) of the adjugate matrix M^A of clean matrix of different sizes. It shows that larger relative errors of adjugate matrix elements indeed correspond to smaller significances of the sub-matrix determinant.

For a specified input deviation, Gaussian noises are added to each matrix element before determinant is calculated using precision arithmetic. Figure 33 shows that the significance of matrix determinant decreased linearly with increased input uncertainty of matrix elements, while it remains almost the same for different matrix size. Figure 34 further shows that such decrease is independent of the determinant significances of the original clean testing matrix, until the added noise is comparable to the value of each element. Such homogeneous linear decrease of determinant significance with increased input precisions suggests that the determinant significance is a relatively stable characterization of a matrix in regard to element uncertainty.

6.4 Testing Uncertainty Propagation in Adjugate Matrix

When the adjugate matrix is calculated using precision arithmetic, Figure 35 shows that average output deviations for adjugate matrix increase linearly with input precision, which is in good agreement with Equation 5-14. Such relation is also true for maximal and average output values errors. Equation 5-14 is expected to describe general value error propagation for linear algorithms in which L is the amount of calculations [12]. The question is what value L should be when calculating the adjugate matrix of a square matrix of size N . Figure 35 suggests that L increases with N^2 for average output precision and average output error, while Figure 36 suggests that L increases with N . The discrepancy between Figure 35 and Figure 36 is unexplained at the moment.

Figure 37 shows that the average output error significances of the adjugate matrix using precision arithmetic is approximately a constant of 0.8. Figure 37 is very similar to Figure 18. Similar to the maximal output bounding ratios of FFT algorithms, the maximal output bounding

ratios for adjugate matrix using precision also obey Equation 5-15 well, with a β of 1.005, meaning a slow increase with matrix size. Added to the similarity is the normalized error distribution shown in Figure 38, which is very similar to Figure 14. Even though FFT and calculating adjugate matrix are two very different set of linear transformational algorithms, their uncertainty propagation characteristics are remarkably similar even at quantitative details. This similarity indicates that precision arithmetic is a generic arithmetic.

7 Validation Using Regressive Calculation of Sine Values

Starting from Equation 7-1, Equation 7-2 and Equation 7-3 can be used regressively to calculate the phase array $\phi[n]$ in Equation 5-13.

$$\text{Equation 7-1: } \sin(0) = \cos\left(\frac{\pi}{2}\right) = 0; \quad \sin\left(\frac{\pi}{2}\right) = \cos(0) = 1;$$

$$\text{Equation 7-2: } \sin\left(\frac{\alpha + \beta}{2}\right) = \sqrt{\frac{1 - \cos(\alpha + \beta)}{2}} = \sqrt{\frac{1 - \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)}{2}};$$

$$\text{Equation 7-3: } \cos\left(\frac{\alpha + \beta}{2}\right) = \sqrt{\frac{1 + \cos(\alpha + \beta)}{2}} = \sqrt{\frac{1 + \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)}{2}};$$

This algorithm is very different from both FFT and matrix inversion in nature, because Equation 7-2 and Equation 7-3 are no longer linear, and the test is of pure theoretical calculation without input uncertainty. The regression count L is a good measurement for the amount of calculations. Each repeated use of Equation 7-2 and Equation 7-3 accumulates calculation errors to the next usage, so that both value errors and uncertainty are expected to increase with L . Each regression count L corresponds to 2^{L-2} outputs, which enables statistical analysis for large L .

Figure 39 shows that both maximal output value errors and the corresponding output deviation increase exponentially with regression count for all three arithmetics, and Figure 40 shows that in response to increased amount of calculations: (1) the average error significands

for precision arithmetic is a constant about 0.25; (2) the maximal output bounding ratio for precision arithmetic increases slowly; (3) the average error significands for interval arithmetic decreases exponentially; and (4) the maximal output bounding ratio for interval arithmetic remains roughly a constant. Unlike FFT algorithms, the initial precise sine values participate in every stage of the regression, which results in few small output deviations at each regression. Detailed inspection shows that the maximal output bounding ratio for interval arithmetic are all obtained from small output deviations, and bounding ratios using interval arithmetic in general decreases exponentially with the amount of calculations. Thus, the result uncertainty propagation characteristics of regressive calculation of sine values are very similar to those of both FFT and calculating adjugate matrix; even though all these algorithms are quite different in nature. This may indicate again that the stability of precision arithmetic is generic regardless of algorithms used.

8 Validation Using Taylor Expansion

When a Taylor expansion is implemented using conventional floating-point arithmetic, the rounding errors are ignored, so that the result of a higher order of expansion is assumed to be more accurate, because the Cauchy estimator of the expansion, which gives an upper bound for the remainder of the expansion, decreases with the order of the expansion for analytic expressions [35]. However, because higher order of expansion also accumulate more calculation errors, there must exist an optimal order of expansion beyond which the actual remainder of the expansion can no longer be reduced due to rounding errors.

Equation 8-1 provides an example test for using precision arithmetic in Taylor expansion.

Equation 8-1:
$$f_n(x) \equiv \sum_{j=0}^n (-x)^j, \quad 1/(1+x) = \lim_{n \rightarrow +\infty} f_n(x)$$

In Equation 8-1, the absolute value of $(n+1)^{\text{th}}$ term in the expansion is the Cauchy remainder estimator of n^{th} order expansion. So Equation 8-1 is analytic when $|x| < 1$, and a smaller value $|x|$

means faster convergence to the correct value $1/(1+x)$. Mathematically, when Equation 8-1 is carried out using conventional floating-point arithmetic with 53 significant bits, if $x = 1/2^k$, in which k is a positive integer, the expansion reaches its accurate value at order $53/k$ of expansion, and remains accurate at higher order.

Because Equation 3-31 gives uncertainty deviations due to calculation errors, precision arithmetic can be used to find the optimal order of Taylor expansion. When $\delta x \ll 1$, $\delta f_n(x) = \delta x$. So using conventional floating-point arithmetic a Taylor expansion will converge to the correct value if the Cauchy estimator can be less than input uncertainty range before the expansion order exceeds 53. This is confirmed by Figure 41, which shows that:

- when $x=1/4$, the expansion reaches its accurate value at 26th order,
- when $x=1/2$, the expansion reaches its accurate value at 53th order, and
- when $x=3/4$, the expansion never reaches its accurate value because at 53th order the deviation of the expansion is still much larger than the input deviation of 2.22×10^{-16} .

When Taylor expansion is carried out by precision arithmetic, the optimal order of expansion should be reached whenever the Cauchy estimator of the expansion is no longer detectable. Thus, the optimal order of expansion should increase with increased bits calculated inside uncertainty. Figure 42, Figure 43 and Figure 44 respectively show the remainders of the expansion for different input values of $1/4$, $1/2$ and $3/4$ but same input uncertainty deviation of 10^{-4} using precision arithmetic with 0-, 2- and 4-bit calculated inside uncertainty. Also shown in these figures are (1) the result uncertainty deviation at input value of $1/2$ for different expansion order in solid line, and (2) the fraction of the result uncertainty deviation corresponding to bits calculated inside uncertainty, e.g., for 2-bit calculated inside uncertainty, the fraction is $1/4$. Figure 42, Figure 43 and Figure 44 show that more bits calculated inside uncertainty corresponds to larger expansion orders, and in each case the remainder decreases to zero when the Cauchy estimator is slightly less than the corresponding fraction of the result

uncertainty deviation. Thus for Taylor expansion, the result uncertainty deviation for each input values can be calculated according to Equation 3-31, and it can be used to obtain the optimal order of the expansion. In another word, the validity of using Equation 3-31 to calculate result uncertainty deviation is demonstrated by these cases of Taylor expansion.

Figure 42, Figure 43 and Figure 44 also show how the stable uncertainty level is reached in Equation 3-31. It shows a sudden drop of uncertainty level at the 2nd order expansion. The reason is that $f_2'(1/2) = 0$ for the 2nd order expand, $f_2(x) = (1 - x + x^2)$, so the uncertainty level drops down to $\sim(\delta x)^2$ instead of $\sim\delta x$. Higher order of expansions restores the expansion uncertainty deviations back to $\sim\delta x$. Taylor expansion using precision arithmetic is required to be more than some minimal order to reach a stable value of Equation 3-31. Fortunately, Figure 42 shows that Equation 3-31 converges to its stable value much faster than its corresponding Taylor expansion.

9 Validation Using Numerical Integration

In numerical integration over variable x using conventional floating-point arithmetic, finer sampling of the function to be integrated $f(x)$ is associated with better result [10], and it is assumed that $f(x)$ can be sampled at infinitive fine intervals of x . In reality, a floating-point arithmetic has limited significant bits, so $f(x)$ cannot be sampled at infinitive fine intervals of x , and rounding errors will increase with finer sampling of $f(x)$. Because precision arithmetic tracks rounding errors of calculation efficiently, it can be used to search for optimal intervals for numerical integration for the best possible result. A simplest depth-first binary-tree search algorithm is used in this paper. For each integration interval $[x_{\text{start}}, x_{\text{end}}]$, define:

$$\text{Equation 9-1: } x_{\text{mid}} \equiv (x_{\text{start}} + x_{\text{end}}) / 2;$$

$$\text{Equation 9-2: } f_{\text{err}} \equiv (f(x_{\text{start}}) + f(x_{\text{end}})) / 2 - f(x_{\text{mid}});$$

$$\text{Equation 9-3: } f_{\Delta} \equiv f(x_{\text{mid}}) (x_{\text{end}} - x_{\text{start}});$$

If f_{err} becomes insignificant, the interval $[x_{\text{start}}, x_{\text{end}}]$ is considered to be fine enough, and f_{Δ} is added to the total integration. Otherwise, the search continues on the intervals $[x_{\text{start}}, x_{\text{mid}}]$ and $[x_{\text{mid}}, x_{\text{end}}]$, which is the next depth for searching. This searching algorithm is very adaptive, with local search depth depending only on how $f(x)$ changes locally. However, such adaptation to local change of $f(x)$ brings one weakness to this searching algorithm—when $f(0)=f'(0)=0$, the algorithm spends the majority of execution time around $x=0$, searching in tiny intervals of great depth, and adding tiny significant values to the result each time. This weakness is called zero trap here. It cannot be removed by simply offsetting $f(x)$ by a constant because doing so will change the precision of each sampling of $f(x)$, and increase output uncertainty deviation. For a proof-of-principle demonstration, zero trap is avoided in this paper.

Equation 9-4 provides an example test for the above simple algorithm, in which n is a positive integer.

Equation 9-4:
$$\frac{4^{n+1} - 10^{-6(n+1)}}{n+1} = \int_{10^{-6}}^4 x^n dx ;$$

Table 5 shows that the result of numerical integration is very comparable to the expected value. It shows that the above integration algorithm introduces no broadening of result uncertainty, so the above algorithm always selects optimal integration intervals when calculating best possible result for a numerical integration. Tests of integration using different polynomials with different integration ranges all confirm the above result.

One thing worth noticing in Table 5 is that even Equation 9-3 consistently underestimate integration for each integration interval $[x_{\text{start}}, x_{\text{end}}]$, the final underestimation is quite small, and comparable to the uncertainty deviation. This example shows that bias inside uncertainty range has insignificant contribution to final result using precision arithmetic, and calculating deep inside uncertainty does not seem worthwhile.

10 Validation Using Progressive Moving-Window Linear Regression

10.1 Progressive Moving-Window Linear Regression Algorithm

Equation 10-1 gives the result of least-square line-fit of $Y = \alpha + \beta X$ between two set of data $\{Y_j\}$ and $\{X_j\}$, in which j is an integer index to identify (X, Y) pairs in the sets [10].

$$\text{Equation 10-1: } \alpha = \frac{\sum_j Y_j}{\sum_j 1} ; \quad \beta = \frac{\sum_j X_j Y_j \sum_j 1 + \sum_j X_j \sum_j Y_j}{\sum_j X_j^2 \sum_j 1 + \sum_j X_j \sum_j X_j} ;$$

In many applications data set $\{Y_j\}$ is an input data stream collected with fixed rate in time, such as a data stream collected by an ADC [5]. $\{Y_j\}$ is called a time-series input [10], in which j indicates time. A moving window algorithm [10] is performed in a small time-window around each j . For each window of calculation, $\{X_j\}$ can be chosen to be integers in the range of $[-H, +H]$ in which H is an integer constant specifying window's half width, so that $\sum_j X_j = 0$, to reduce

Equation 10-1 into Equation 10-2:

$$\text{Equation 10-2: } \hat{\alpha} = \alpha 2H = \sum_{j=1}^{2H} Y_j ; \quad \hat{\beta} = \beta \frac{H(H+1)(2H+1)}{3} = \sum_{X=-H}^{+H} XY_j ;$$

According to Figure 45, in which H takes an example value of 4, the calculation of $(\hat{\alpha}, \hat{\beta})$ can be obtained from the previous values of $(\hat{\alpha}, \hat{\beta})$, to reduce full calculation of Equation 10-2 into a progressive moving-window calculation of Equation 10-3:

$$\text{Equation 10-3: } \hat{\beta}_j = \hat{\beta}_{j-1} - \hat{\alpha}_{j-1} + (Y_{j-2H-1} + Y_j)H ; \quad \hat{\alpha}_j = \hat{\alpha}_{j-1} - Y_{j-2H-1} + Y_j ;$$

10.2 Dependence Problem in a Progressive Algorithm

Equation 10-3 uses each input multiple times, so it will have dependency problem for all the three uncertainty-bearing arithmetics. The question is how the overestimation of uncertainty evolves with time.

The moving-window linear regression is done on a straight line with a constant slope of exactly $1/1024$ for each advance of time, with a full window width of 9 data points, or $H=4$. Both maximal output value errors and deviations of all three arithmetic increases linearly with input deviations, and increase monotonically with time, e.g., Figure 46 shows such trend for the maximal output value errors using precision arithmetic. Thus both average output error significands and maximal output bounding ratios are largely independent of input precisions. Such independence to input precision is expected for linear algorithms in general [12]. Therefore, only results with input deviation of 10^{-3} are shown for the remaining discussions. Figure 47 shows the output deviation and value errors vs. time while Figure 48 shows the output average error significands and maximal bounding ratios vs. time for all three arithmetics.

For interval and independence arithmetics, the output value errors remain on a constant level, while the output deviations increase with time, so that both output average error significands and maximal bounding ratios decrease with time. The stable linear increase of output deviation with time using either interval arithmetic or independence arithmetic in Figure 47 suggests that the progressive linear regression calculation has accumulated every input uncertainty, which results in the stable linear decrease of average output error significands with time using both arithmetics in Figure 48.

In contrast, while precision arithmetic has slightly larger output deviations than those of independence arithmetic, its output value errors follows its output deviations, so that both its error significands and bounding ratios remain between 0.2 and 0.7. The reason for such increase of output value errors with time is because precision arithmetic does not calculate

infinitively inside uncertainty, and uses larger granularity of values in calculation for larger uncertainty deviation. This mechanism of error tracking in precision arithmetic is demonstrated in Figure 49 and Figure 50. Figure 49 shows that for fewer bits calculated inside uncertainty, output value errors follow output deviation closer in time, but such usage of larger granularity of values in calculation causes the result to become insignificant sooner, while for more bits calculated inside uncertainty, average error significands initially follow the result using independence arithmetic longer, and then follow output deviation for longer duration. The similarity in patterns of average error significands for different bits calculated inside uncertainty using precision arithmetic in Figure 49 suggests that they are all driven by a same mechanism but on different time scale, which is expected when smaller grain of error needs more time to accumulate to a same level. Figure 50 shows that for same bits calculated inside uncertainty, smaller input uncertainty results in longer tracking of output value errors to output deviations, because output deviations increases slower for smaller input deviations. The similar pattern of average error significands is repeated on slower time scale for smaller input deviations in Figure 50, revealing similar underline error-tracking mechanism in both cases. Figure 50 also shows that for same bits calculated inside uncertainty, average error significands deviates from independence at exactly the same time. Figure 49 and Figure 50 thus demonstrate a uniform and granular error tracking mechanism of precision arithmetic.

Is such increase of value errors with the increase of uncertainty deviation using precision arithmetic undesired? First, in real calculations the correct answer is not known, and the reliability of a result depends statistically on the uncertainty of the result, so that there is no reason to assume that calculating more bits inside uncertainty is any better. Conceptually, when the uncertainty of a calculation increases, the value error of the calculation is also expected to increase, which agrees with the trend shown by precision arithmetic. Second, the stability of average output error significands and maximal bounding ratios of precision arithmetic is quite valuable in interpretation results. For example, even the output deviation may have

unexpectedly changed, as in this case if dependency problem were not known and expected, such stability still gives a good estimation of value errors in the result using precision arithmetic. Third, such stability ensures that the result of algorithm at each window does not depend strongly on the usage history of the algorithm, which makes precision arithmetic the only practically usable uncertainty-bearing arithmetic for this progressive algorithm. To test the effect of usage history on each uncertainty-bearing arithmetic, noise is increased by 10-fold at the middle 1/3 duration of the straight line. In Figure 51, the input deviation is also increased by 10-fold to simulate an increase in measured uncertainty, while in Figure 52, the input deviation remains the same to simulate natural deviation away from the straight line. The question is how each uncertainty-bearing arithmetic responses to this change of data in the last 1/3 duration of calculation. Using either independence or interval arithmetic, both average output error significands and maximal output bounding ratios are decrease by about 10-fold in Figure 51 while they are not affected at all in Figure 52. They show extreme sensitivity to usage history. Because real input data are neither controllable nor predictable, the result uncertainty for this progressive algorithm using either interval arithmetic or independence arithmetic may no longer be interpretable. In contrast, using precision arithmetic, both average output error significands and maximal output bounding ratios are relatively stable, while output deviations are sensitivity to usage history, so that the result is still interpretable.

10.3 Choosing a Better Algorithm for Imprecise Inputs

Equation 10-3 has much less calculations than Equation 10-2, and it seems a highly efficient and optimized algorithm according to conventional criterion [10]. However, from the perspective of uncertainty-bearing arithmetic, Equation 10-3 is progressive while Equation 10-2 is expressive, so that Equation 10-2 should be better. Figure 53 and Figure 54 respectively show the output deviations and value errors vs. time for using either Equation 10-3 or Equation 10-2 of a straight line with 10-fold increase of input uncertainty in middle. They show that while the

progressive algorithm carries all historical calculation uncertainty, expressive algorithm is clean from any previous results. For example, at the last 1/3 duration when the moving window is already out of the area for larger input uncertainty, progressive algorithm still gives large result uncertainty, while expressive algorithm gives output result only relevant to the input uncertainty within the moving window. So instead of Equation 10-3, Equation 10-2 is confirmed to be a better solution for this linear regression problem. Figure 55 shows that the result uncertainty of the expressive linear regression can be approximated by a Gaussian distribution.

In Equation 10-3, $\hat{\alpha}_j$ and $\hat{\beta}_0$ has no dependence problem. Each Y_k term in $\hat{\beta}_j$ is calculated progressively by Equation 10-4, which demonstrate quantitatively the uncertainty overestimation in Equation 10-3. According to Equation 10-4, such uncertainty overestimation should increase linearly with time, which is confirmed by Figure 50.

$$\text{Equation 10-4: } Y_k(j-H) = Y_k(j-H+1) - Y_k, \quad 2H < k < j;$$

If such uncertainty overestimation is subtracted from every application of Equation 10-3, it has the same result as that using Equation 10-2. However, such subtraction makes using Equation 10-3 more complicated than using Equation 10-2 directly. Thus, a progressive algorithm may lose all its appeal than its expressive counterpart if uncertainty is considered.

Still, the majority algorithms used today are progressive. Most practical problems are not even mathematical and analytical in nature, so that they may have no expressive solutions. Progressive algorithms are simply just not avoidable in practice. With known expressive counterpart, the progressive moving-window linear regression algorithm can serve as a model progressive algorithm. For example, it shows that the dependency problem of independence and interval arithmetic can manifest as dependency on the usage history of an algorithm. Because of its stability, precision arithmetic should be used generally in progressive algorithms.

11 Conclusion and Discussion

11.1 Summary

The starting point of precision arithmetic is the independent uncertainty assumption, which requires its low-significance input data not to be overwhelmed by systematic errors, and all of its input data not to have messed-up identities. Figure 2 quantifies the statistical requirements for input data to precision arithmetic.

Due to the independent uncertainty assumption, the rounding errors of precision arithmetic are shown to be Gaussian distributed with truncated range. The rounding error distribution is extended to describe uncertainty distribution in general, with the uncertainty deviation of a single precision value given by Equation 3-18, and the result uncertainty deviation of a function given by Equation 3-31 and Equation 3-33.

Equation 5-14 is expected to describe general value error propagation for linear algorithms in which L is the amount of calculations [12], and it is shown to describe general uncertainty deviation propagation in precision arithmetic as well. The average error significances and maximal bounding ratio using precision arithmetic are shown to be independent of input precision, and stable for the amount of calculation for a few very different applications. In contrast, both average error significances and maximal bounding ratio using interval arithmetic are shown to decrease exponentially with the amount of calculations in all tests. Such stability is the major reason why precision arithmetic is better than interval arithmetic in all tests done so far. Equation 3-31 and Equation 3-33 also suggest that precision arithmetic is mathematically simpler than interval arithmetic for expressive algorithms.

The stable average error significances and maximal bounding ratio using precision arithmetic also persist in the algorithms when dependence problem exist. Such stability means that using precision arithmetic the result is still interpretable, and the interpretation of the result

does not depend strongly on the usage history of the arithmetic. Such stability seems to be a result of limited calculation inside uncertainty.

11.2 Improving Precision Arithmetic

The bounding goal of precision arithmetic has rejected the effect of uncertainty on expected value by incorporating value shift due to uncertainty as increase of variance, such as in the case of calculating $f(x) = x^2$. An alternative is to absorb the values shift due to uncertainty into expected values. Another alternative is to provide asymmetrical bounding in the uniform distribution assumption itself. Further study may be needed on this topic.

The convergence property of Equation 3-31 and Equation 3-33 needs to be studied further theoretically, such as their convergence range in term of input variable x when calculating $f(x)=1/x$.

The measured nearly constant values of average error significands of precision arithmetic for each particular algorithm need to be explained theoretically. The optimal bit count to calculate inside uncertainty also needs to be found theoretically. At this moment, precision arithmetic is mostly a pure numerical approach.

Because precision arithmetic is based on generic concepts, it is targeted to be a generic arithmetic for both uncertainty-tracking and uncertainty-bounding. Before applying it generally, precision arithmetic still needs more ground works and testing. However, it seems a worthwhile alternative to interval arithmetic and statistical propagation of uncertainty. It should be tested further in other problems, such as improper integrations, solutions to linear equations, and solutions to differential equations. This paper just provides a proof-of-concept.

11.3 Considering Uncertainty in Mathematics

Uncertainty-tracking arithmetics such as precision arithmetic and interval arithmetic are quite different from conventional arithmetic. Due to dependence problem, uncertainty-bearing

arithmetics have much less operational freedom than conventional arithmetic. Also, the comparison relation in conventional arithmetic needs to be reevaluated in uncertainty-tracking arithmetics.

There may be other conceptual changes required for mathematics in general. For an example, the conventional linear regression algorithm converts the uncertainty associated with each (x, y) position into weights, then assume each (x, y) position is accurate when using least square methods to find a best line fit [10]. It calculated R^2 value to quantify the degree-of-fit, assuming that the fit itself is accurate [10]. In contrast, when the fitting result is uncertain using uncertainty-bearing arithmetic, instead of $Y = \alpha + \beta X$, a better targeted formula for linear regression may be provided by Equation 11-1, which describes a line of uncertain slope β_{YX} crossing a uncertain position (α_X, α_Y) , with degree-of-fit demonstrated directly as uncertainty of β_{YX} and (α_X, α_Y) respectively. Equation 11-1 is also independent of coordinate system because offset may no longer be a valid arithmetic operation in an actual implementation of uncertainty-bearing arithmetic. It would be better if the new mathematics can quantify the degree-of-fit separately in terms of (1) the uncertainty in the input data set, and (2) the deviation of mean position of the input data set from the mean of the fitting result.

Equation 11-1: $(Y - \alpha_Y) = \beta_{YX}(X - \alpha_X) ;$

11.4 Acknowledge

As an independent researcher, the author of this paper feels indebted to encouragements and valuable discussions with Dr. Zhong Zhong from Brookhaven National Laboratory, the organizers of AMCS 2005 with Prof. Hamid R. Arabnia from University of Georgia in particular, and the organizers of NKS Mathematica Forum 2007 with Dr. Stephen Wolfram in particular.

12 Reference

- [1] Sylvain Ehrenfeld and Sebastian B. Littauer, Introduction to Statistical Methods (McGraw-Hill, 1965)
- [2] John R. Taylor, Introduction to Error Analysis: The Study of Output precisions in Physical Measurements (University Science Books, 1997)
- [3] <http://physics.nist.gov/cuu/Constants/bibliography.html>
- [4] Michael J. Evans and Jeffrey S. Rosenthal, Probability and Statistics: The Science of Uncertainty (W. H. Freeman 2003)
- [5] Paul Horowitz and Hill Winfield, Art of Electronics (Cambridge Univ Press, 1995)
- [6] John P Hayes, Computer Architecture (McGraw-Hill, 1988).
- [7] David Goldberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic, Computing Surveys, http://docs.sun.com/source/806-3568/ncg_goldberg.html
- [8] Institute of Electrical and Electronics Engineers, ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic, 1985.
- [9] Kulish, U., Miranker, W.M., The Arithmetic of digital computers: A new approach, SIAM Rev., Vol 28, (1), (1986).
- [10] William H. Press, Saul A Teukolsky, William T. Vetterling, and Brian P. Flannery, Numerical Recipes in C (Cambridge University Press, 1992)
- [11] Oliver Aberth, Precise Numerical Methods using C++, Academic press, 1998
- [12] Gregory L. Baker and Jerry P. Gollub, Chaotic Dynamics: An Introduction (Cambridge University Press, 1990)
- [13] J. Vignes, A stochastic arithmetic for reliable scientific computation, Mathematics and Computers in Simulation vol. 35, pp 233-261, 1993
- [14] B. Liu, T. Kaneko, Error analysis of digital filters realized with floating-point arithmetic, Proc. IEEE, vol. 57, pp 1735-1747, Oct. 1969

- [15] B. D. Rao, Floating-point arithmetic and digital filters, IEEE, Trans. Signal Processing, vol 40, pp 85-95, Jan, 1992
- [16] R.E. Moore, Interval Analysis, Prentice Hall, 1966
- [17] W. Kramer, A prior worst case error bounds for floating-point computations, IEEE Trans. Computers, vol. 47, pp 750-756, Jul. 1998
- [18] R. Alt and J.-L. Lamotte, Some experiments on the evaluation of functional ranges using a random interval arithmetic, Mathematics and Computers in Simulation, 2001, Vol. 56, pp. 17–34.
- [19] J. Stolfi, L. H. de Figueiredo, An introduction to affine arithmetic, TEMA Tend. Mat. Apl. Comput., 4, No. 3 (2003), 297-312.
- [20] http://amp.ece.cmu.edu/Publication/Fang/icassp2003_fang.pdf
- [21] W. Krämer, Generalized Intervals and the Dependency Problem, Proceedings in Applied Mathematics and Mechanics, Vol. 6, Nr. 1, p. 685-686, Wiley-InterScience (2006).
- [22] Propagation of uncertainty, http://en.wikipedia.org/wiki/Propagation_of_uncertainty.
- [23] Significance Arithmetic, http://en.wikipedia.org/wiki/Significance_arithmetic.
- [24] Max Goldstein, Significance Arithmetic on a Digital Computer, Communications of the ACM, Volume 6 (1963), Issue 3, Page 111-117
- [25] R. L. Ashenhurst and N. Metropolis, Unnormalized Floating-point Arithmetic, Journal of the ACM, Volume 6 (1959), Issue 3, Page 415-428
- [26] Chengpu Wang, Error Estimation of Floating-point Calculations by a New Floating-point Type That Tracks the Errors, AMCS'05
- [27] Feldstein A., Goodman R. Convergence Estimates for the Distribution of Trailing Digits, Journal of Association for Computing Machinery, Vol 23, (2), (1976), 287 - 297.
- [28] Integrating The Bell Curve, <http://www.mathpages.com/home/kmath045/kmath045.htm>
- [29] C. Pennachin, M. Looks, João A. de Vasconcelos, Robust Symbolic Regression with Affine Arithmetic, Genetic and Evolutionary Computation Conference (GECCO), 2010,

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/36294.pdf

- [30] Parity of a permutation, Wikipedia, http://en.wikipedia.org/wiki/Parity_of_a_permutation
- [31] Invertible matrix, Wikipedia, http://en.wikipedia.org/wiki/Invertible_matrix
- [32] Adjuate matrix, Wikipedia, http://en.wikipedia.org/wiki/Adjuate_matrix
- [33] Condition number, Wikipedia, http://en.wikipedia.org/wiki/Condition_number
- [34] Digital Signal Processor, Wikipedia, http://en.wikipedia.org/wiki/Digital_signal_processor.
- [35] Taylor's theorem, Wikipedia, http://en.wikipedia.org/wiki/Taylor_polynomial

13 Tables

\sim_1 VS. \sim_2	$\sim_1 = \sim_2$	$\sim_1 \neq \sim_2$				
		$\sim_1 = ?$	$\sim_2 = ?$	$R_1 > R_2$	$R_1 < R_2$	$R_1 = R_2$
\sim	\sim_1	\sim_2	\sim_1	\sim_1	\sim_2	$?$

Table 1: Result \sim in $S_1 \sim_1 R_1 @ E + S_2 \sim_2 R_2 @ E = (S_1 + S_2) \sim (R_1 + R_2) @ E$.

\sim_1 VS. \sim	$\sim_1 = \sim_2$				$\sim_1 \neq \sim_2$	
	$\sim_1 = ?$	$R_1 > R_2$	$R_1 < R_2$	$R_1 = R_2$	$\sim_1 = ?$	$\sim_1 \neq ?$
\sim	$?$	\sim_1	\sim_2	$?$	\sim_2	\sim_1

Table 2: Result \sim in $S_1 \sim_1 R_1 @ E - S_2 \sim_2 R_2 @ E = (S_1 - S_2) \sim (R_1 + R_2) @ E$.

R_{\max}	16	64	256
Normalized bounding range	$[4.9\sigma, 9.8\sigma)$	$[9.8\sigma, 19.6\sigma)$	$[19.6\sigma, 39.2\sigma)$
Maximal normalization leakage	10^{-6}	10^{-22}	10^{-85}
Bit calculated inside uncertainty	0	1	2
$0.5 \pm 0.001 =$	512?6.29@-10	1024?25.16@-11	2048?100.64@-12
$1 \pm 0.001 =$	1024?6.29@-10	2048?25.16@-11	4096?100.64@-12
$1 \pm 0.002 =$	512?6.29@-9	1024?25.16@-10	2048?100.64@-11

Table 3: Characterization of precision arithmetic with different R_{\max} for precision representation S~R@E

χ	0	1	2
$0.5 \pm 0.001 =$	512?6.29@-10	1024?12.58@-11	2048?25.16@-12
$1 \pm 0.001 =$	1024?6.29@-10	2048?12.58@-11	4096?25.16@-12
$1 \pm 0.002 =$	512?6.29@-9	1024?12.58@-10	2048?25.16@-11

Table 4: Examples of precision arithmetic with different χ for $R_{\max}=16$, in which χ stands for bits calculated inside uncertainty.

Power n	$\int_{10^{-6}}^4 x^n dx$ Deviation	Search Depth	$\int_{10^{-6}}^4 x^n dx - \frac{4^{n+1} - 10^{-6(n+1)}}{n+1}$ Value
2	1.32×10^{-14}	[25, 47]	-0.705×10^{-14}
3	2.52×10^{-14}	[25, 47]	-1.42×10^{-14}
4	1.16×10^{-13}	[26, 47]	-1.13×10^{-13}
5	5.08×10^{-13}	[26, 48]	-6.82×10^{-13}
6	1.92×10^{-12}	[26, 48]	-2.72×10^{-12}

Table 5: Error and uncertainty of numerical integration vs. expected results using precision arithmetic

14 Figures

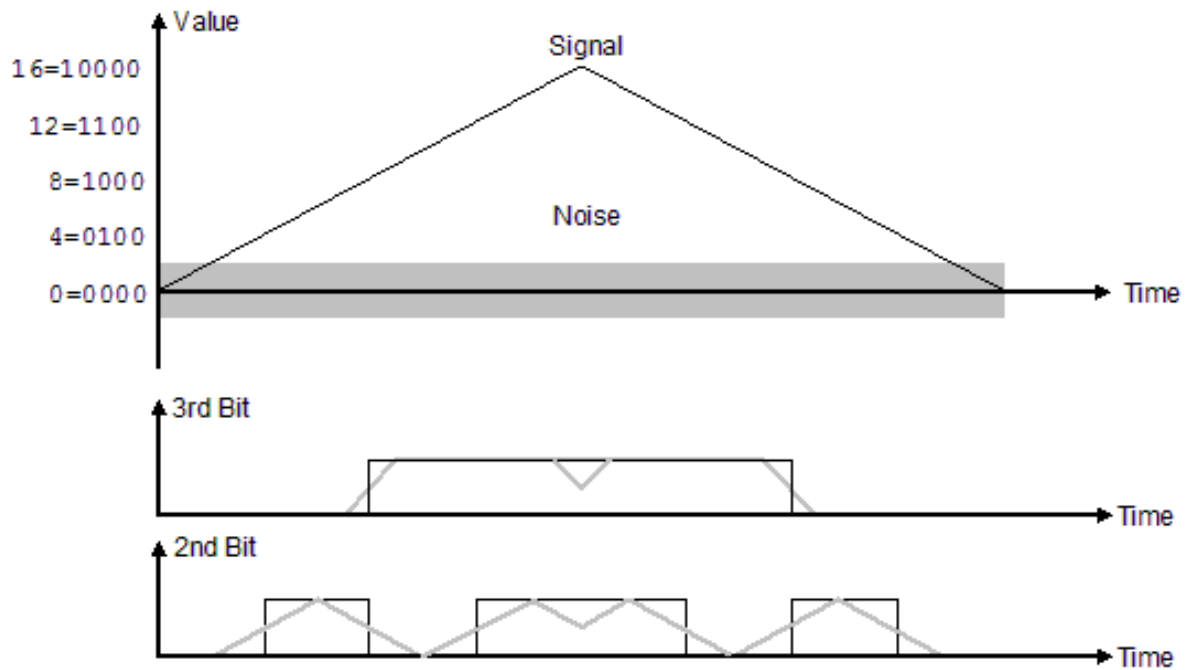


Figure 1: Effect of noise on bit values of a measured value. The triangular wave signal and the added white noise are shown at top. The values are measured by a theoretical 4-bit DAC in ideal condition, assuming LSB is the 0th bit. The measured 3rd and 2nd bits without the added noise are shown using thin black lines, while the mean values of the measured 3rd and 2nd bits with the added noise are shown using thin grey lines.

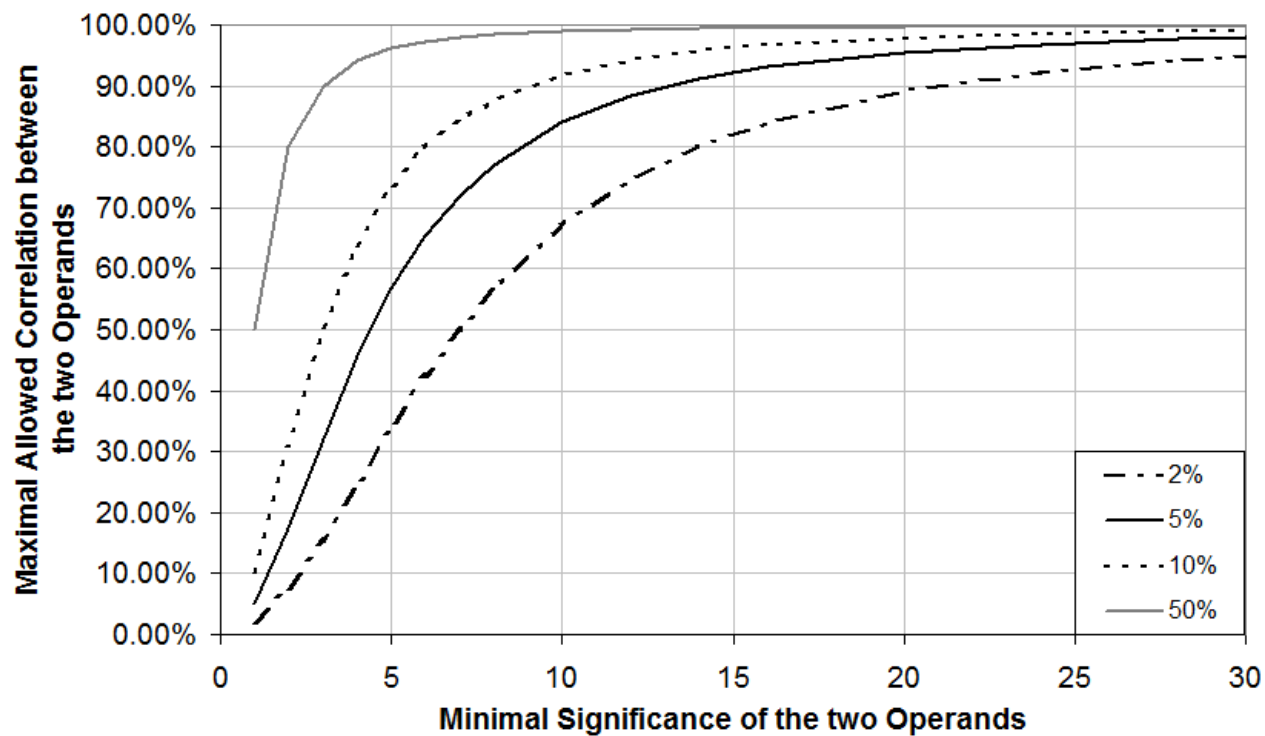


Figure 2: Allowed maximal correlation for precision arithmetic between two identities vs. input significances and independence standard (as shown in legend) for the independence uncertainty assumption of precision arithmetic.

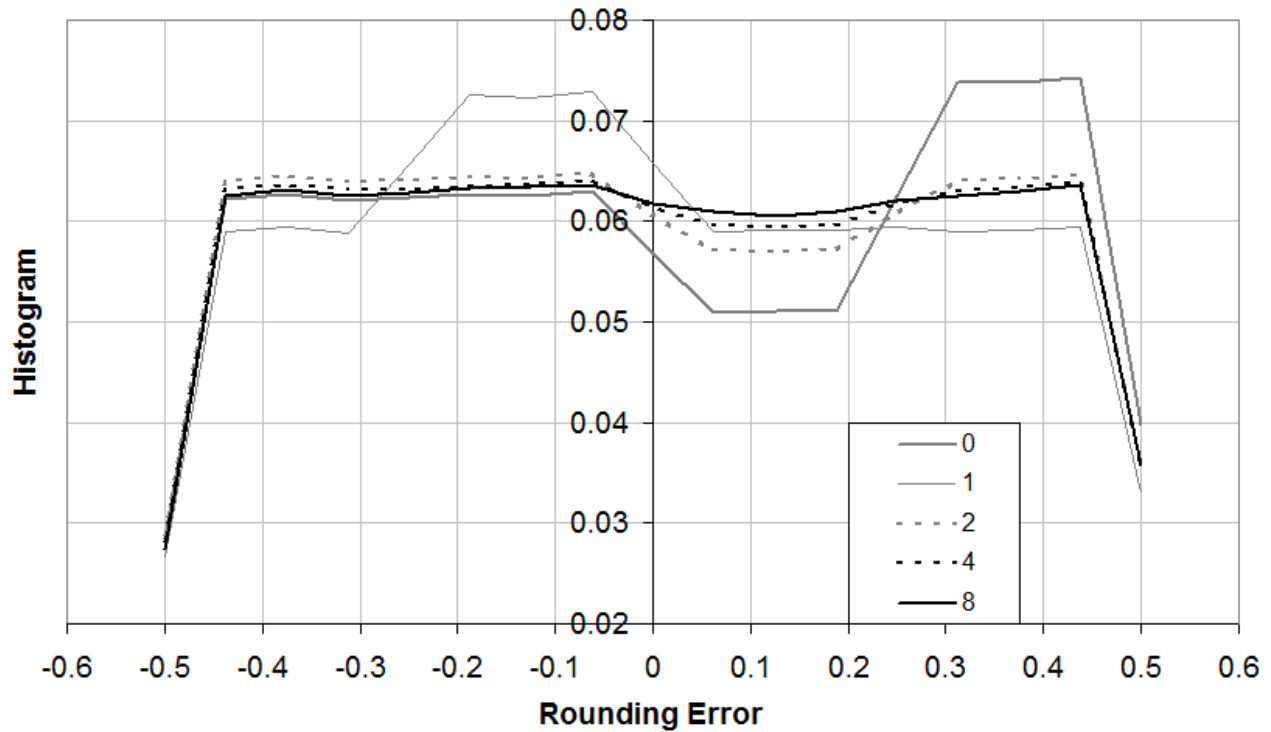


Figure 3: Measured probability distribution of rounding errors of precision round-up rule for the minimal significant thresholds 0, 1, 2, 4, and 8 respectively. Mathematically the probability should be either defined in range $(-1/2, +1/2]$ or in range $[-1/2, +1/2)$, but not in range $[-1/2, +1/2]$. The counting of histogram in range $[-1/2, +1/2]$ introduces the artificially smaller probability density in the sections near $+0.5$ and -0.5 .

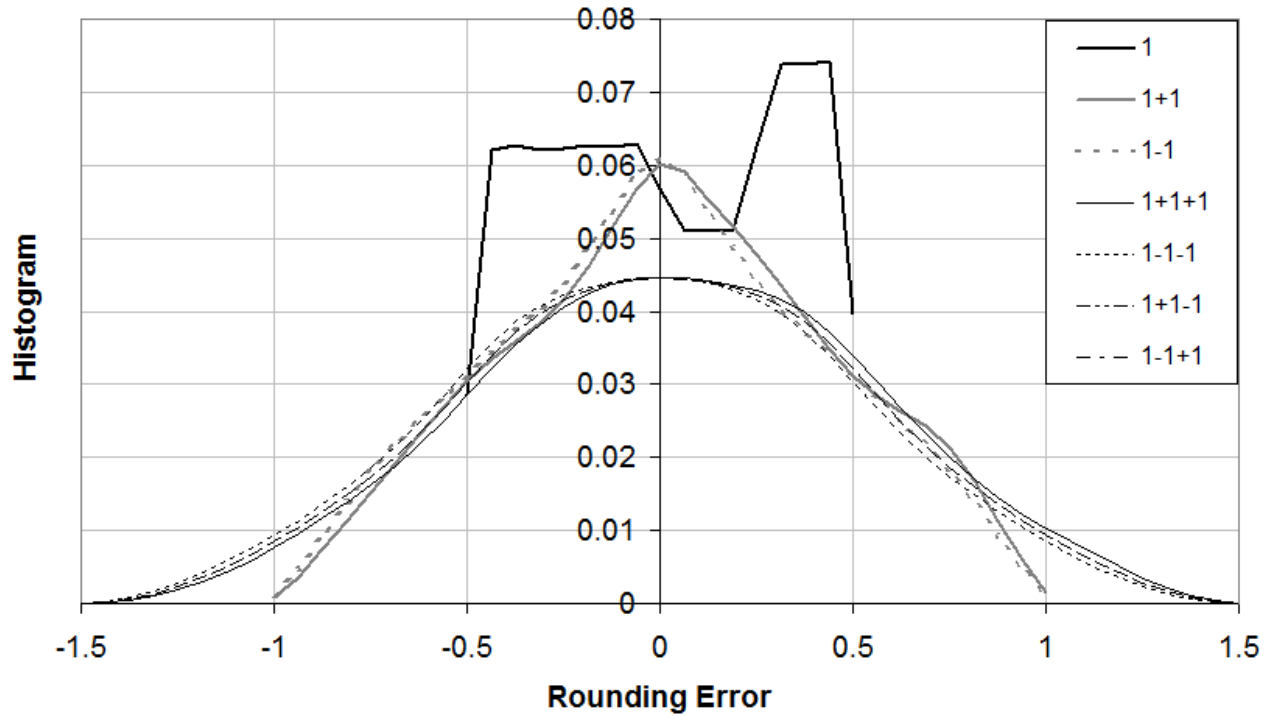


Figure 4: Measured probability distribution of rounding error after addition and subtraction. In the legend, “1” for measured rounding error distribution for the minimal significand thresholds 0, “1+1” for addition once and “1-1” for subtraction once using the rounding error distribution of “1”, while “1+1+1” for addition twice, “1-1-1” for subtraction twice, “1+1-1” for addition once then subtraction once, and “1-1+1” for subtraction once then addition once.

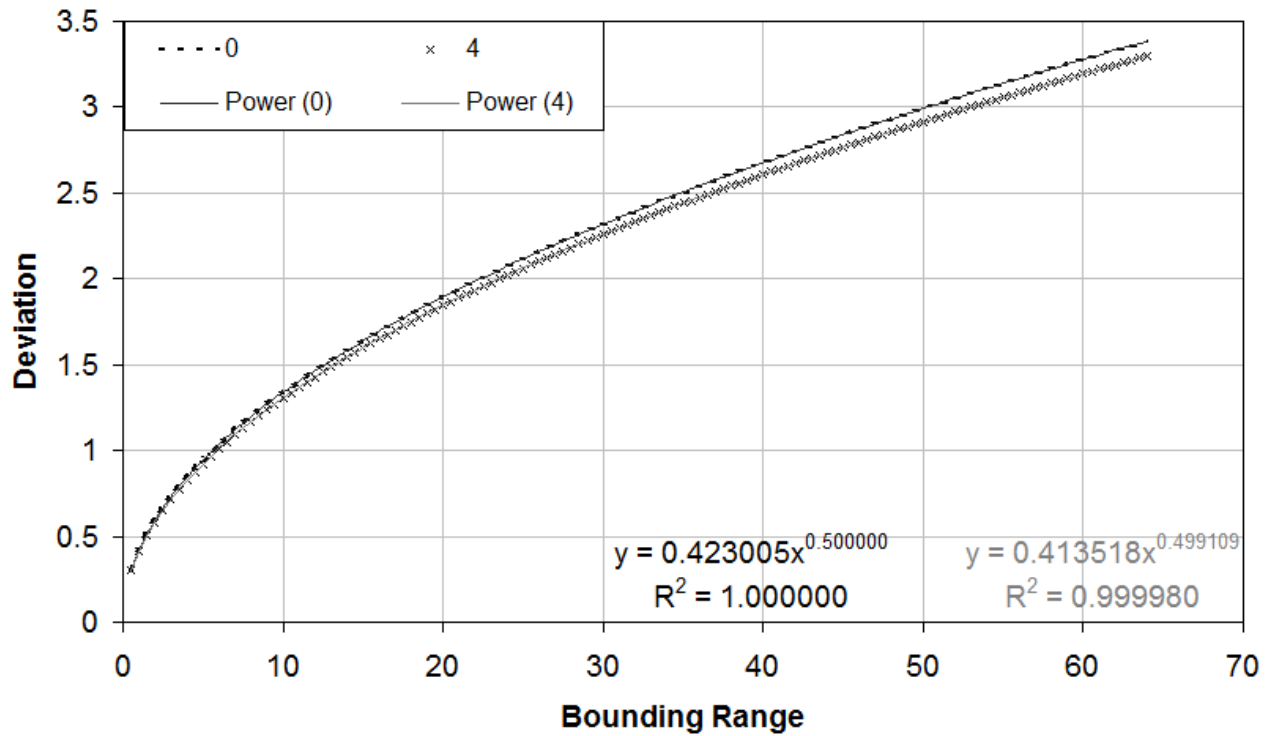


Figure 5: Measured deviations vs. bounding ranges of measured rounding error distributions of different minimal significant thresholds (as shown in legend). The relation of deviation σ vs. bounding R is fitted by $\sigma = \alpha R^\beta$, in which α and β are empirical fitting parameters. The fitting results as well as fitting reliable factor R^2 are displayed in color matching the corresponding legend.

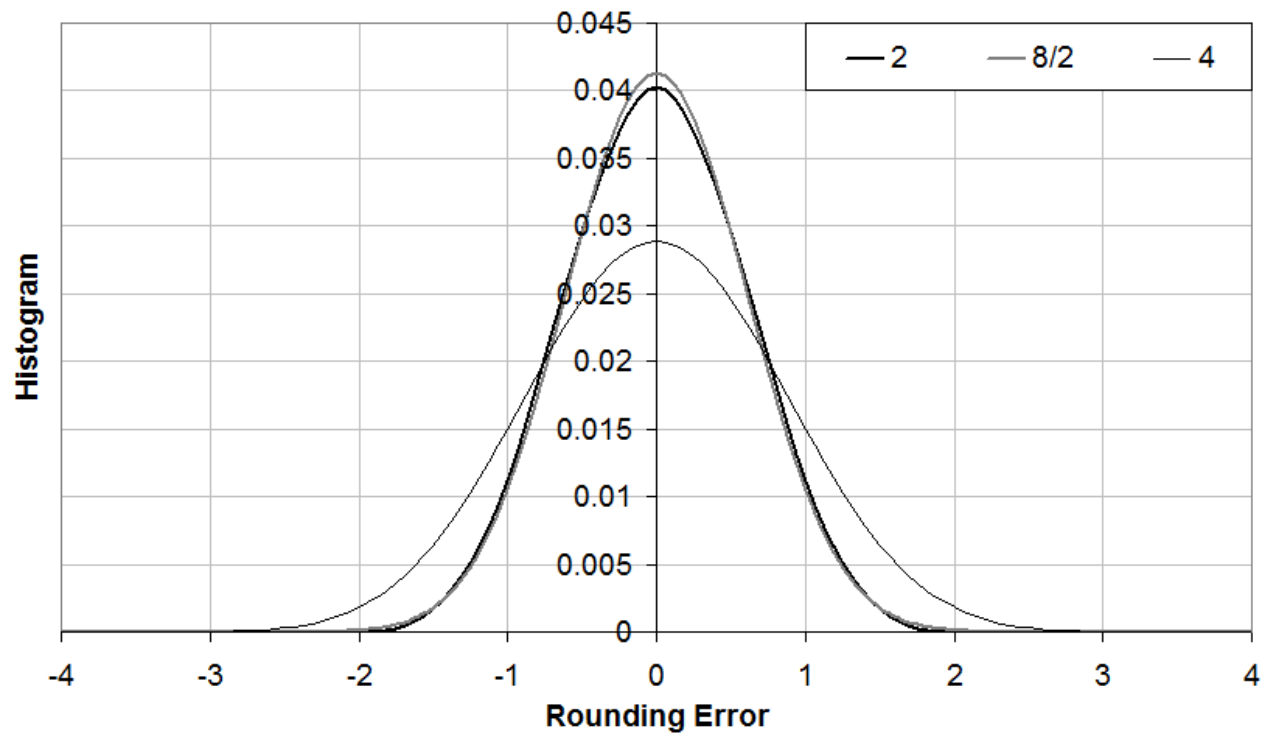


Figure 6: The result rounding error distribution $R=8/2$ after the original error distribution $R=8$ is rounded up once. The $R=8/2$ distribution is compared with the $R=4$ distribution and the $R=2$ distribution, which have same bounding range and deviation respectively.

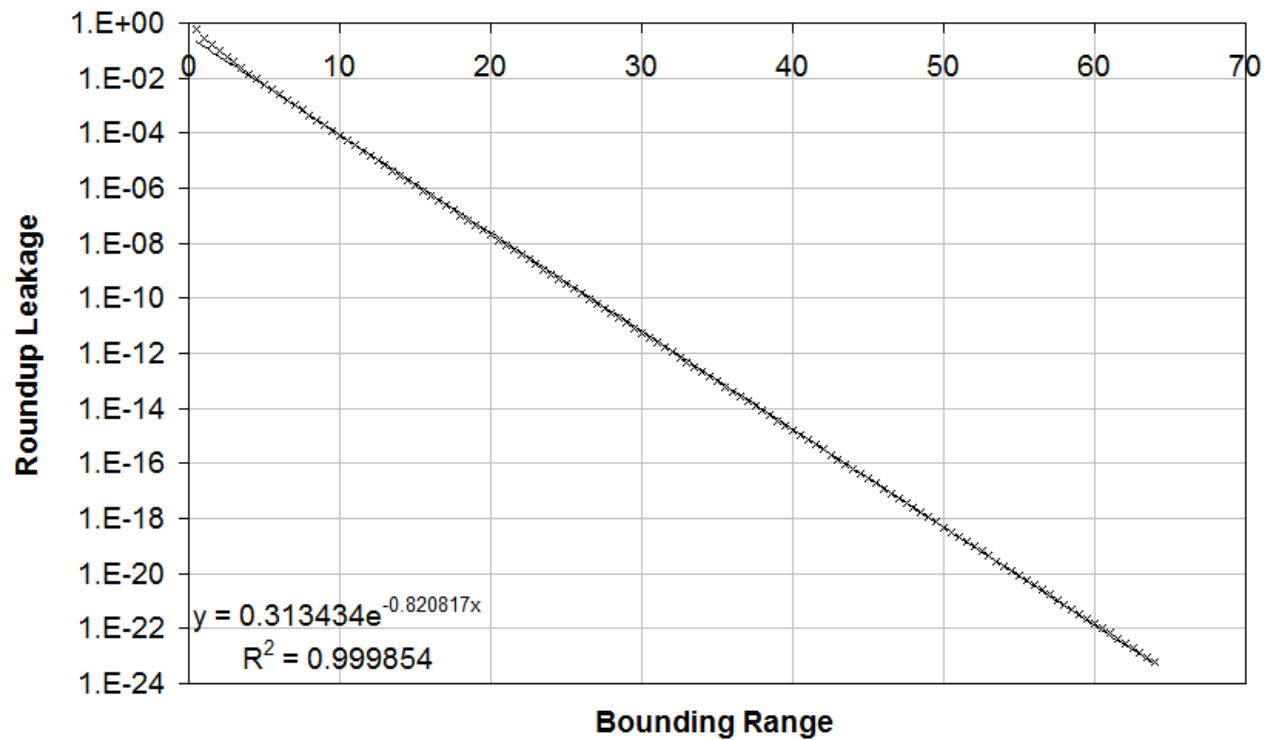


Figure 7: Roundup leakage vs. bounding ranges after rounding up by deviation once.
 The bounding range in the chart is the bounding range before rounding up by deviation.
 The roundup leakage is fitted by an exponential relation, which is displayed in the chart.

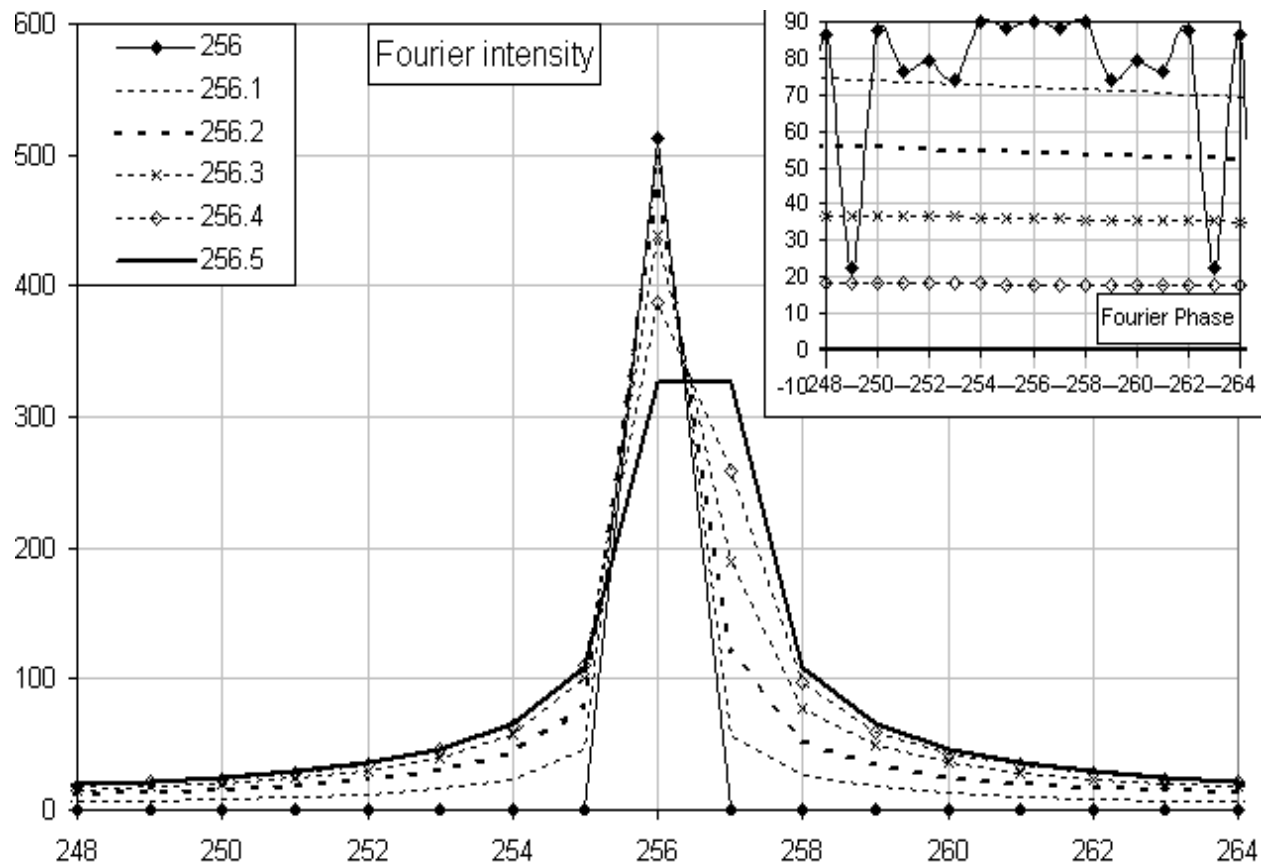


Figure 8: Unfaithful representations of perfect sine signals in discrete Fourier transformation. The calculation is done on 1024 samples using FFT on a series of perfect sine signals having amplitude of 1 and slightly different frequencies as shown in legends. In the drawing, x axis shows the frequency, y axis shows either intensity or phase (inlet). A faithful representation at 256 is also included for comparison, whose phase is 90° at 256, and undetermined at other frequencies.

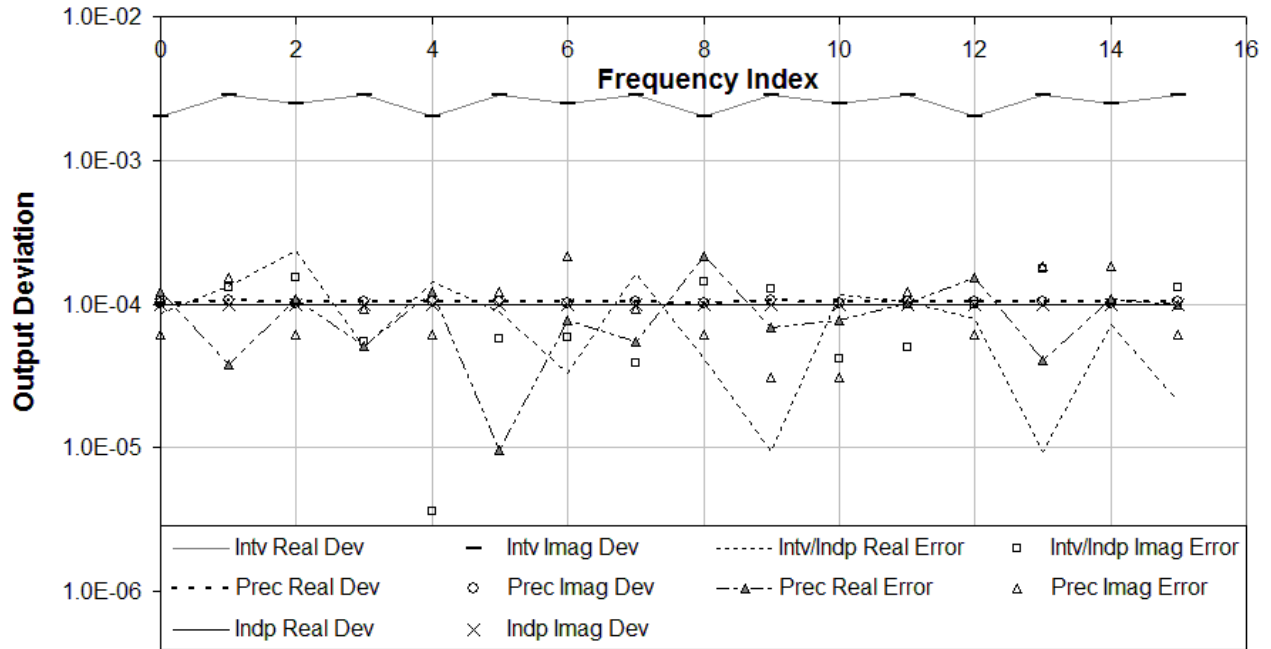


Figure 9: The output deviations and value errors of forward FFT on a noisy signal of FFT order 4, index frequency 1 and input deviation 10^{-4} . In the legend, “Intv” means interval arithmetic, “Indp” means independence arithmetic, “Prec” means precision arithmetic, “Dev” means output uncertainty deviations, “Error” means output value errors, “Real” means real part, and “Imag” means imaginary part. Because both interval arithmetic and independence arithmetic using conventional floating arithmetic for underline calculations, they have the same value errors.

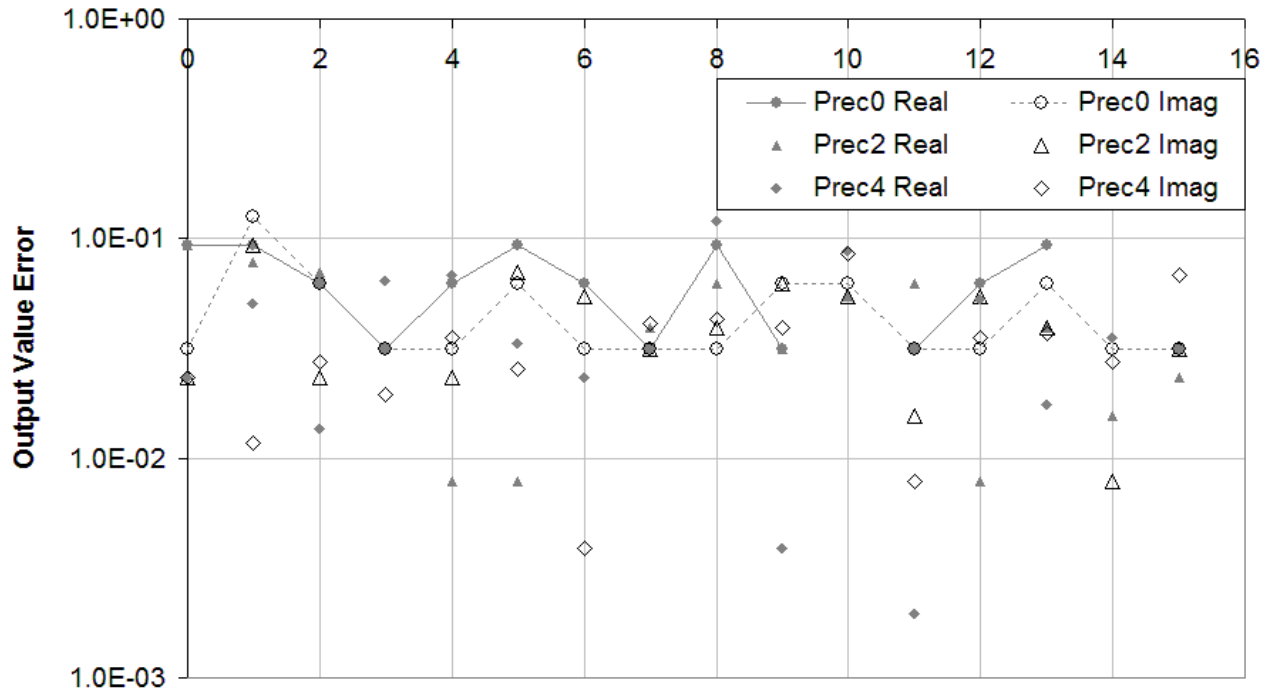


Figure 10: The output value errors of forward FFT on a noisy signal of index frequency 1 and input deviation 10^{-2} using precision arithmetic with different bit inside uncertainty. In the legend, “Prec0” means precision arithmetic with no calculated inside uncertainty, “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty, and “Prec4” means precision arithmetic with 4-bit calculated inside uncertainty.

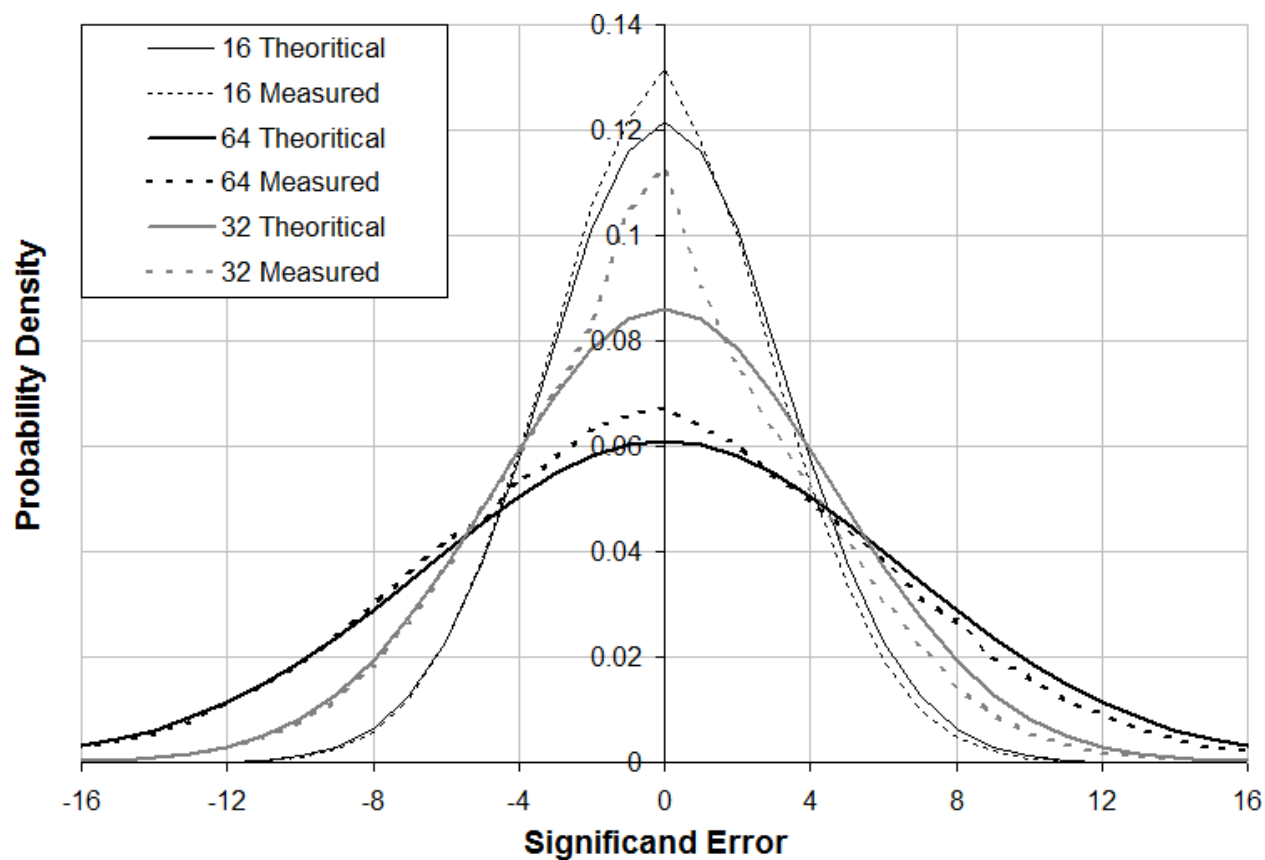


Figure 11: The theoretical and empirical distribution of significant errors for different bounding ranges (as shown in legend) using precision arithmetic with 2-bit calculated inside uncertainty for forward FFT on noisy sine signals.

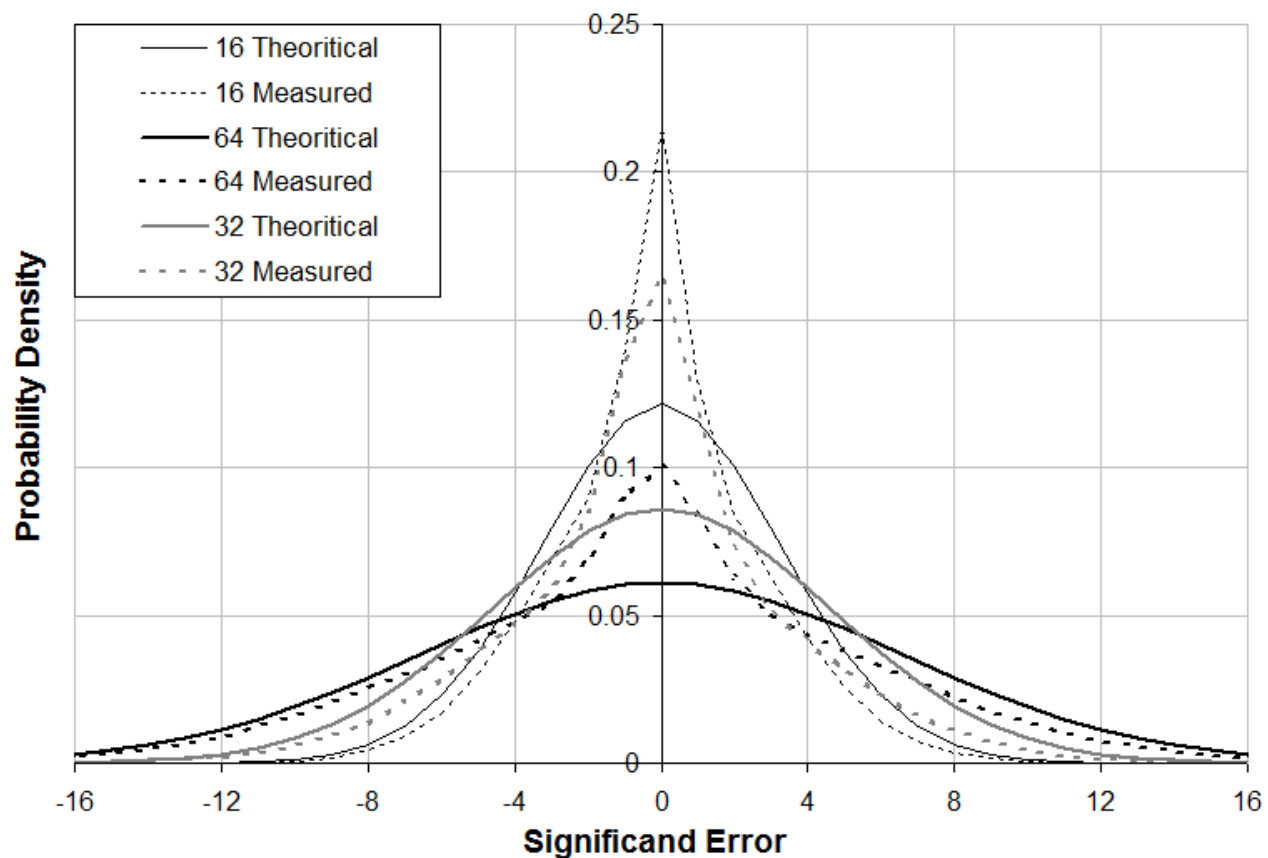


Figure 12: The theoretical and empirical distribution of significant errors for different bounding ranges (as shown in legend) using precision arithmetic with 2-bit calculated inside uncertainty for reverse FFT on noisy sine signals.

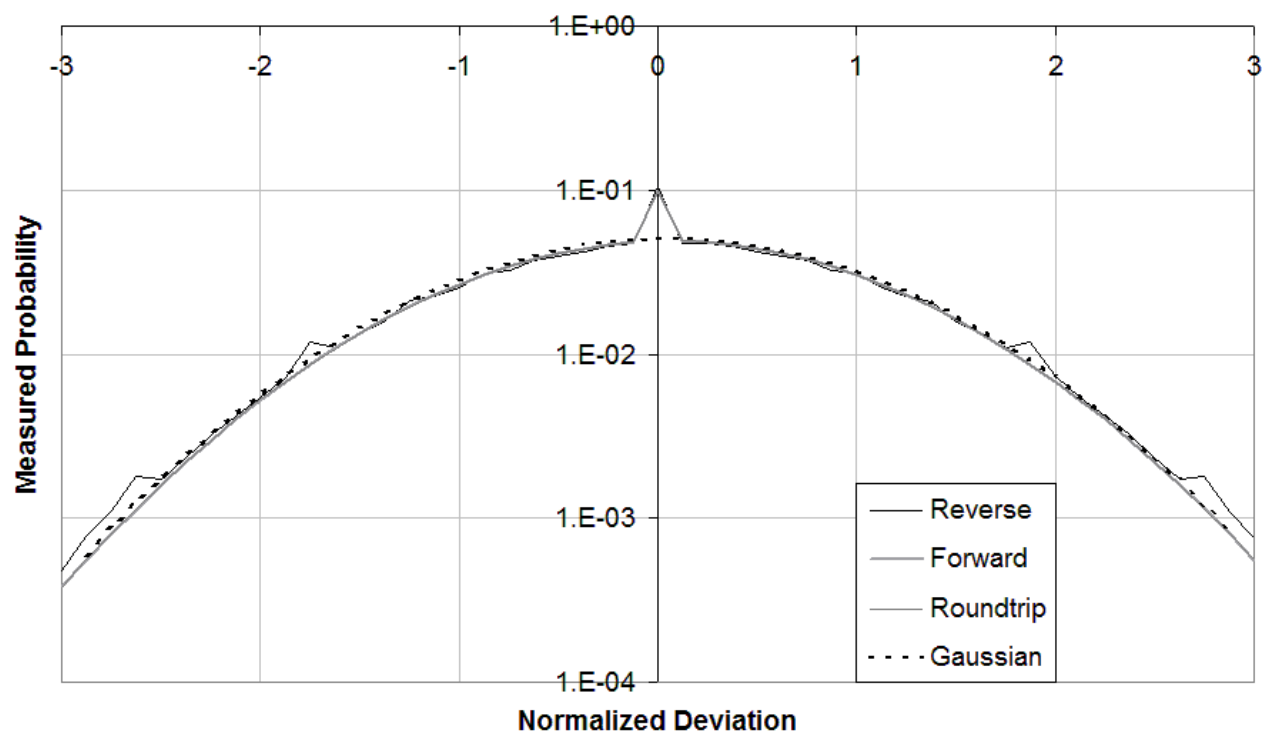


Figure 13: The measured normalized value error distributions using independence arithmetic for FFT algorithms (as shown in legend). They are best fitted by a Gaussian distribution with mean of 0.06 and deviation of 0.98.

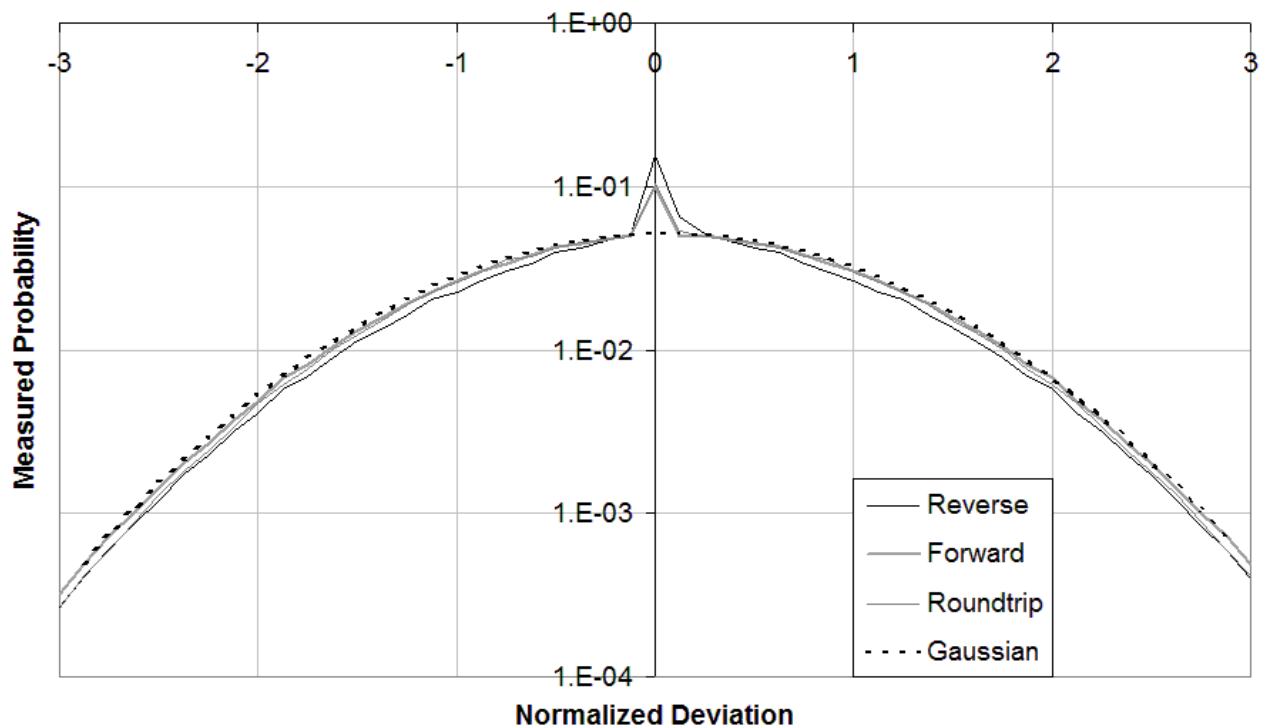


Figure 14: The measured normalized value error distributions using precision arithmetic with 4-bit calculated inside uncertainty for FFT algorithms (as shown in legend). They are best fitted by a Gaussian distribution with mean of 0.06 and deviation of 0.97.

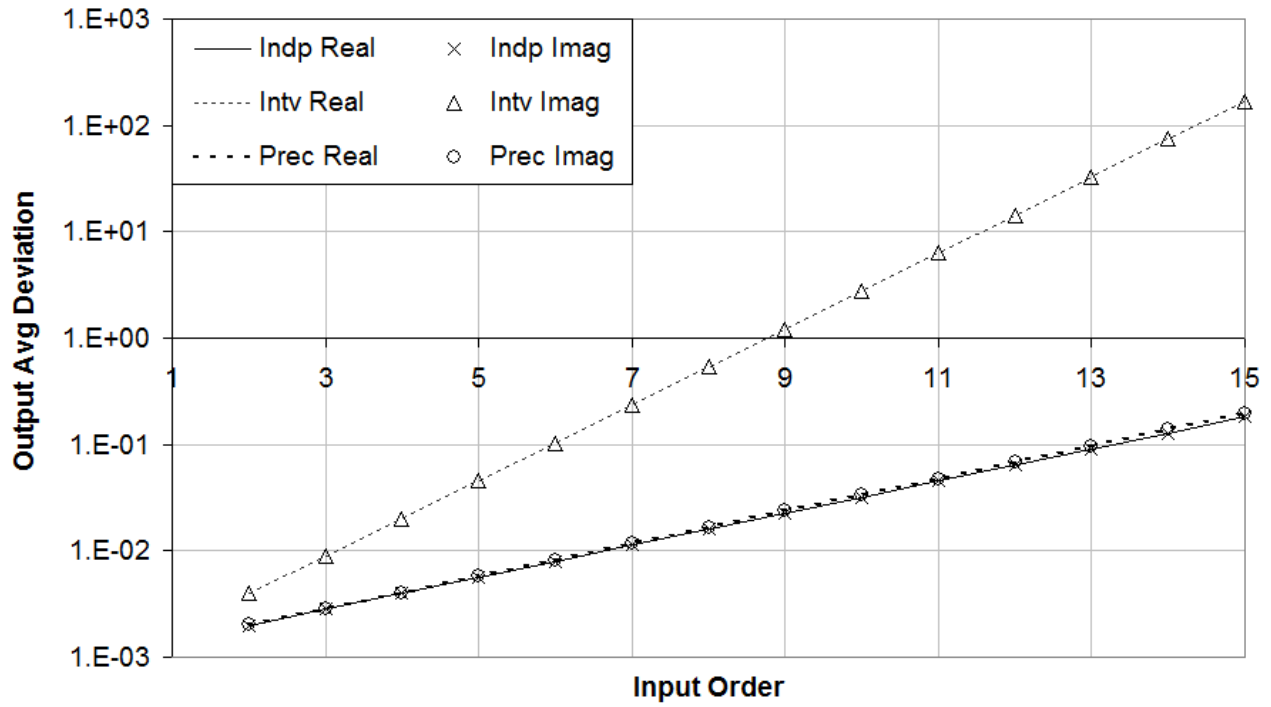


Figure 15: For a same input deviation of 10^{-3} , the average output deviations of forward FFT increase exponentially with FFT order for all uncertainty-bearing arithmetics. In the legend, “Intv” means interval arithmetic, “Indp” means independence arithmetic, “Prec” means precision arithmetic, “Real” means real part, and “Imag” means imaginary part.

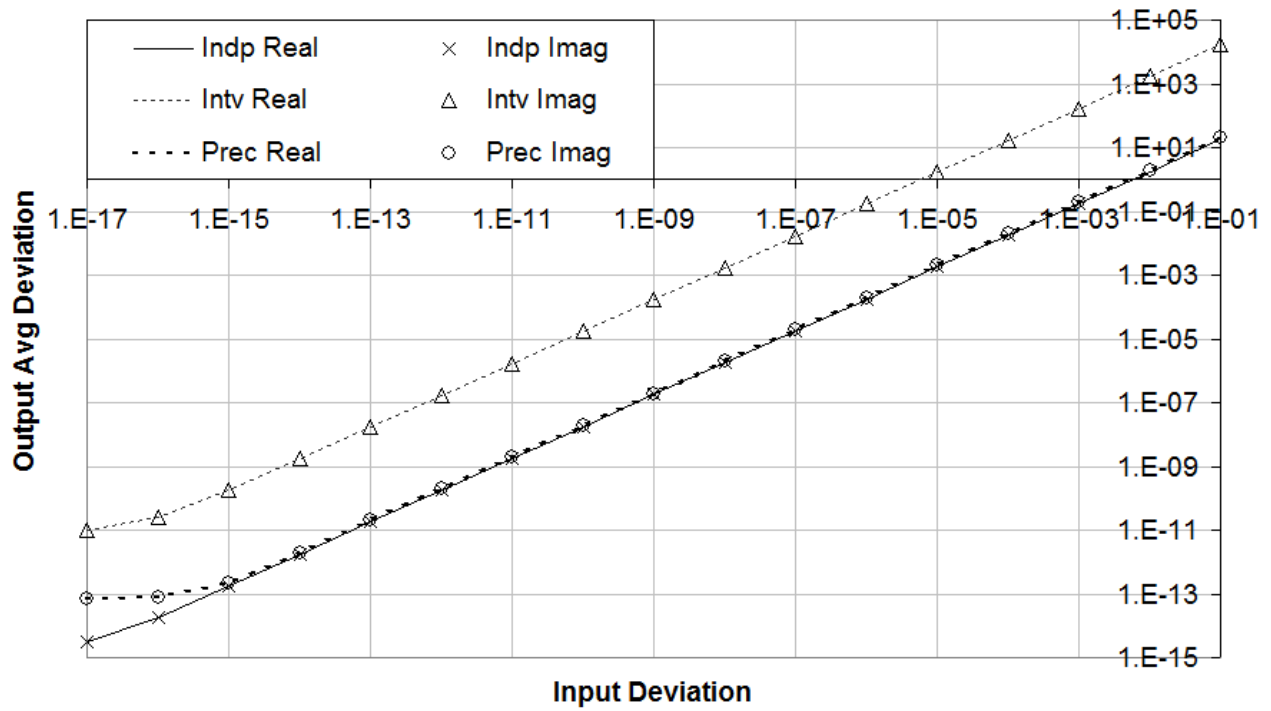


Figure 16: For a same order of FFT calculation of 15, the average output deviations of forward FFT increases linearly with input deviation for all uncertainty-bearing arithmetics.

In the legend, “Intv” means interval arithmetic, “Indp” means independence arithmetic, “Prec” means precision arithmetic, “Real” means real part, and “Imag” means imaginary part.

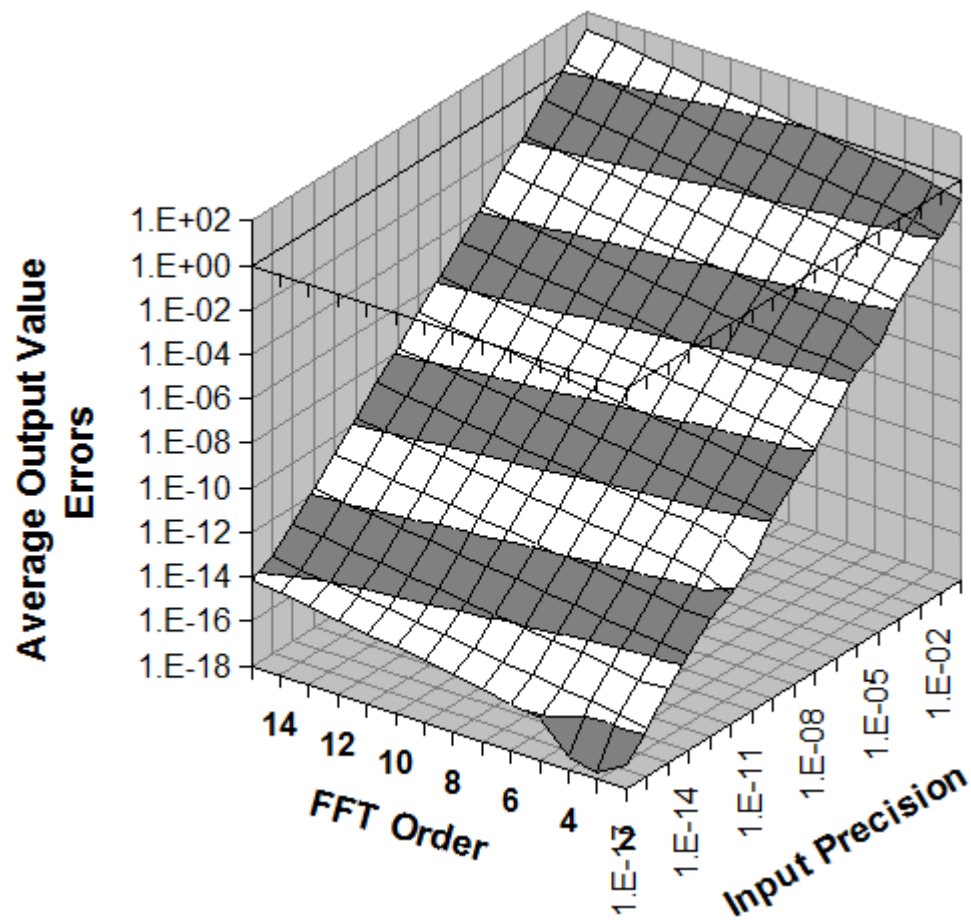


Figure 17: The average output value errors using precision arithmetic increase exponentially with FFT order and linearly with input deviation respectively.

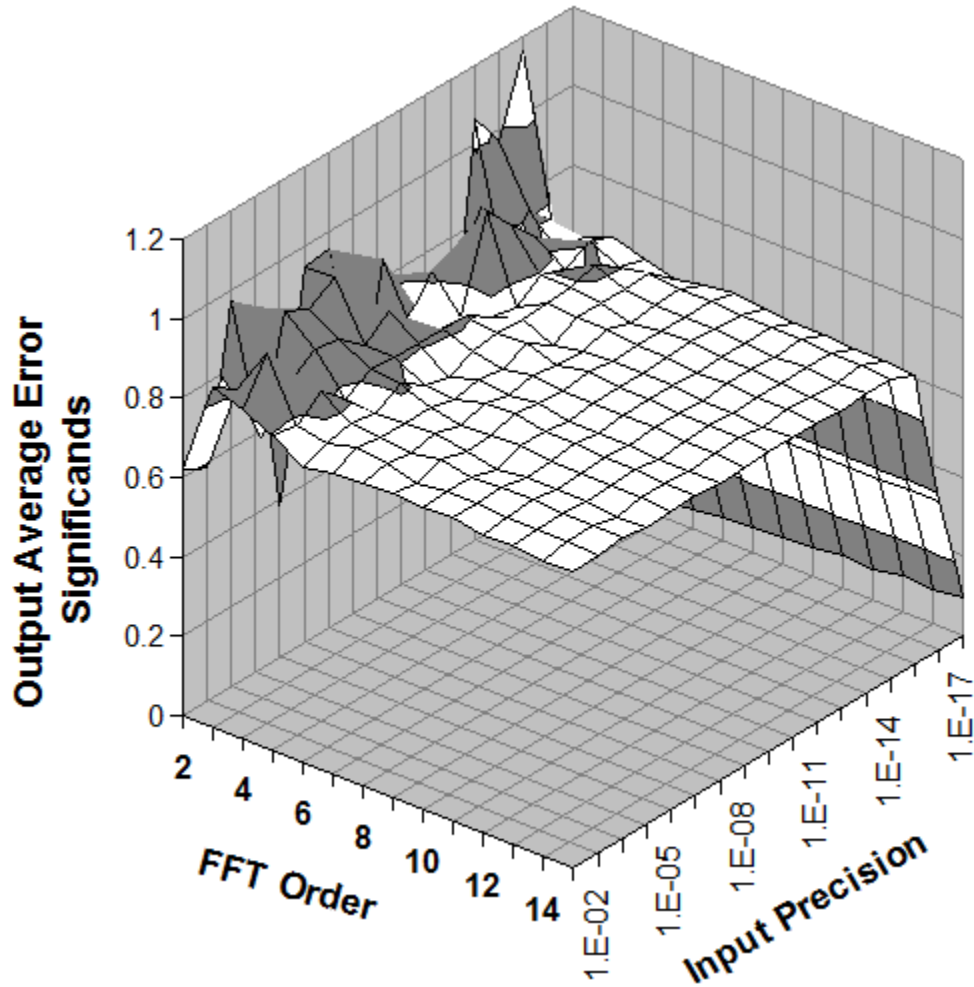


Figure 18: The average output error significands using precision arithmetic is a constant when the input deviation is larger than 10^{-14} and the FFT order is more than 5 for forward FFT algorithms. Because the precision of conventional floating-point representation is at 10^{-16} , adding Gaussian noises with deviation of 10^{-17} should have little effect on the input data. For the same reason, the output error significands are stable only when the input deviation is more than 10^{-14} . When FFT order is 2, a FFT calculation actually involves no arithmetic calculation between input data. For the same reason, when FFT order is less than 5, there is not enough calculation for the result error significands to reach equilibrium.

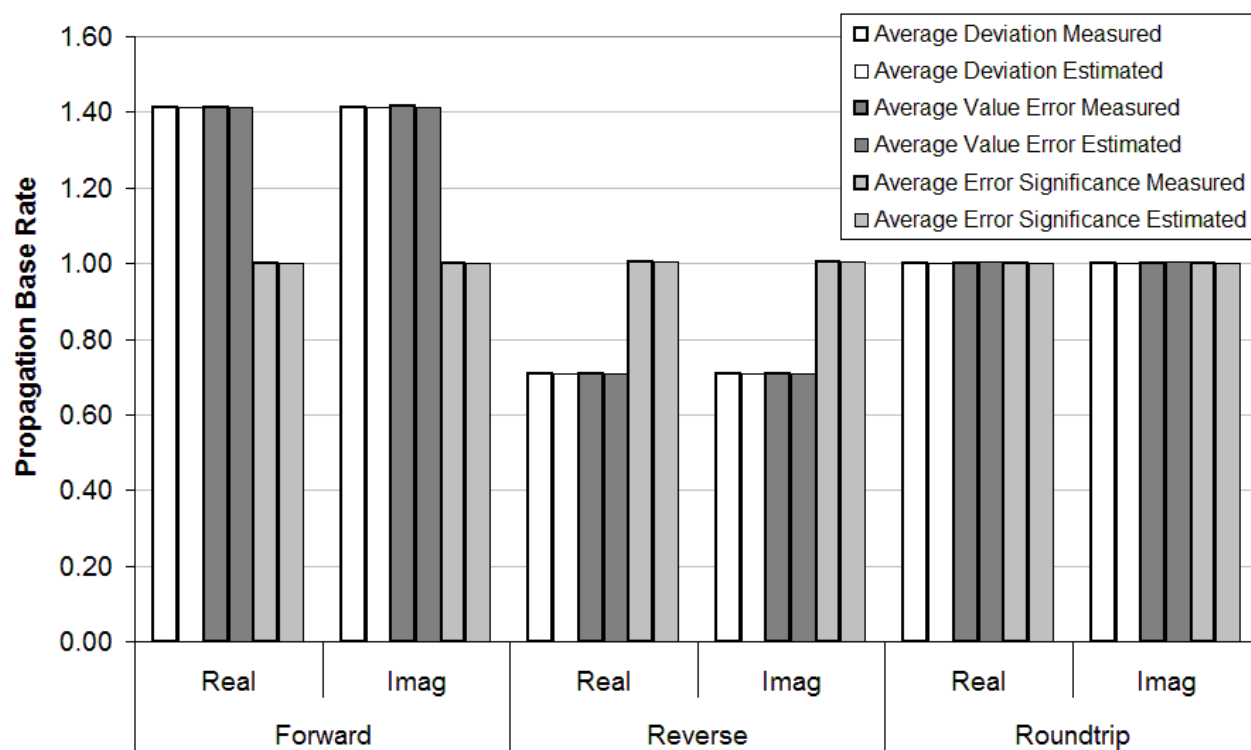


Figure 19: Empirical and theoretical β for fitting average output deviations, value errors and error significands for forward, reverse and roundtrip FFT using independence arithmetic on noisy sine signals. In the chart, “Real” means real part, and “Imag” means imaginary part.

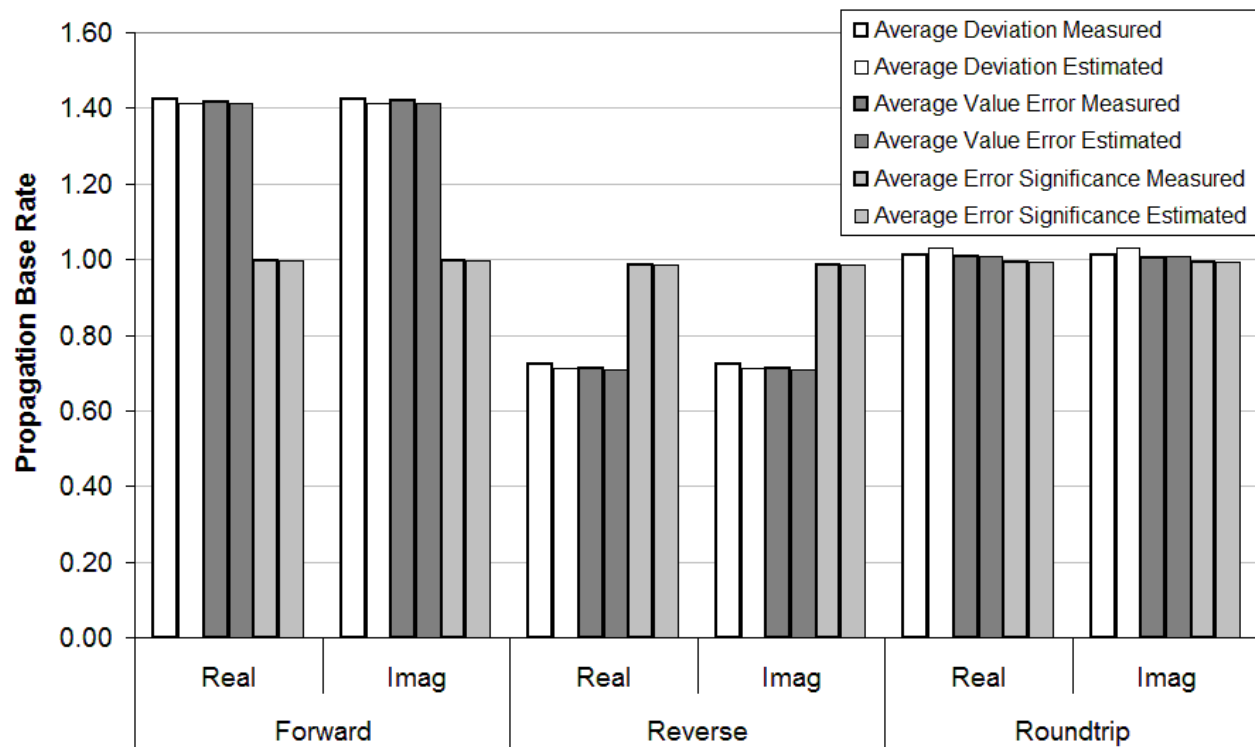


Figure 20: Empirical and theoretical β for fitting average output deviations, value errors and error significands for forward, reverse and roundtrip FFT using precision arithmetic on noisy sine signals. In the chart, “Real” means real part, and “Imag” means imaginary part.

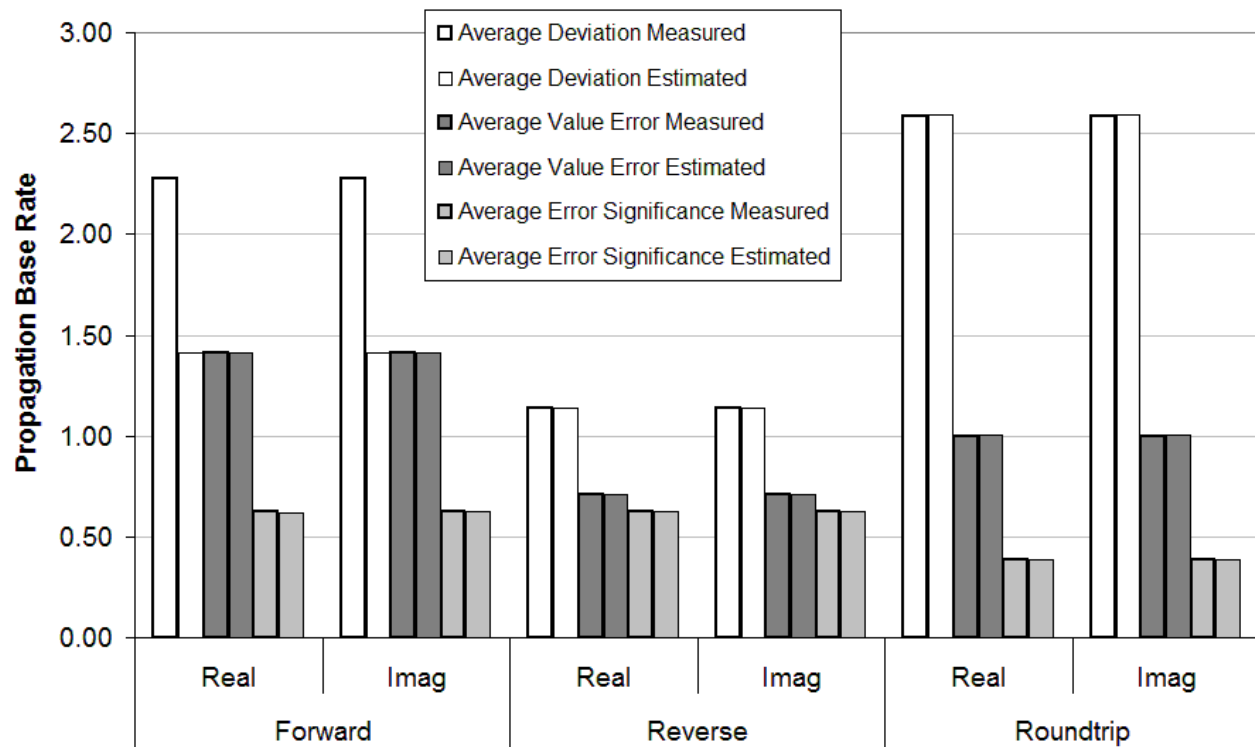


Figure 21: Empirical and theoretical β for fitting average output deviations, value errors and error significands for forward, reverse and roundtrip FFT using interval arithmetic on noisy sine signals. In the chart, “Real” means real part, and “Imag” means imaginary part.

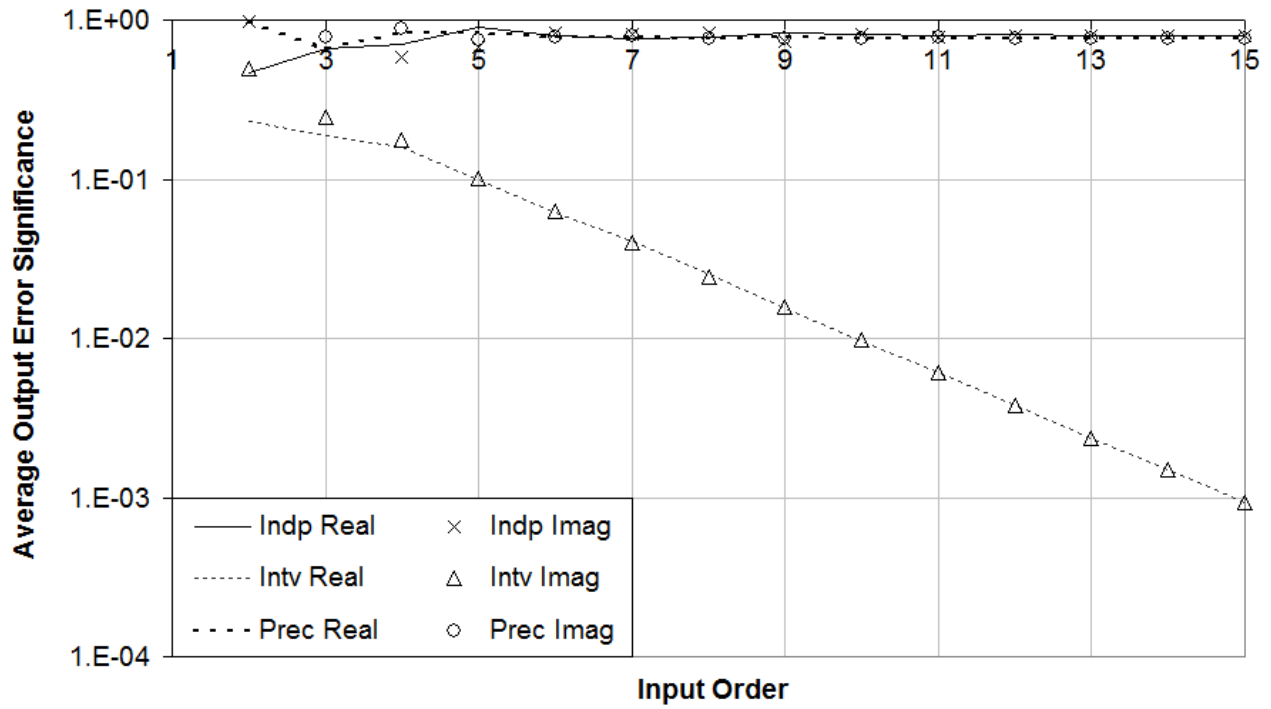


Figure 22: The output average error significands vs. FFT order of forward FFT for all three arithmetics when the input uncertainty deviation is 10^{-3} . In the legend, “Intv” means interval arithmetic, “Indp” means independence arithmetic, “Prec” means precision arithmetic, “Real” means real part, and “Imag” means imaginary part.

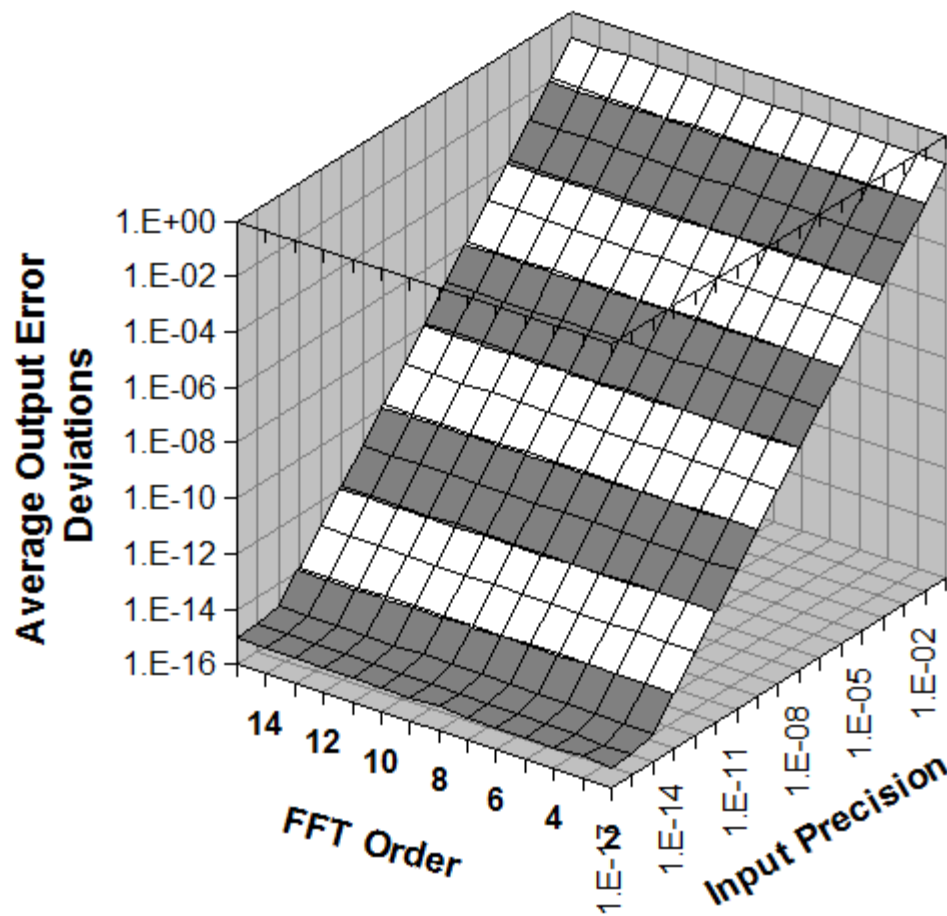


Figure 23: The average output deviations vs. FFT order and input deviations using precision arithmetic for round-trip FFT algorithm.

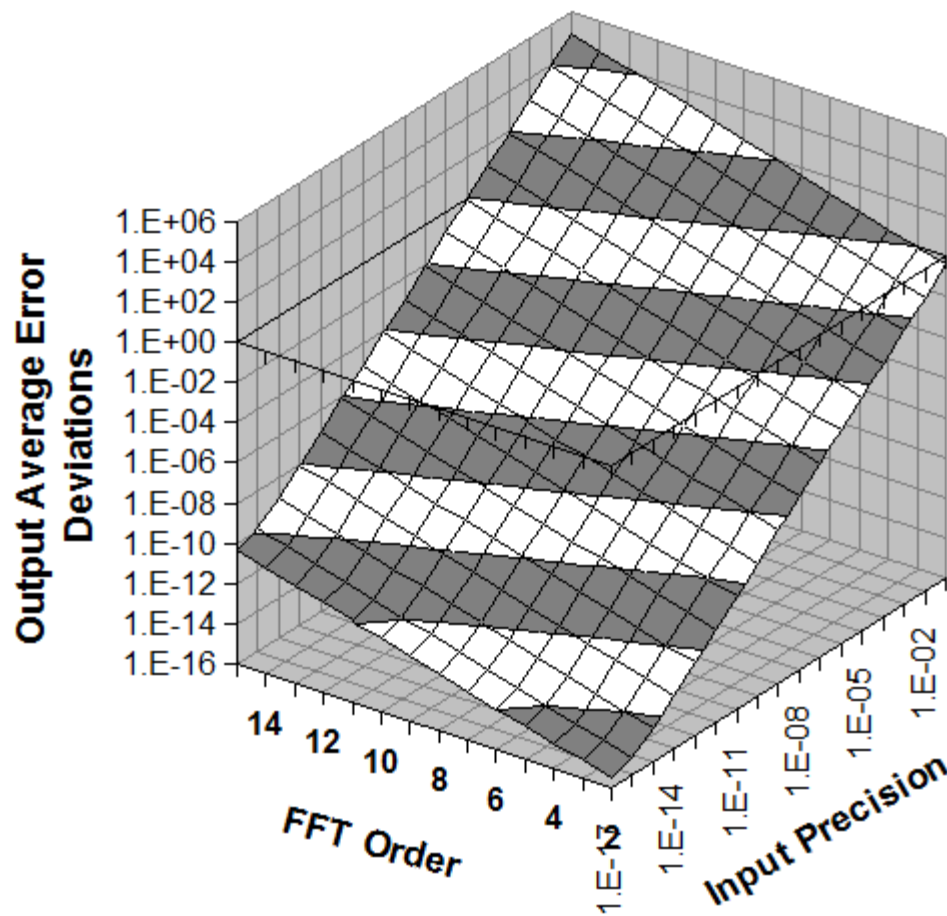


Figure 24: The average output deviations vs. FFT order and input deviations using interval arithmetic for round-trip FFT algorithm.

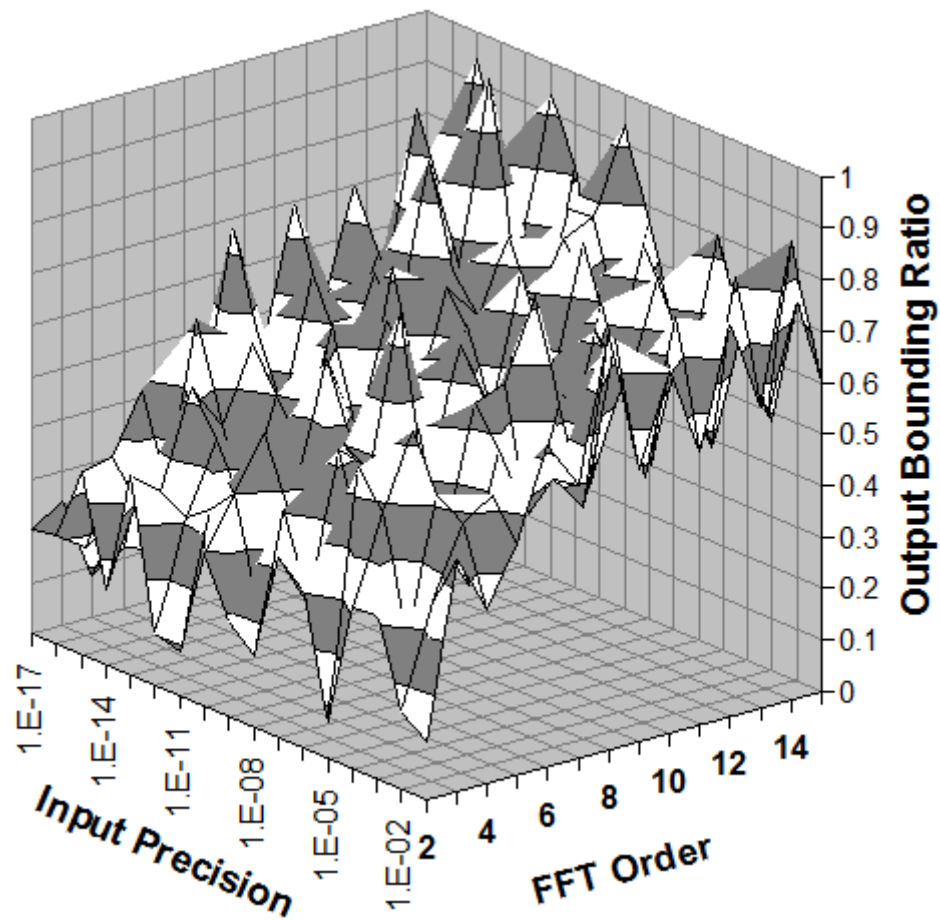


Figure 25: The maximal output bounding ratios using precision arithmetic increases slowly with FFT order.

Precision Arithmetic: A New Floating-Point Arithmetic

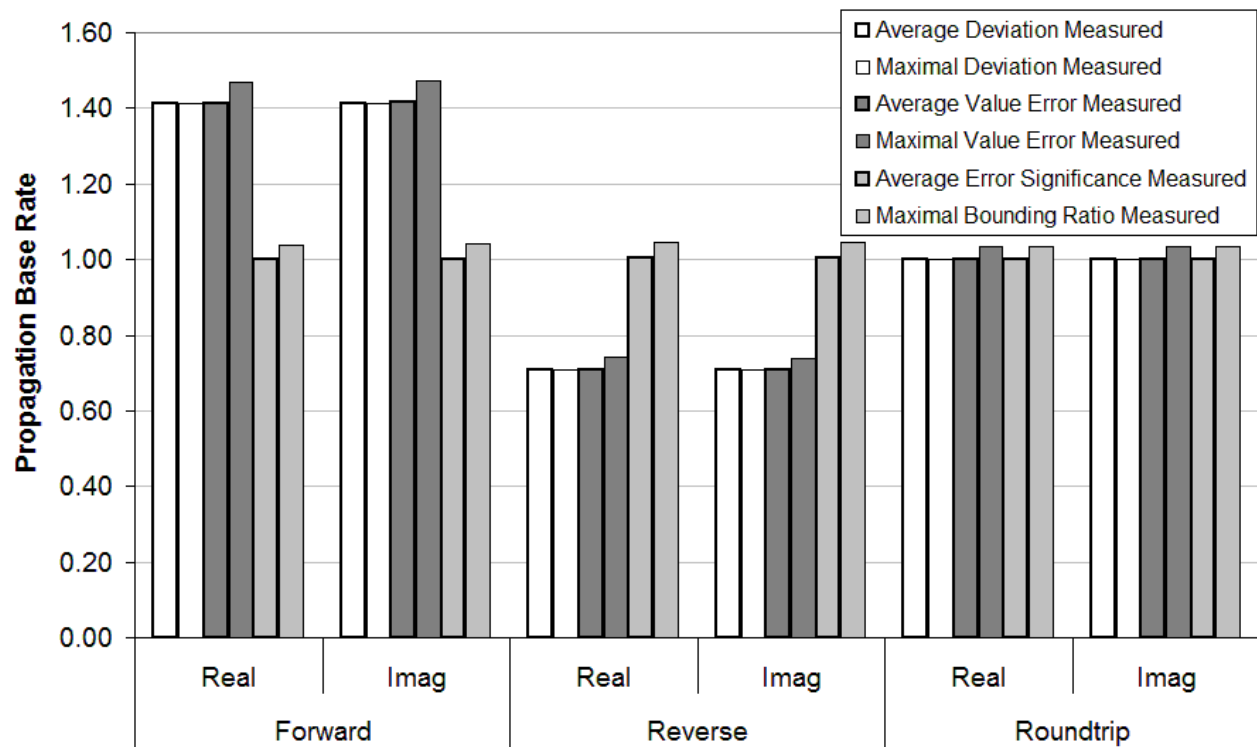


Figure 26: The characterization plot of independence arithmetic on noisy sine signal. In the legend, “Real” means real part, and “Imag” means imaginary part.

Precision Arithmetic: A New Floating-Point Arithmetic

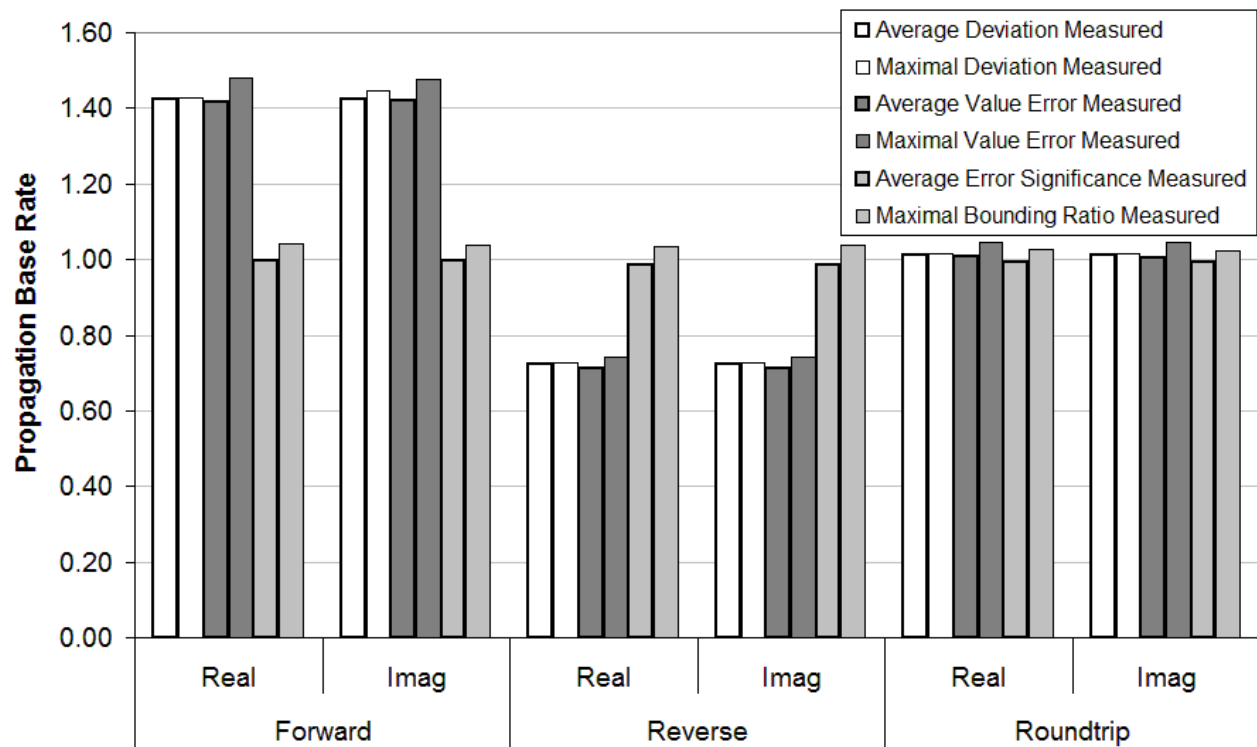


Figure 27: The characterization plot of precision arithmetic on noisy sine signal. In the legend, “Real” means real part, and “Imag” means imaginary part.

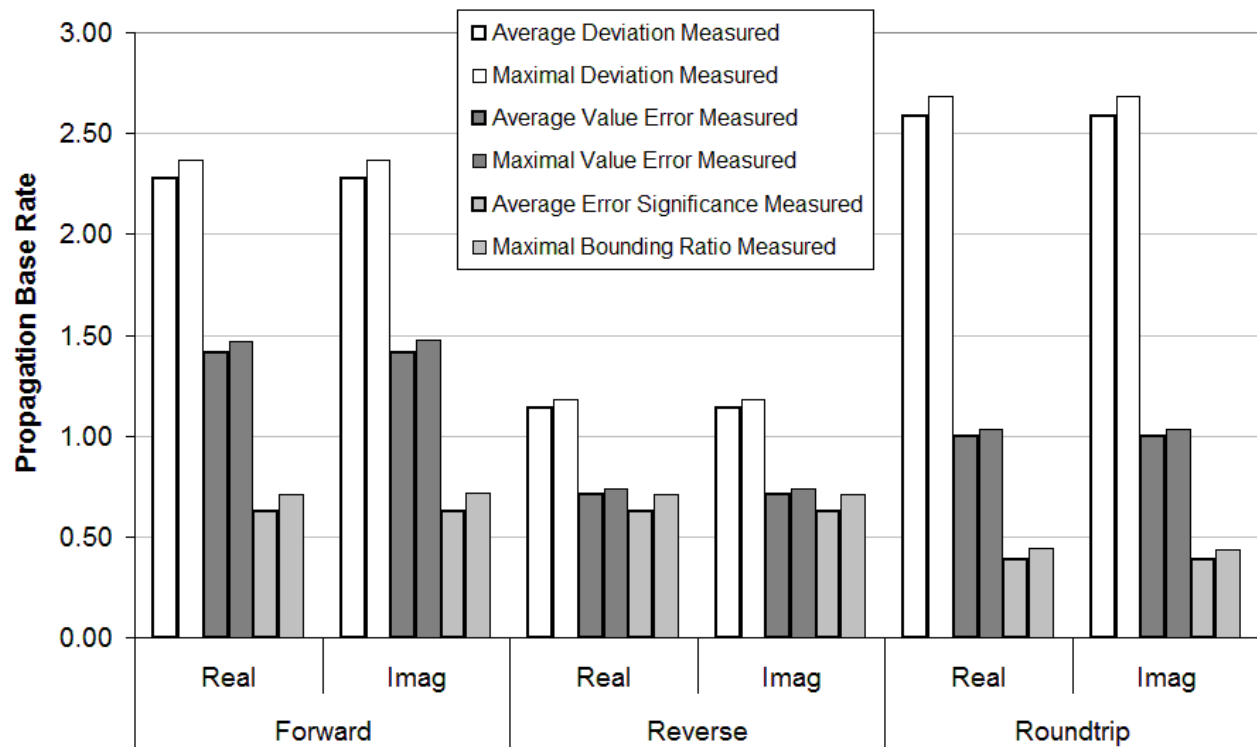


Figure 28: The characterization plot of interval arithmetic on noisy sine signal. In the legend, “Real” means real part, and “Imag” means imaginary part.

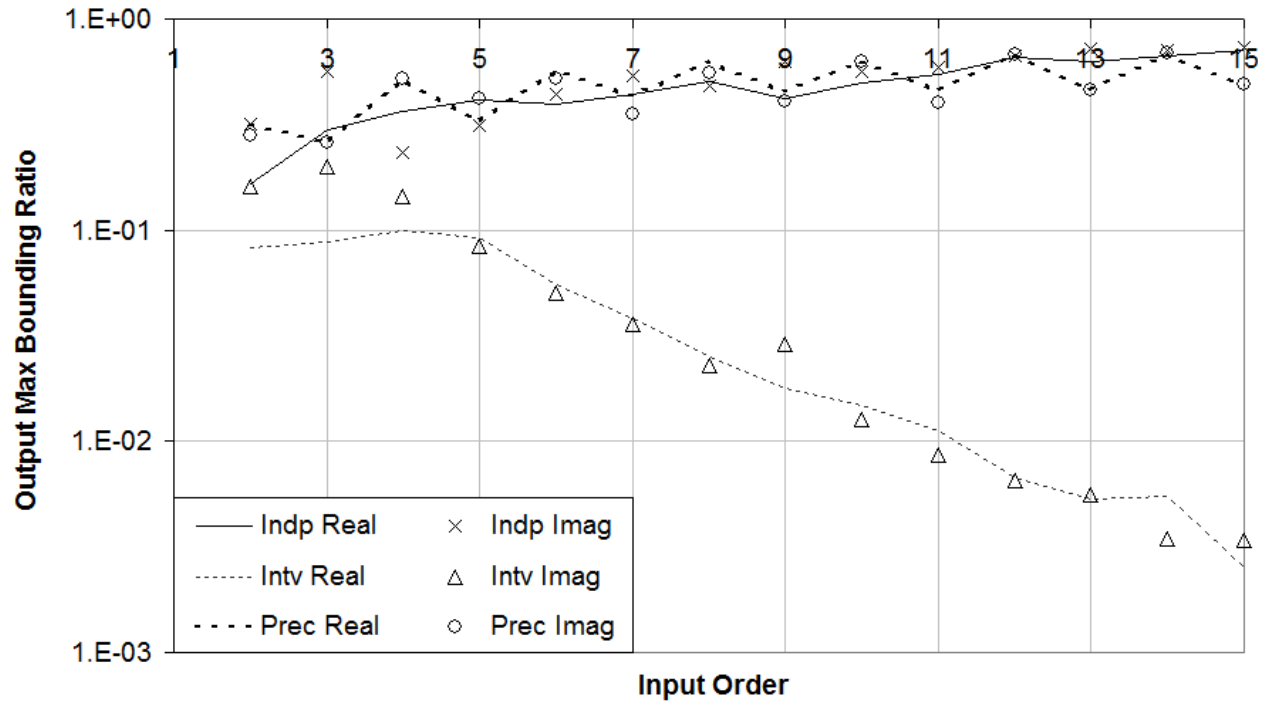


Figure 29: The output maximal bounding ratios vs. FFT order of forward FFT for all three arithmetics. In the legend, “Intv” means interval arithmetic, “Indp” means independence arithmetic, “Prec” means precision arithmetic, “Real” means real part, and “Imag” means imaginary part.

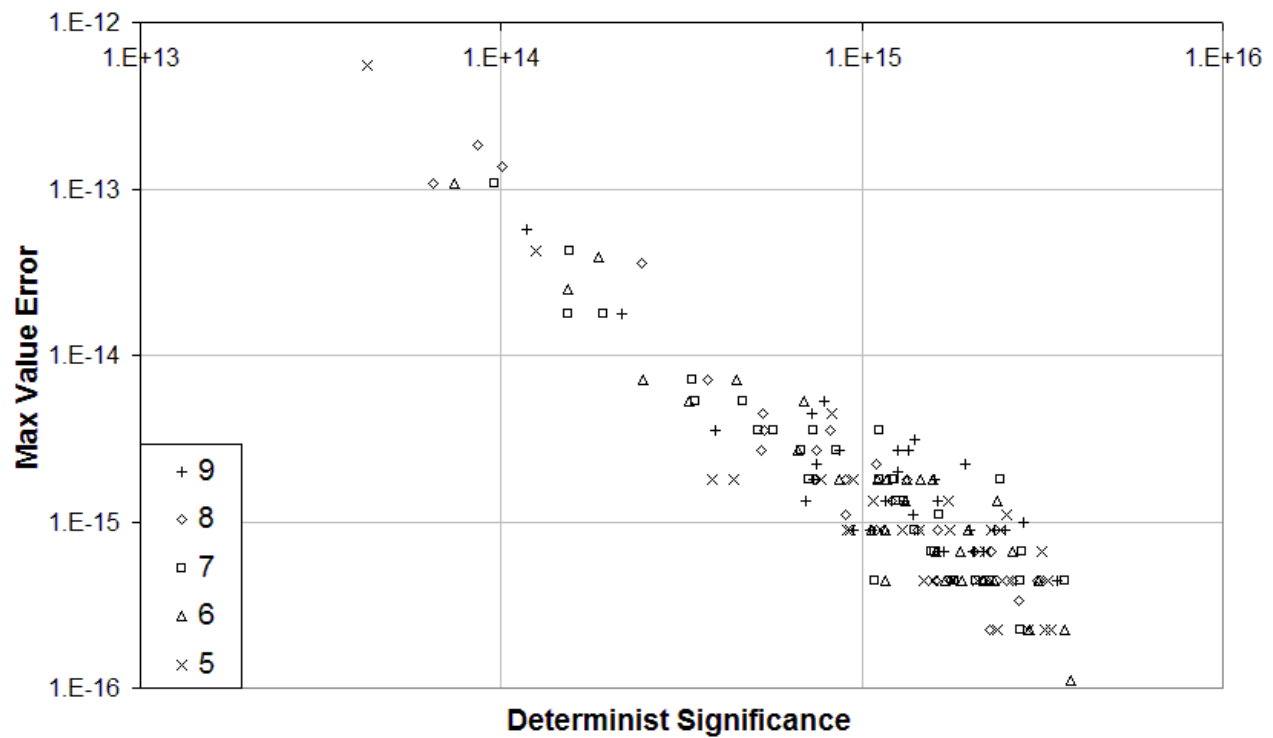


Figure 30: Maximal value errors of inverted matrix using conventional floating-point arithmetic vs. matrix determinant significance using precision arithmetic for clean matrixes of different size (as shown in legend).

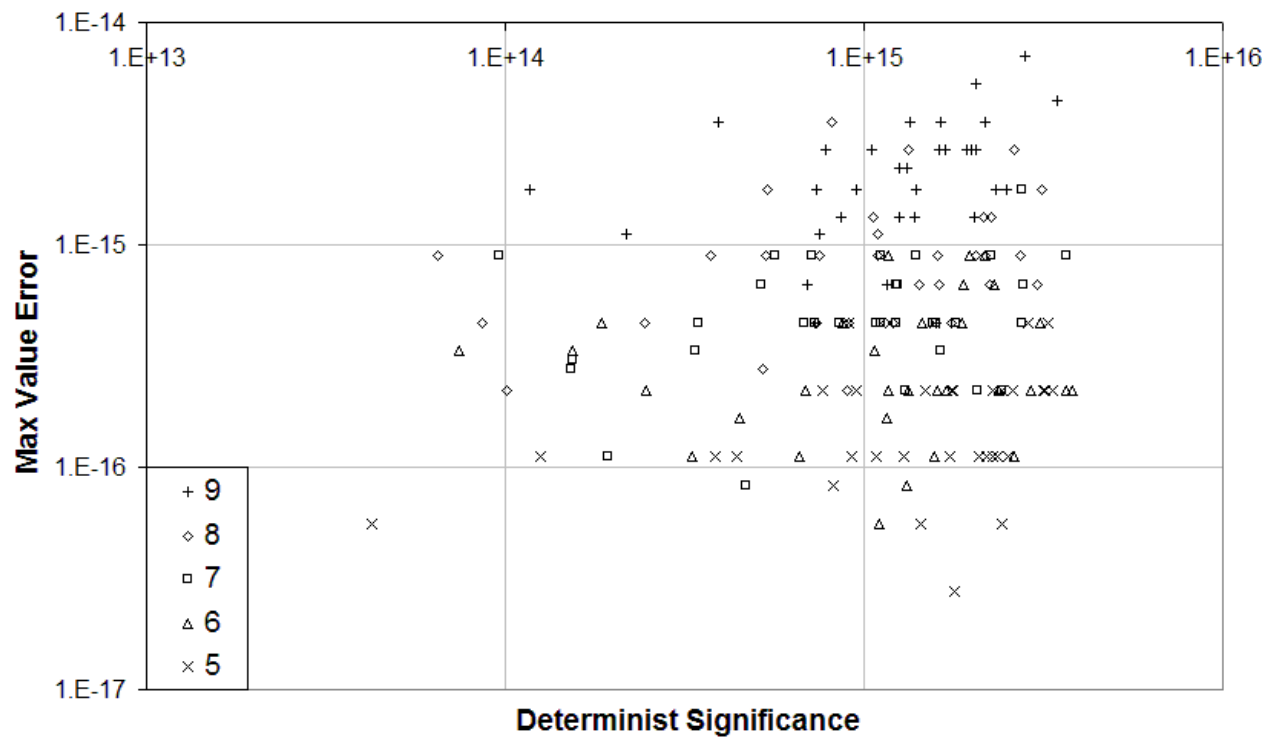


Figure 31: Maximal value errors of adjugate matrix using conventional floating-point arithmetic vs. matrix determinant significance using precision arithmetic for clean matrixes of different size (as shown in legend).

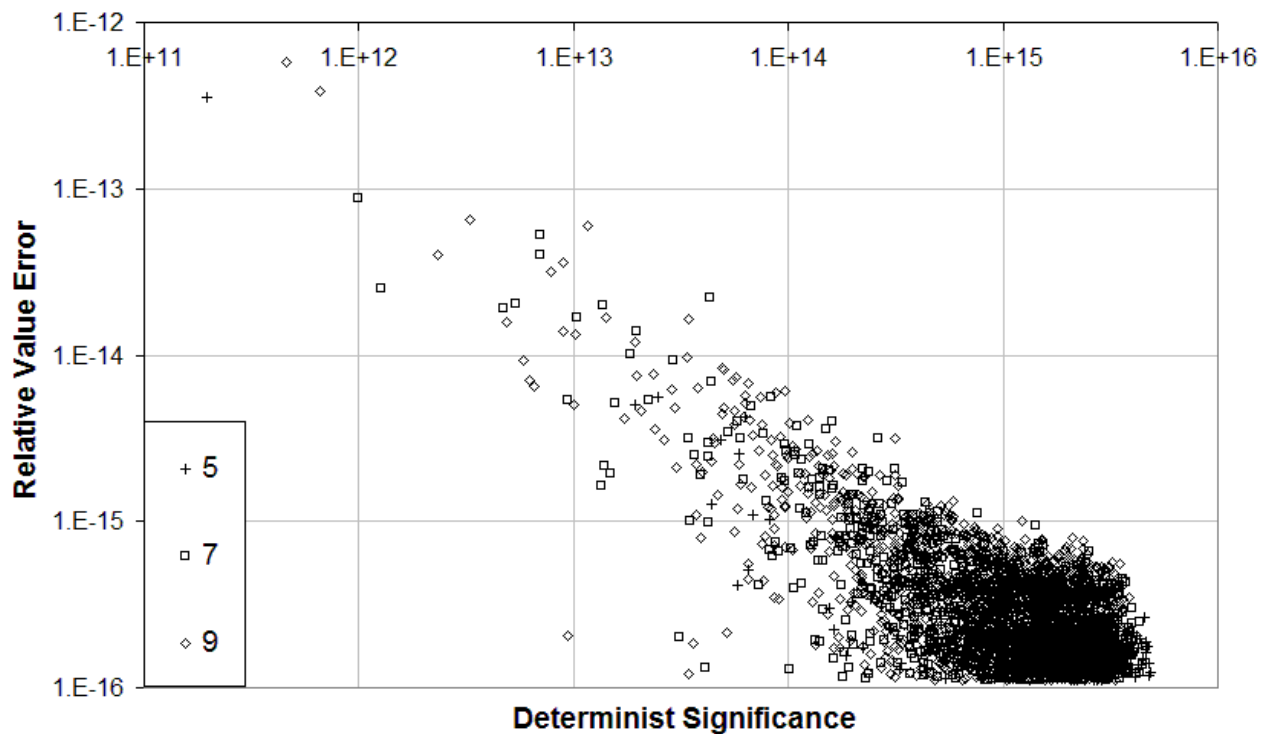


Figure 32: Relative value errors of adjugate matrix using conventional floating-point arithmetic vs. corresponding sub-matrix determinant significance using precision arithmetic for clean matrixes of different size (as shown in legend).

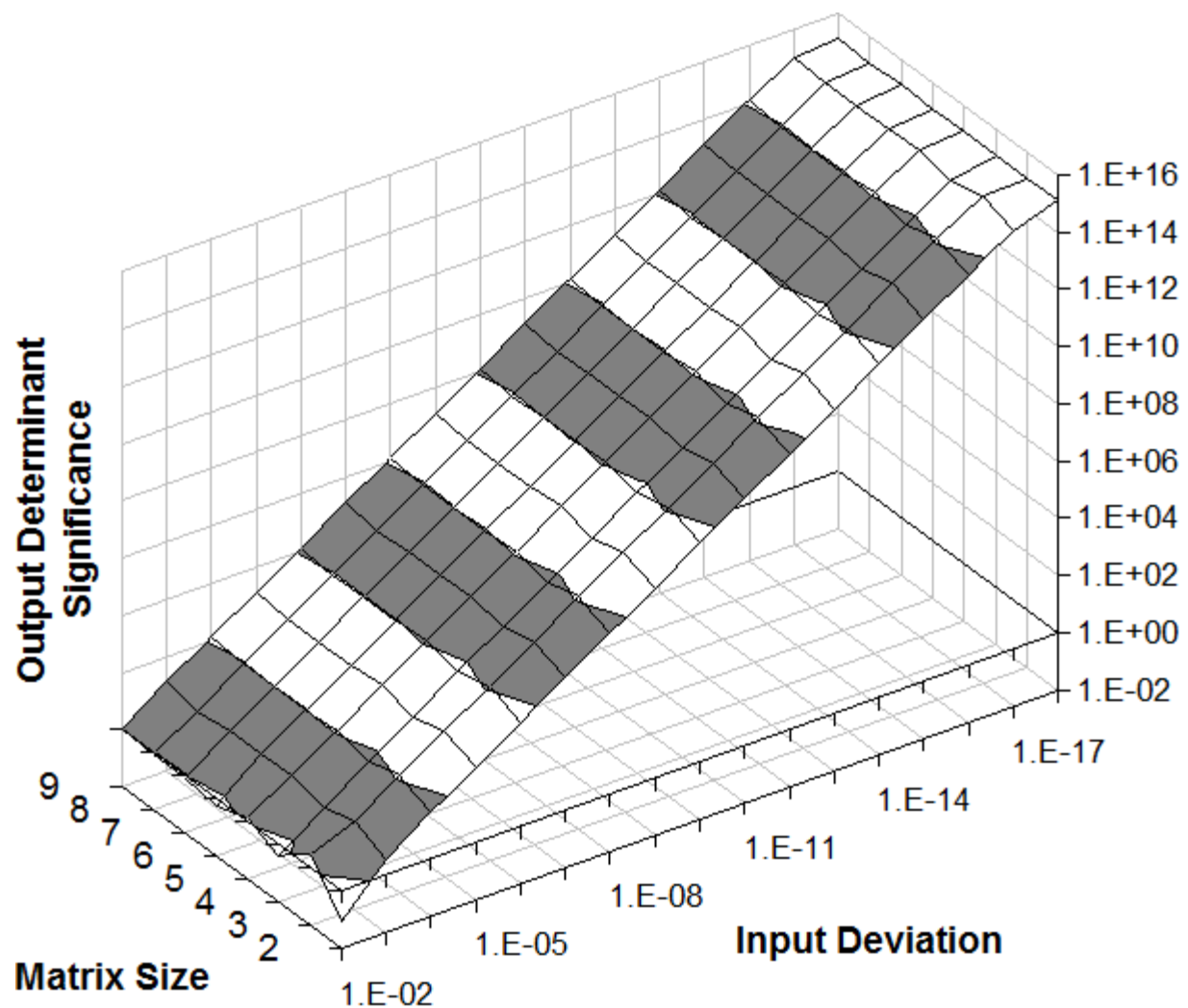


Figure 33: Using precision arithmetic, the significance of matrix determinant vs. input uncertainty and matrix size. Clean testing matrixes of different sizes but same significances are selected, and Gaussian noises of different deviations are added to construct noisy matrixes of different input uncertainties.

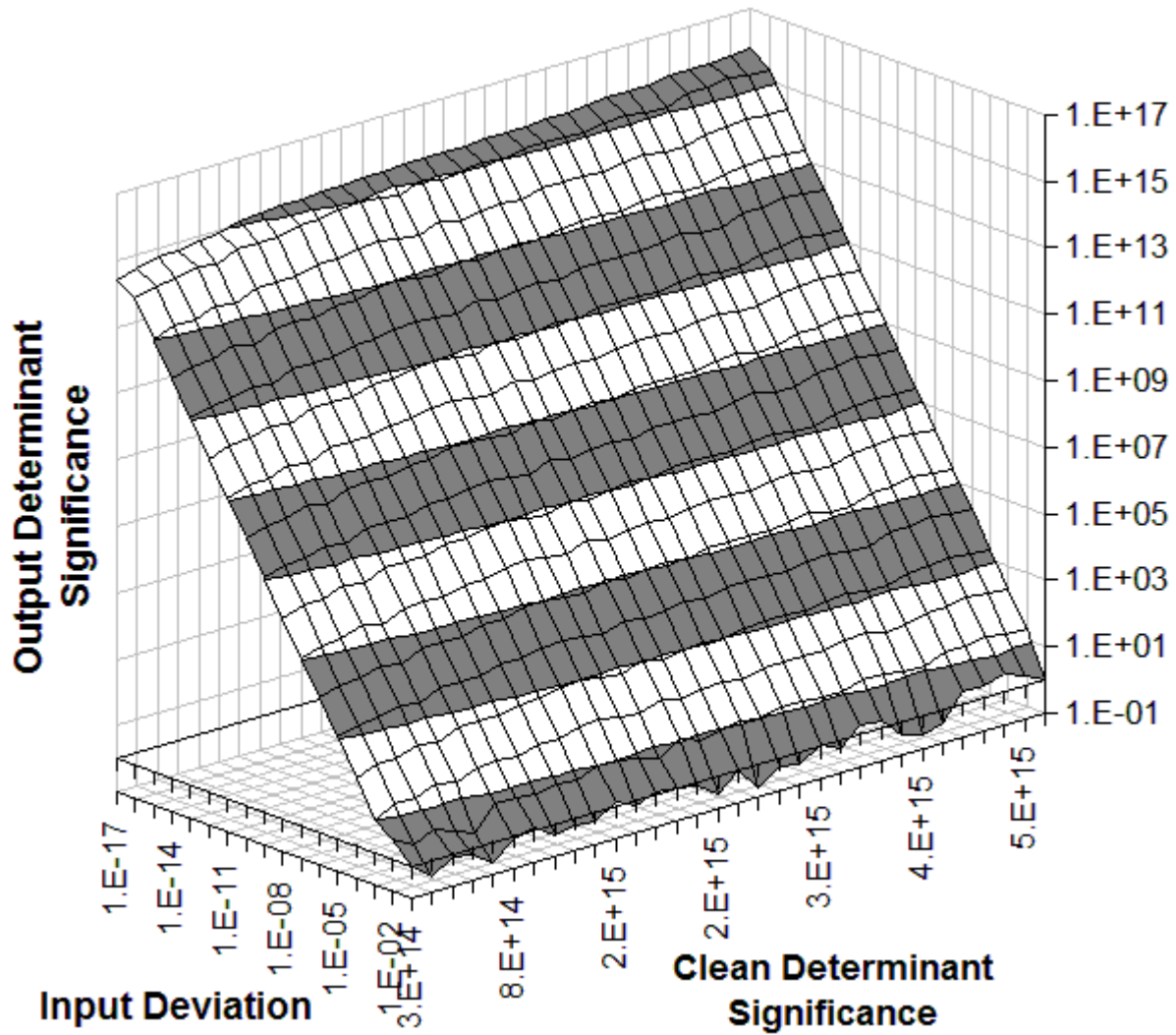


Figure 34: Using precision arithmetic, the significance of matrix determinant vs. input precision and the determinant significance of clean testing matrixes. Clean testing matrixes of same size are sorted according to their determinant significances, and Gaussian noises of different deviations are added to construct noisy matrixes of different input uncertainties.

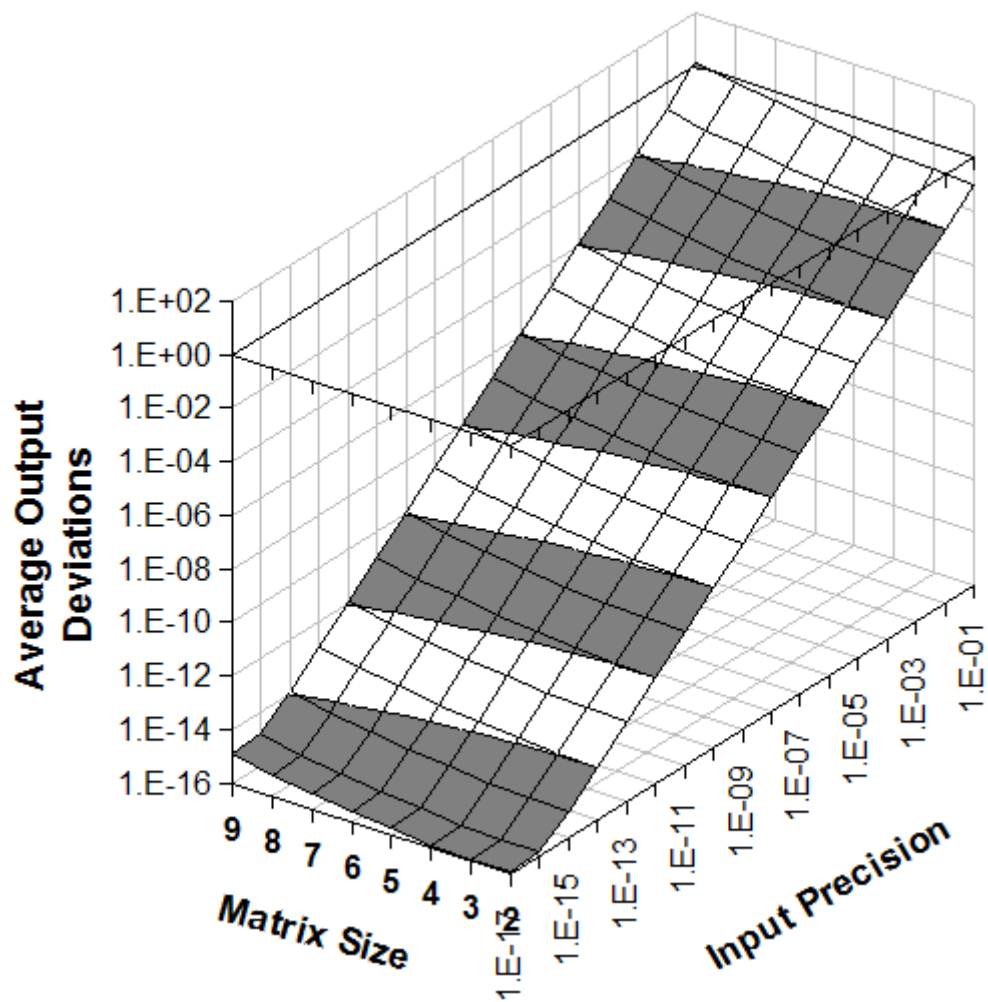


Figure 35: Using precision arithmetic, the average output deviations of adjugate matrix vs. input precision and matrix size.

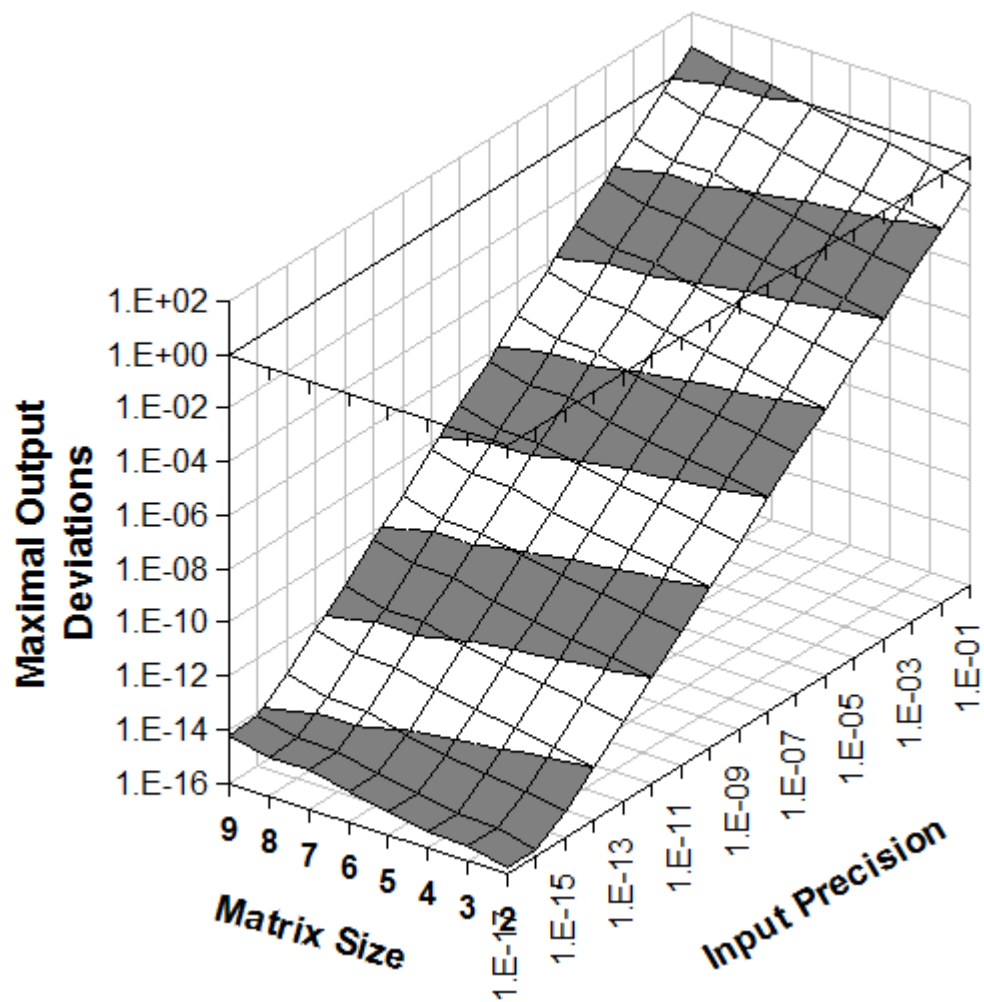


Figure 36: Using precision arithmetic, the maximal output deviations of adjugate matrix vs. input precision and matrix size.

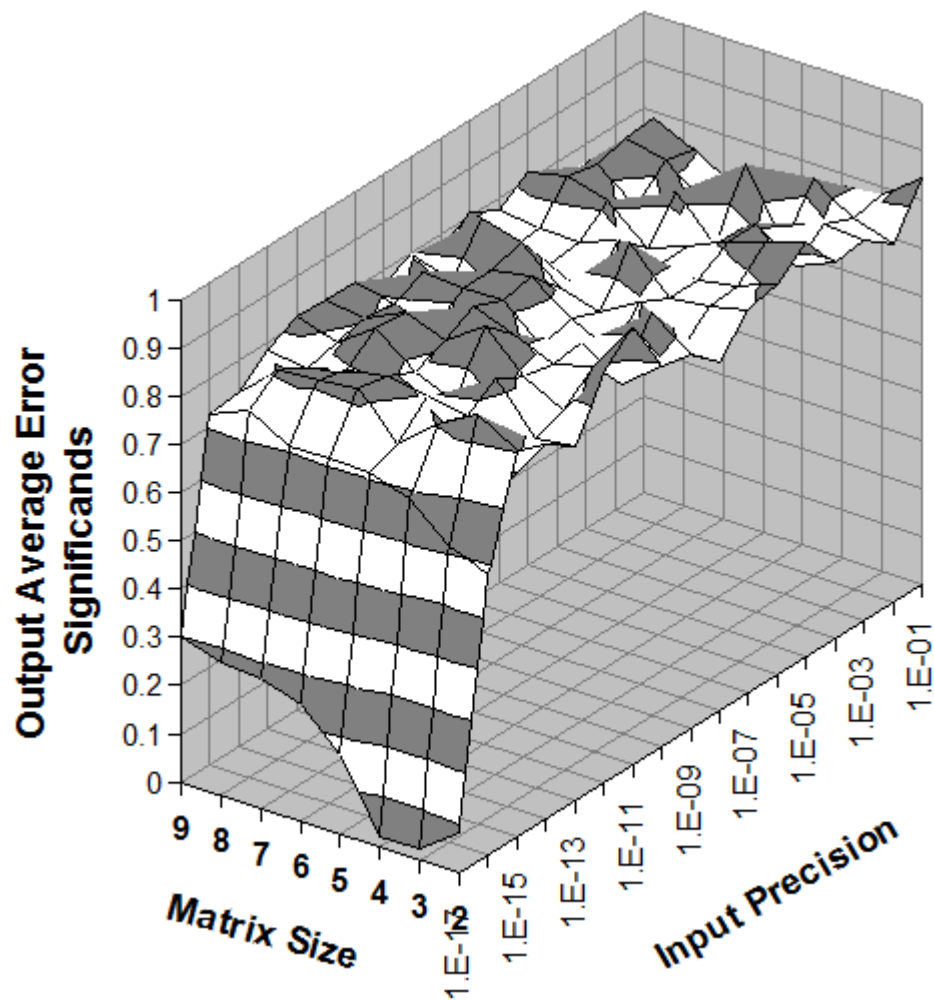


Figure 37: Using precision arithmetic, the average output error significances of adjugate matrix vs. input precision and matrix size.

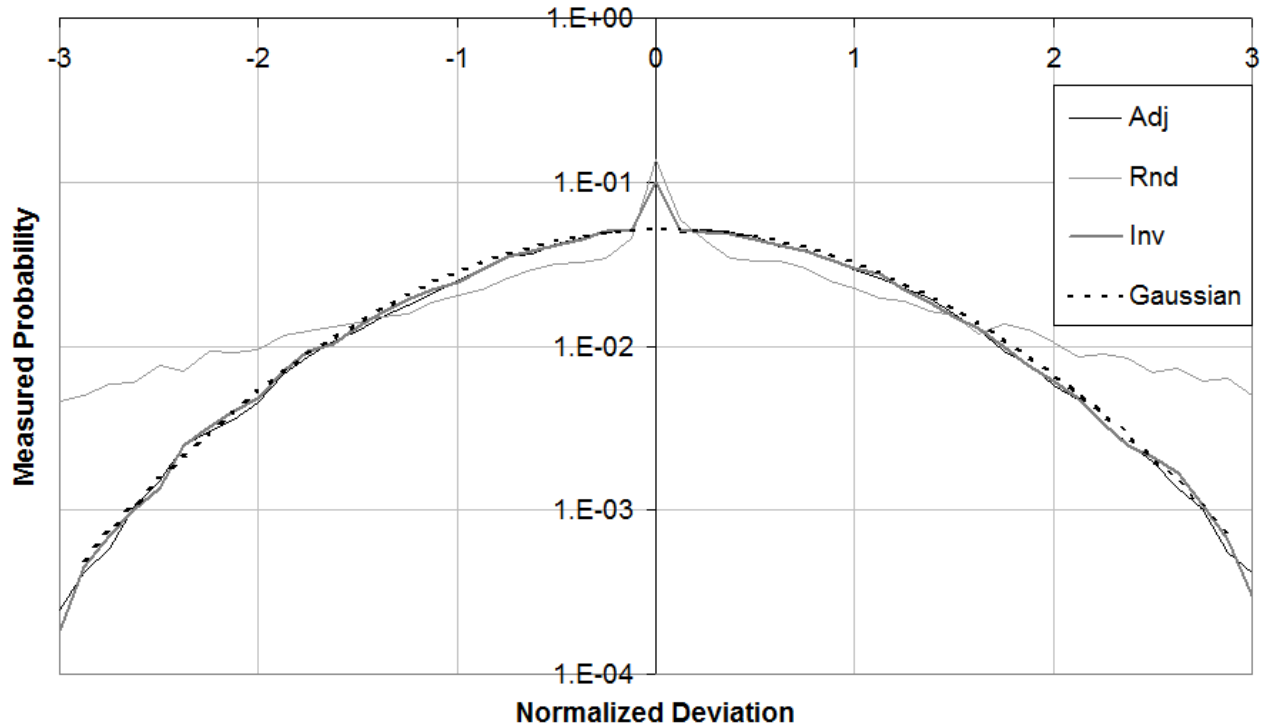


Figure 38: The measured normalized value error distributions using precision arithmetic with 4-bit calculated inside uncertainty for matrix calculations of matrix size 9. They are best fitted by a Gaussian distribution with mean of 0.06 and deviation of 0.96. In the legend, “Adj” for calculating adjugate M^A , “Inv” for calculating inverted M^{-1} , and “Rnd” for calculating double inverted $(M^{-1})^{-1}$. Please also see Footnote 4 for additional explanations.

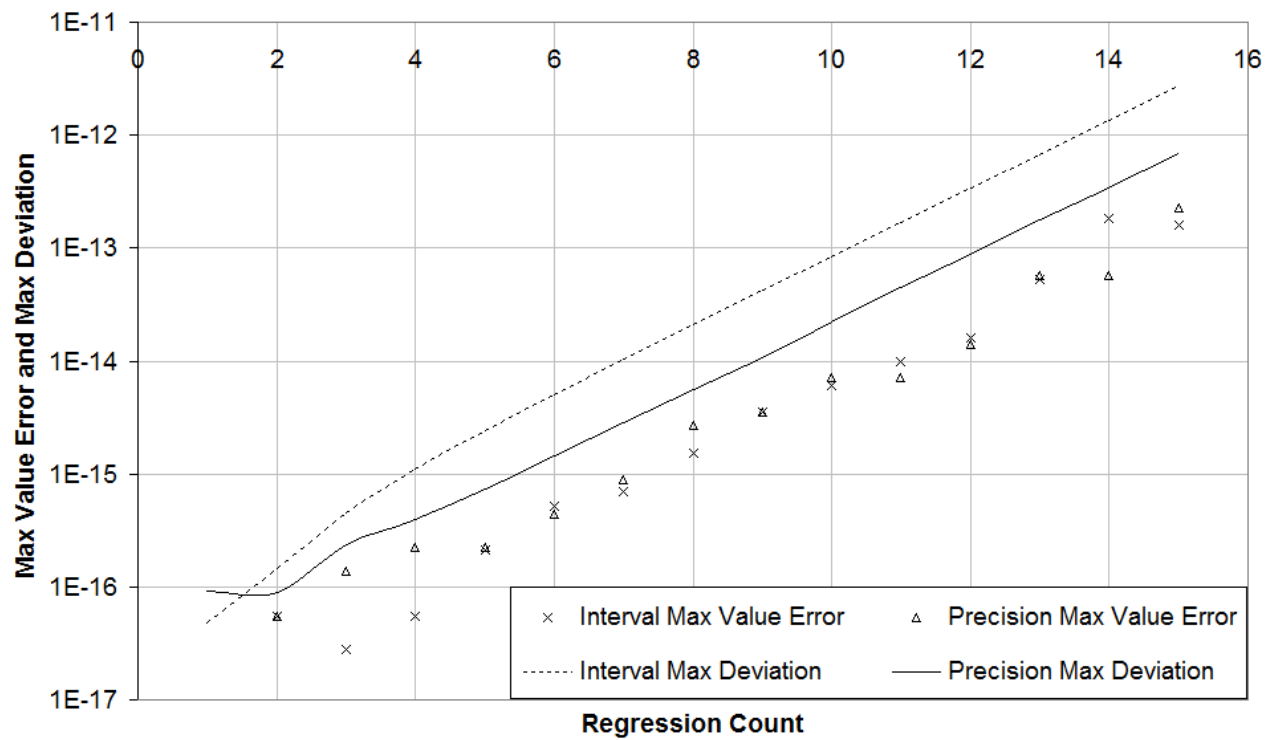


Figure 39: The output maximal value errors and corresponding output deviations vs. regression count of regressive calculation of sine values using interval arithmetic and precision arithmetic.

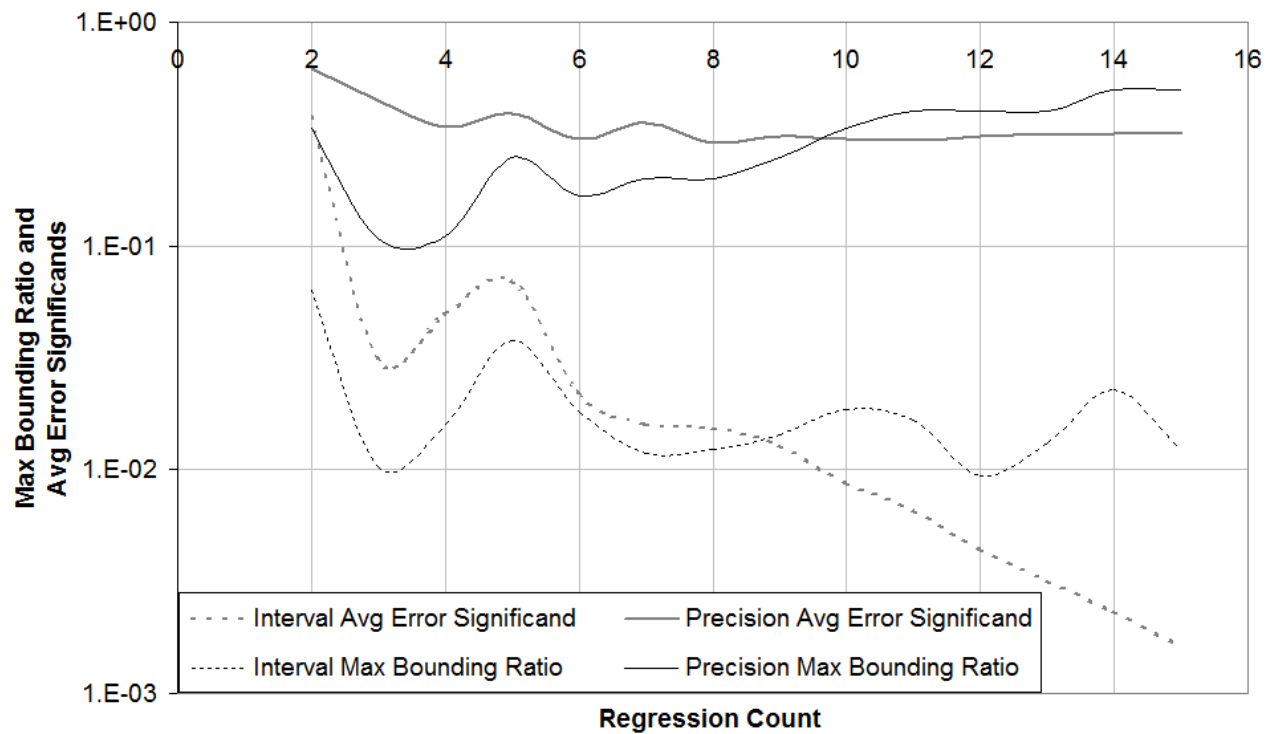


Figure 40: The output maximal bounding ratios and average error significands vs. regression count of regressive calculation of sine values using interval and precision arithmetics.

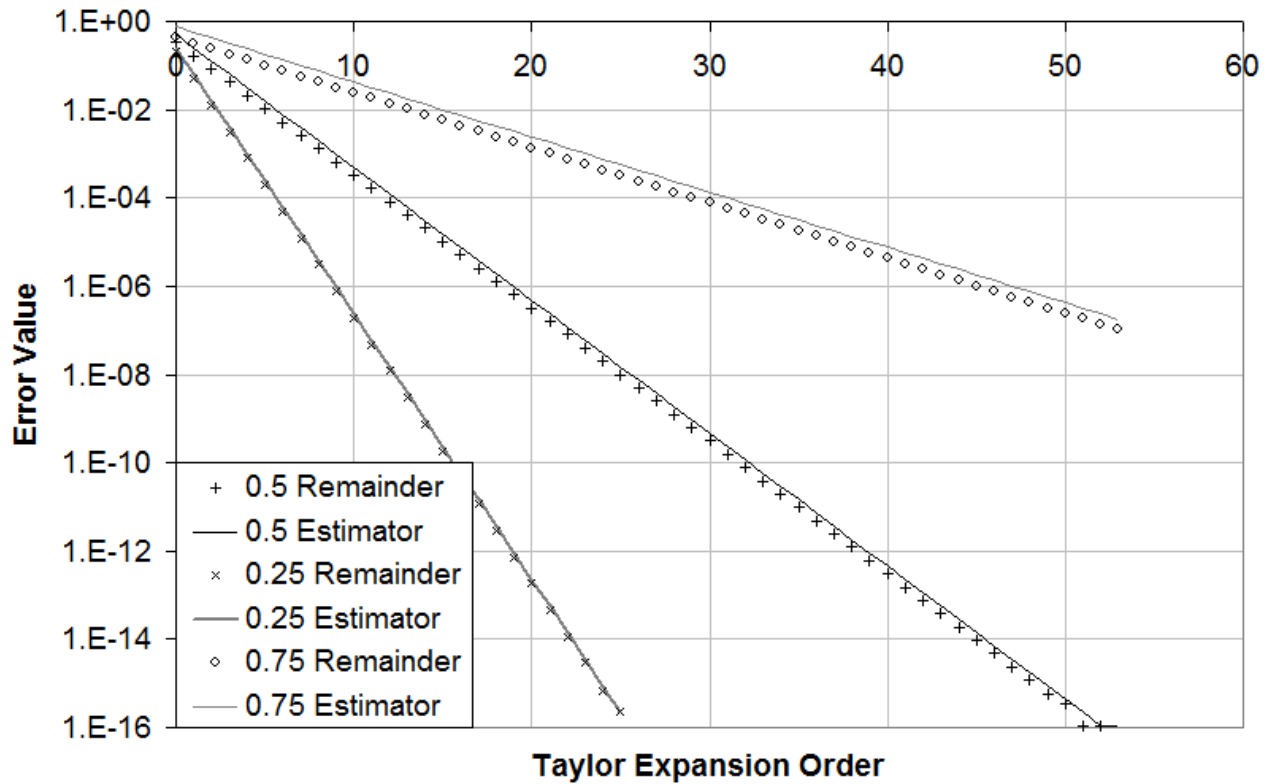


Figure 41: The Cauchy remainder estimator and remainder of a Taylor expansion vs. expansion order for different input value without input uncertainty. Both Cauchy remainder estimator and remainder of a Taylor expansion are calculated using conventional floating-point arithmetic. The rounding errors are estimated as 5.55×10^{-17} , 1.11×10^{-16} and 2.22×10^{-16} respectively for input of 1/4, 1/2 and 3/4 using precision arithmetic.

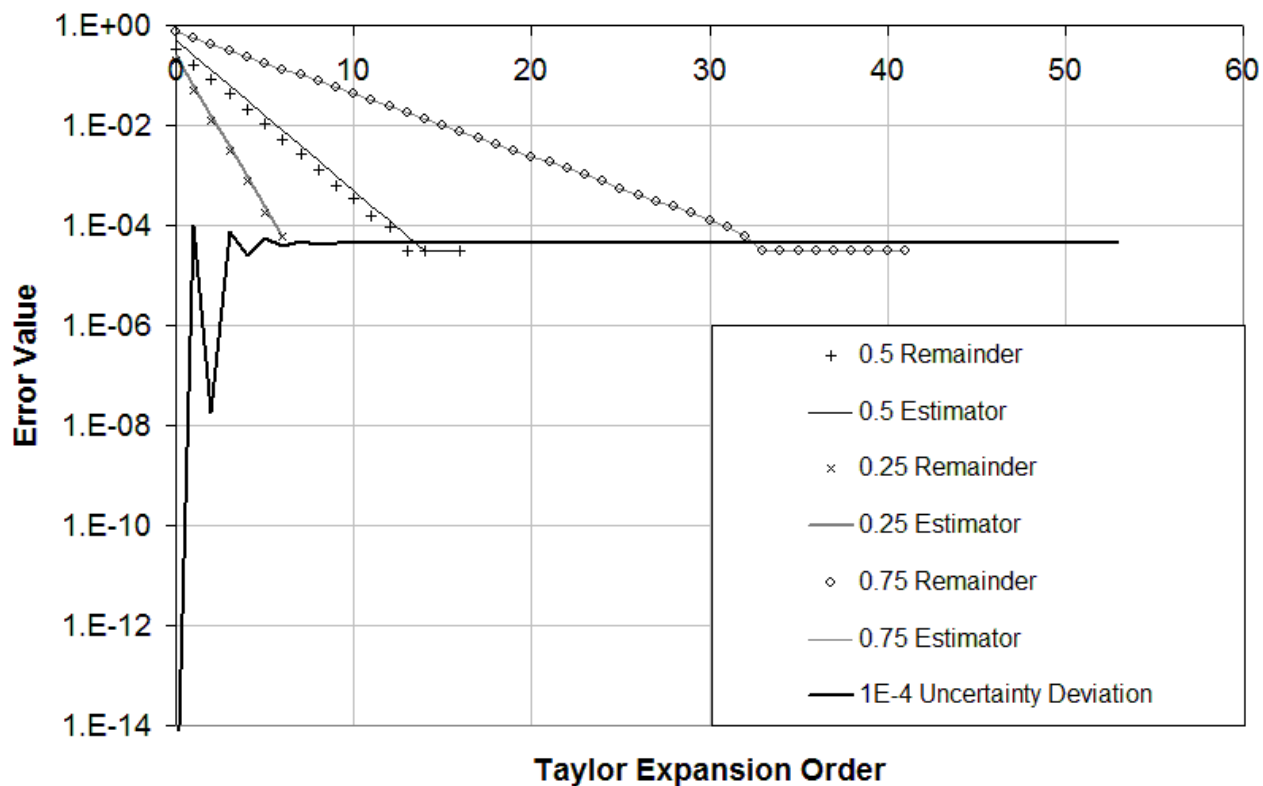


Figure 42: The Cauchy remainder estimator and remainder of a Taylor expansion vs. expansion order for different input value with input uncertainty deviation of 10^{-4} using precision arithmetic with no bit calculated inside uncertainty.

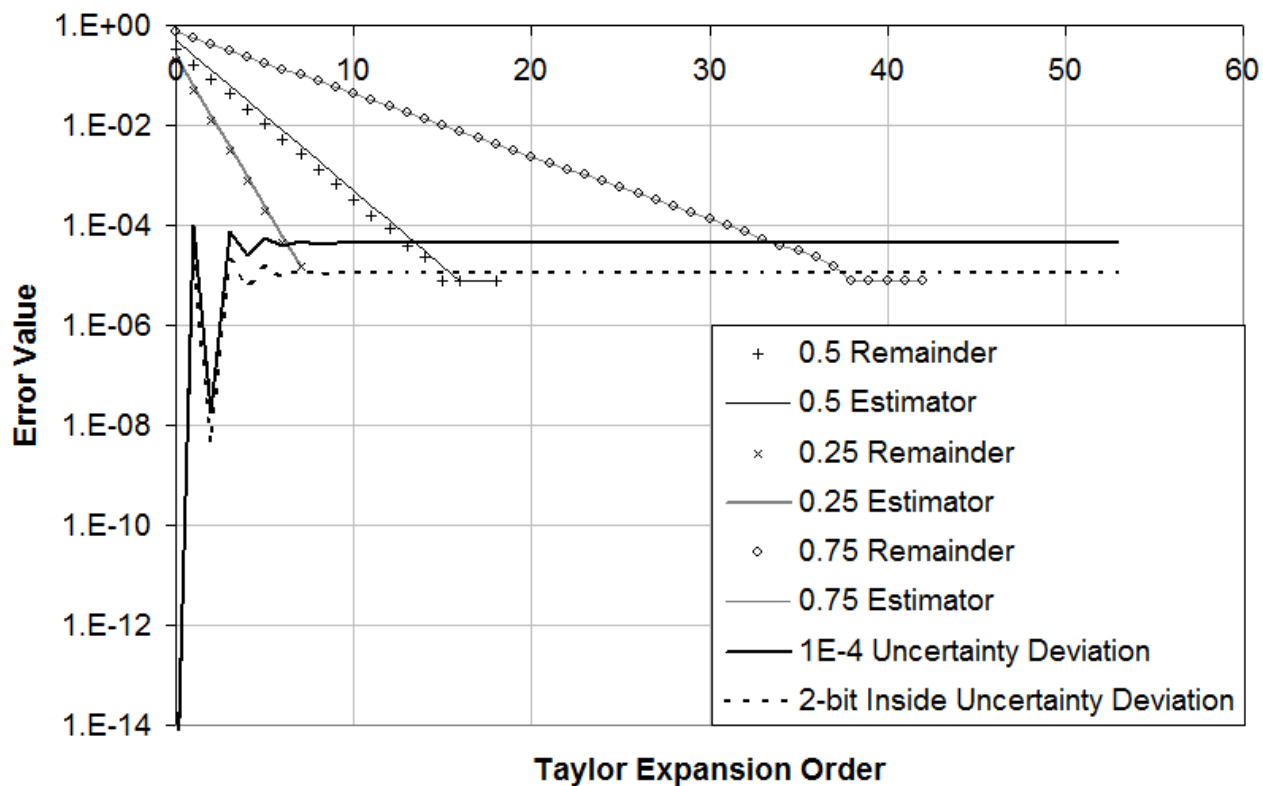


Figure 43: The Cauchy remainder estimator and remainder of a Taylor expansion vs. expansion order for different input value with input uncertainty deviation of 10^{-4} using precision arithmetic with 2-bit bit calculated inside uncertainty.

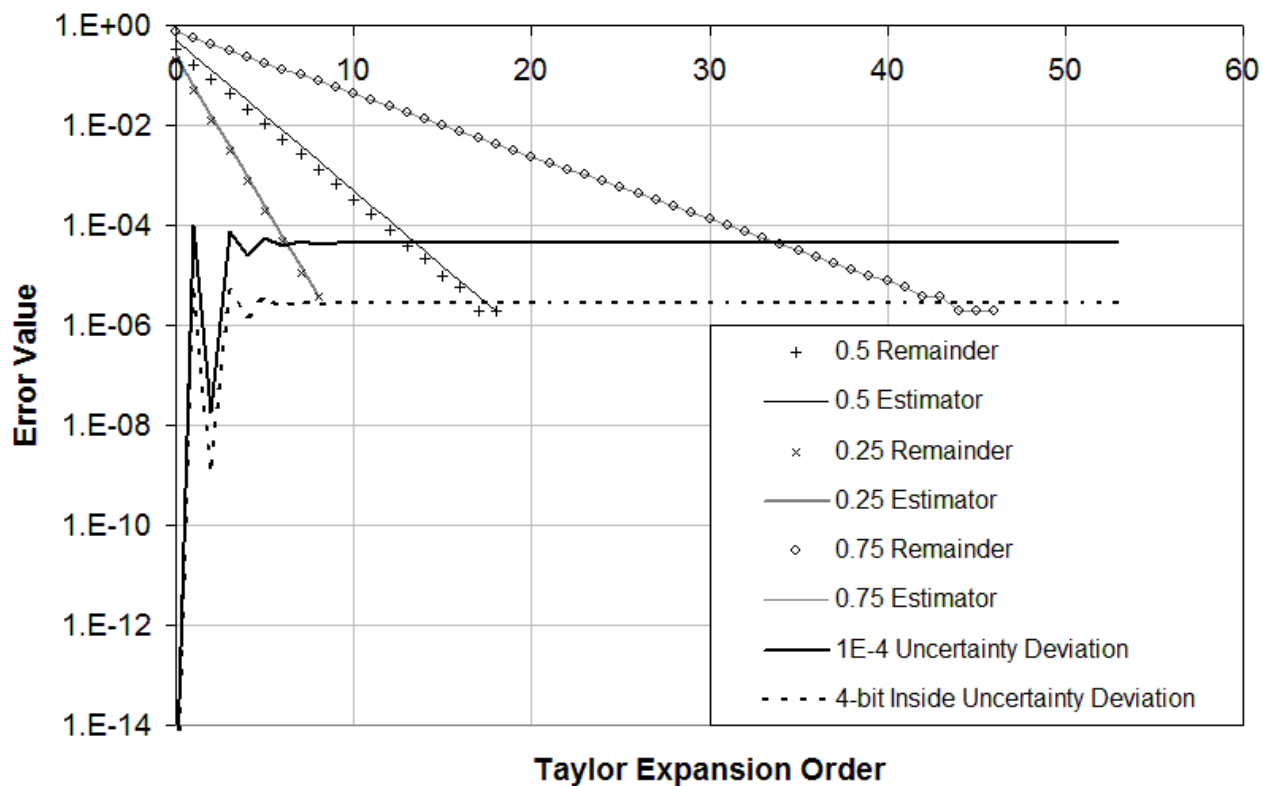


Figure 44: The Cauchy remainder estimator and remainder of a Taylor expansion vs. expansion order for different input value with input uncertainty deviation of 10^{-4} using precision arithmetic with 4-bit bit calculated inside uncertainty.

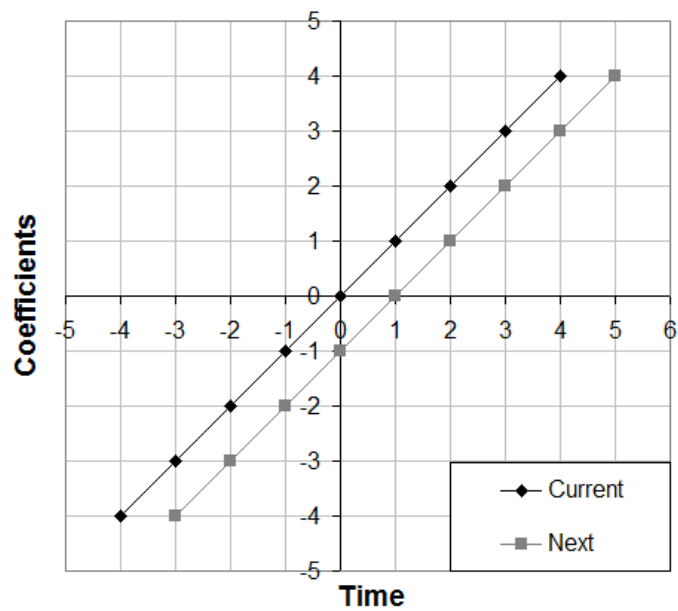


Figure 45: Coefficients of X in $\hat{\beta} = \sum_{X=-H}^{+H} XY_j$ at current and next position in a time series of least square linear regression.

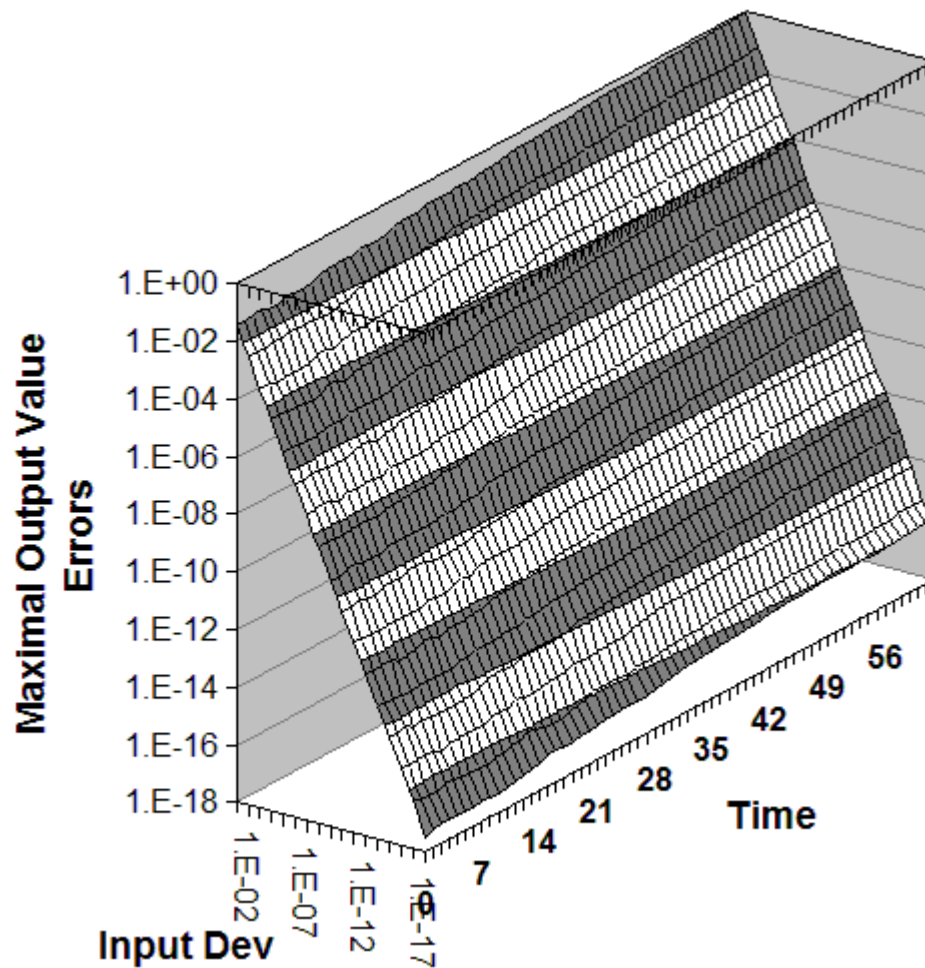


Figure 46: Maximal output value errors vs. time and input deviations for progressive moving-window linear regression of a straight line using precision arithmetic.

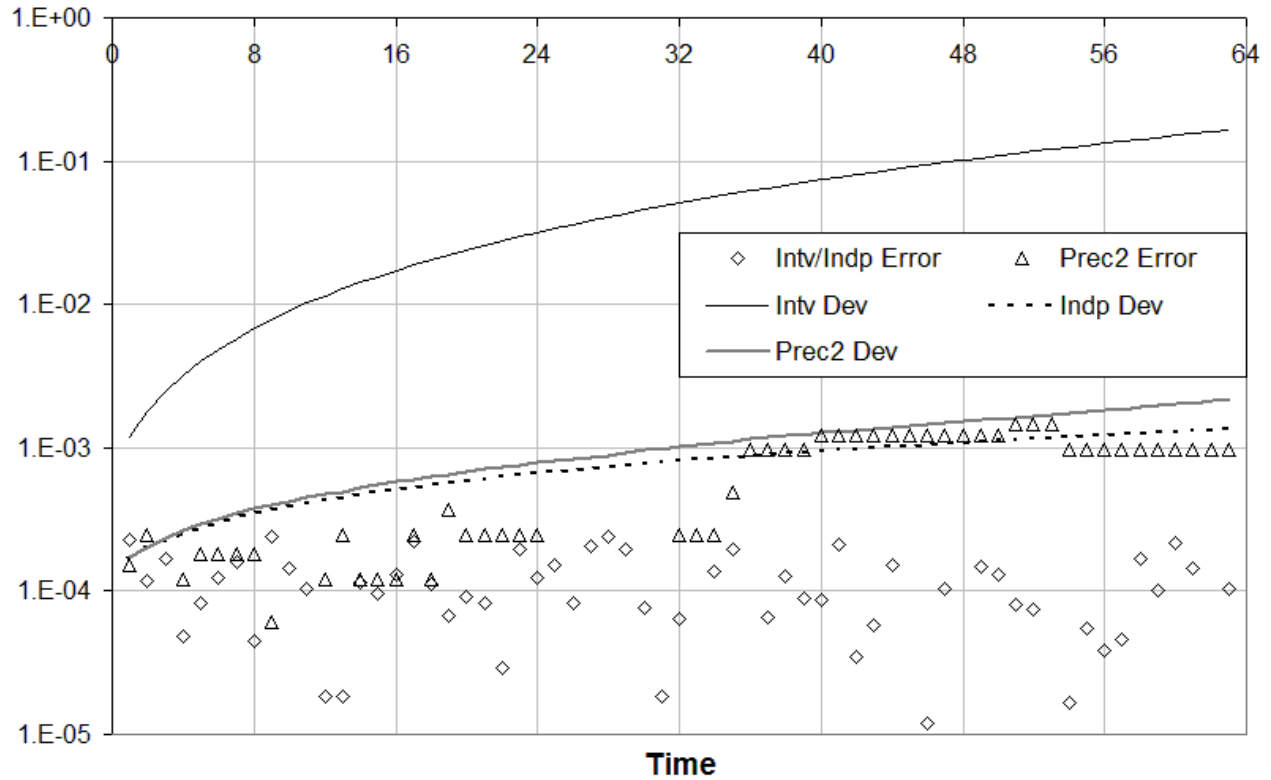


Figure 47: Output uncertainty deviations and value errors vs. time for progressive moving-window linear regression of a straight line. In the legend, “Indp” means independence arithmetic, “Intv” means interval arithmetic, and “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty.

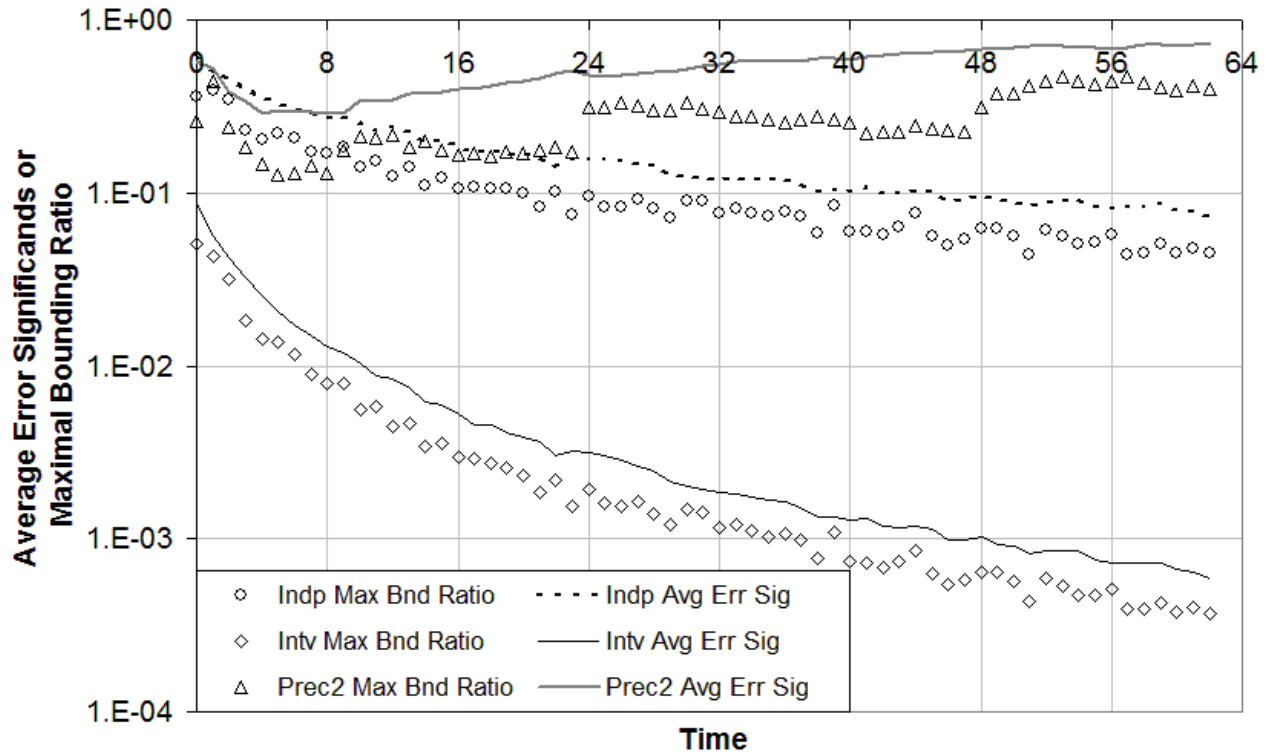


Figure 48: Average error significands and max bounding ratio vs. time for progressive moving-window linear regression of a straight line. In the legend, “Indp” means independence arithmetic, “Intv” means interval arithmetic, and “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty. “Max Bnd Ratio” is the abbreviation for maximal bounding ratio, and “Avg Err Sig” is the abbreviation for average error significance.

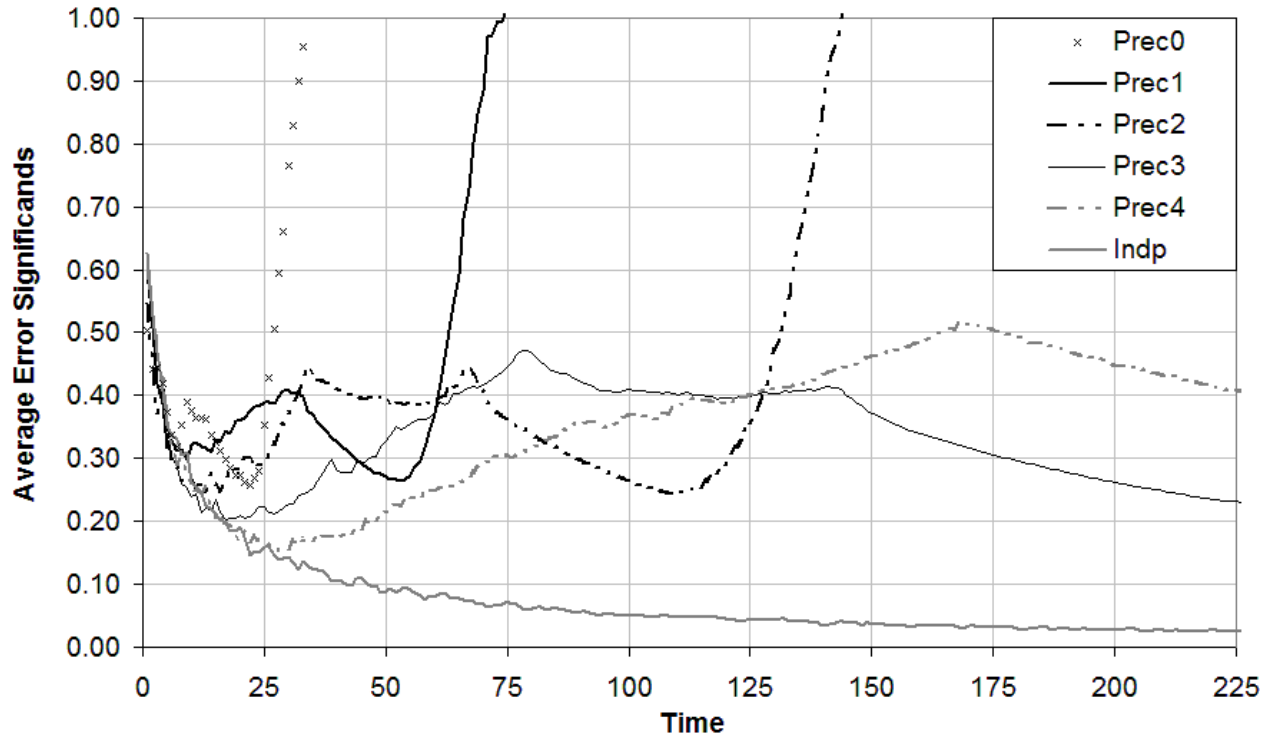


Figure 49: Average error significands vs. time and bits calculated inside uncertainty using precision arithmetic for progressive moving-window linear regression of a straight line. In the legend, “Indp” means independence arithmetic, “Prec0” means precision arithmetic with no calculated inside uncertainty, “Prec1” means precision arithmetic with 1-bit calculated inside uncertainty, “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty, “Prec3” means precision arithmetic with 3-bit calculated inside uncertainty, and “Prec4” means precision arithmetic with 4-bit calculated inside uncertainty.

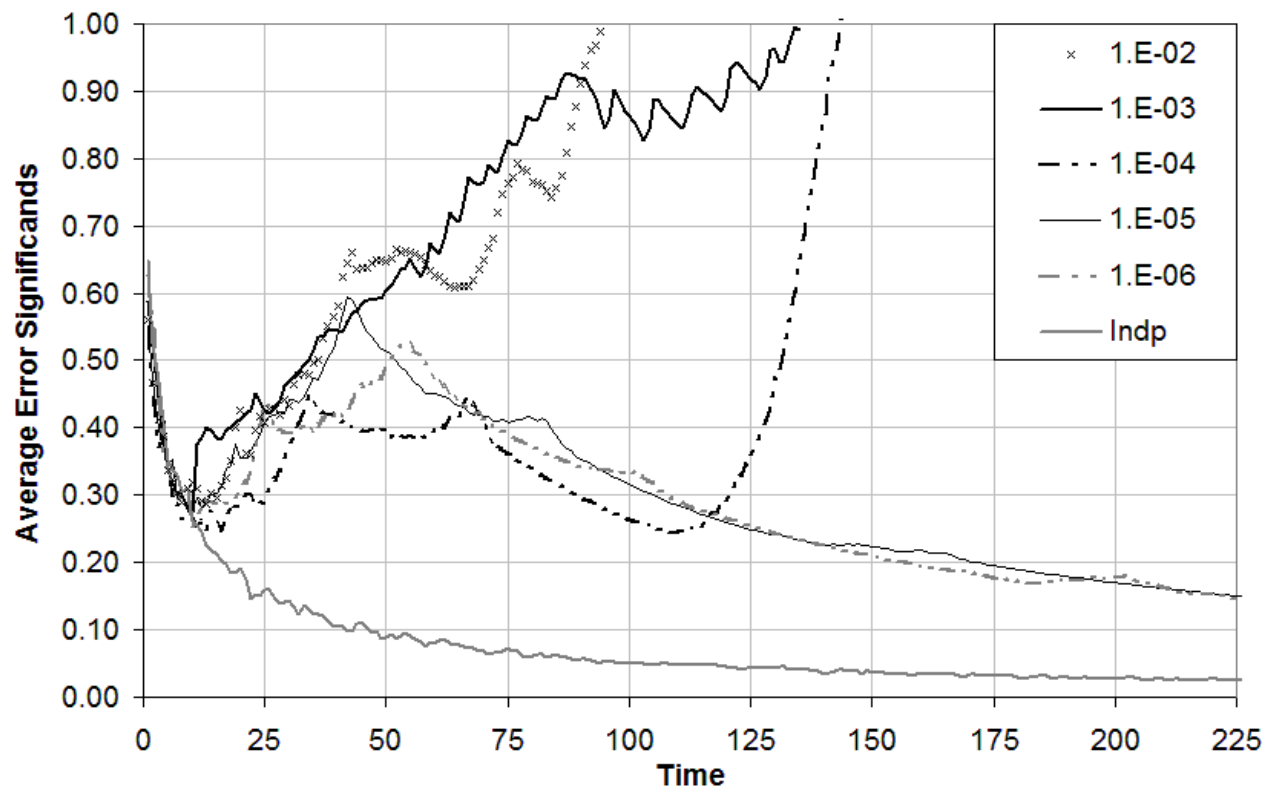


Figure 50: Average error significands vs. time and input precision using precision arithmetic for progressive moving-window linear regression of a straight line. In the legend, “Indp” means calculation without input precision using independence arithmetic, and other values means input precision for calculation using precision arithmetic with 2-bit calculated inside uncertainty.

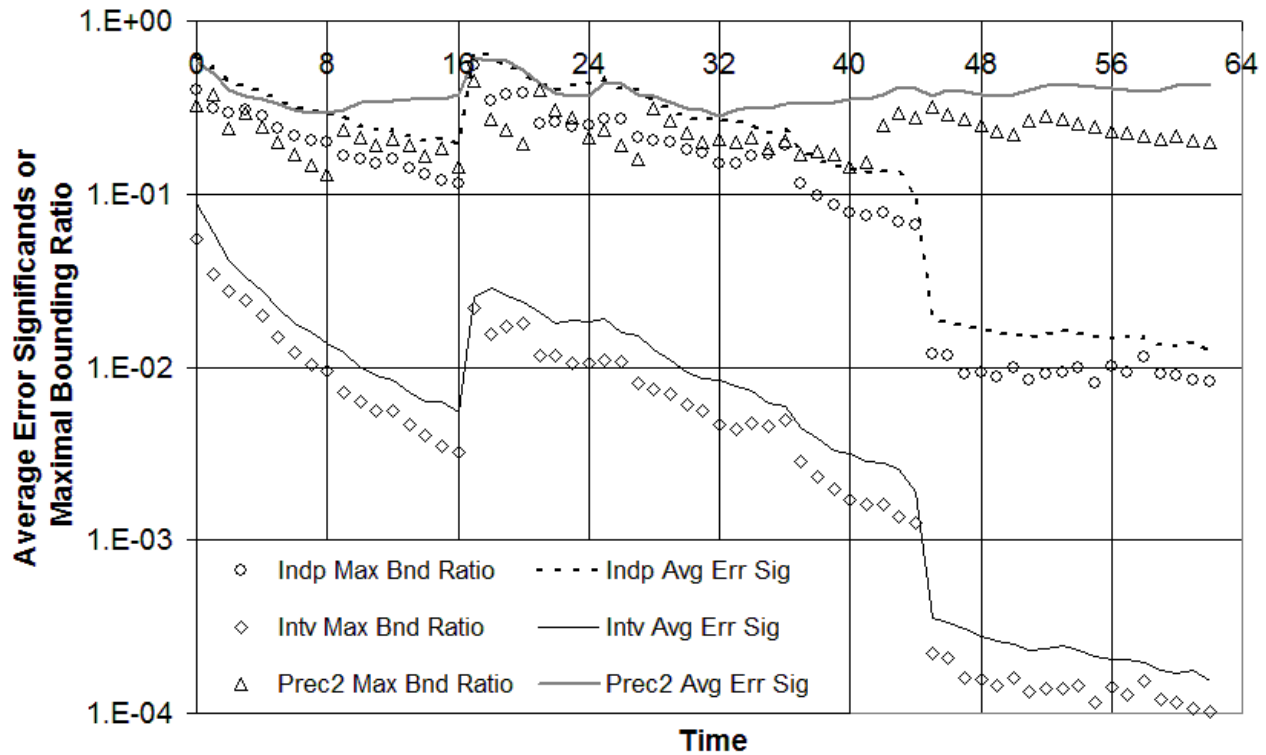


Figure 51: Average error significands and max bounding ratio vs. time for progressive moving-window linear regression of a straight line with larger input deviation in middle. In the legend, “Indp” means independence arithmetic, “Intv” means interval arithmetic, and “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty. “Max Bnd Ratio” is the abbreviation for maximal bounding ratio, and “Avg Err Sig” is the abbreviation for average error significance.

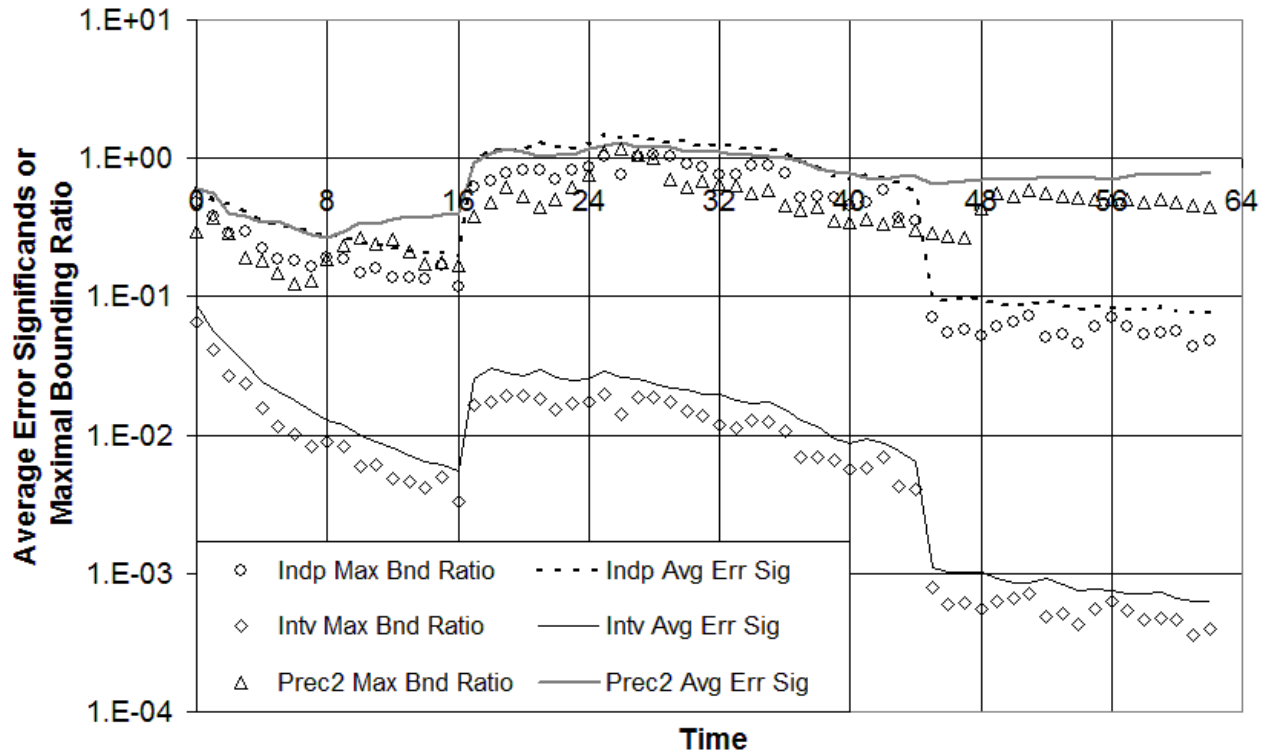


Figure 52: Average error significands and max bounding ratio vs. time for progressive moving-window linear regression of a straight line with larger input noise in middle. In the legend, “Indp” means independence arithmetic, “Intv” means interval arithmetic, and “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty. “Max Bnd Ratio” is the abbreviation for maximal bounding ratio, and “Avg Err Sig” is the abbreviation for average error significance.

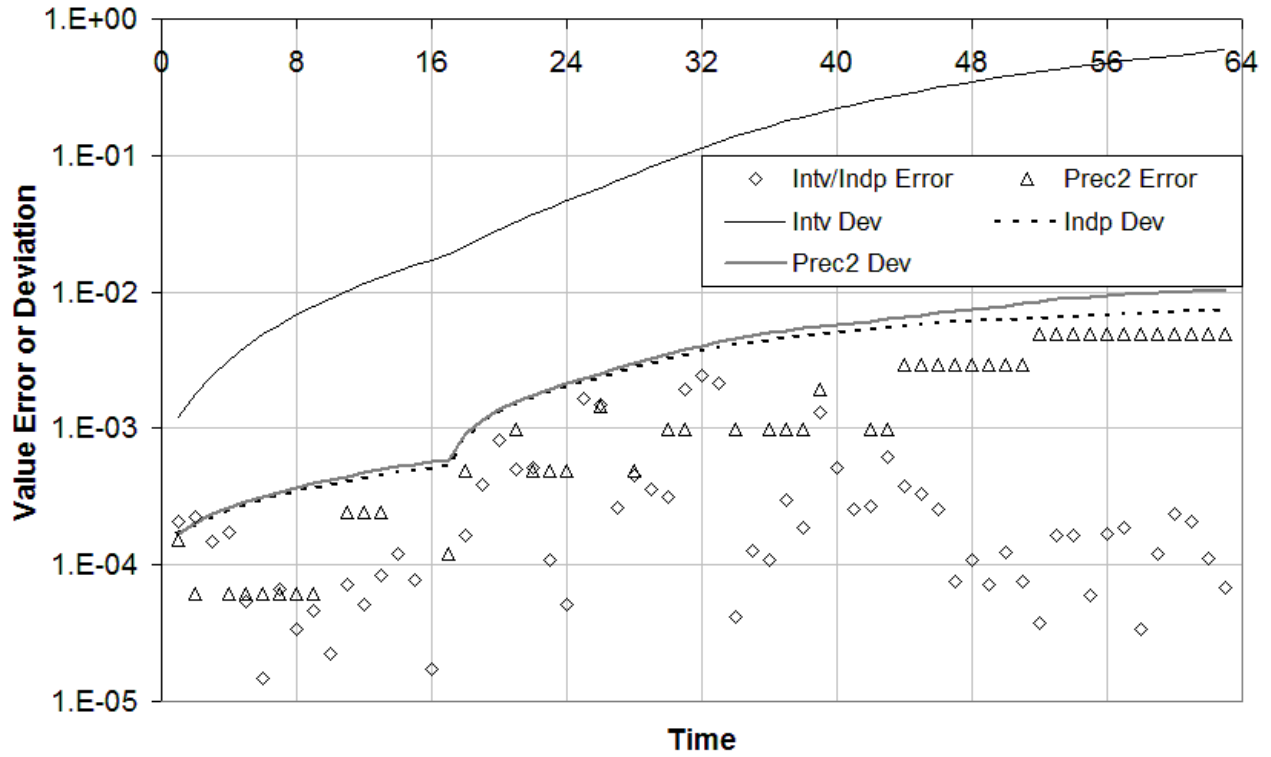


Figure 53: Output deviations and value errors vs. time for progressive moving-window linear regression of a straight line with 10-fold increase of input uncertainty in middle. In the legend, “Indp” means independence arithmetic, “Intv” means interval arithmetic, and “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty.

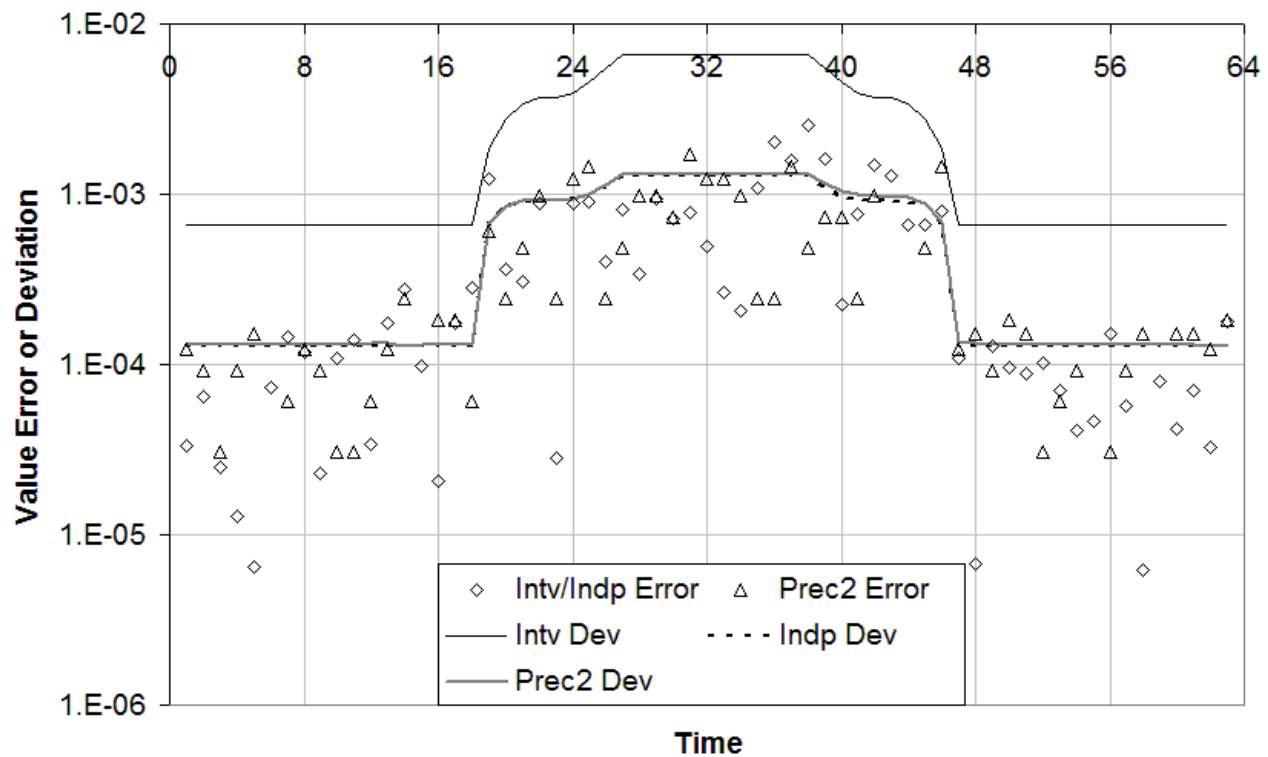


Figure 54: Output deviations and value errors vs. time for expressive moving-window linear regression of a straight line with 10-fold increase of input uncertainty in middle. In the legend, “Indp” means independence arithmetic, “Intv” means interval arithmetic, and “Prec2” means precision arithmetic with 2-bit calculated inside uncertainty.

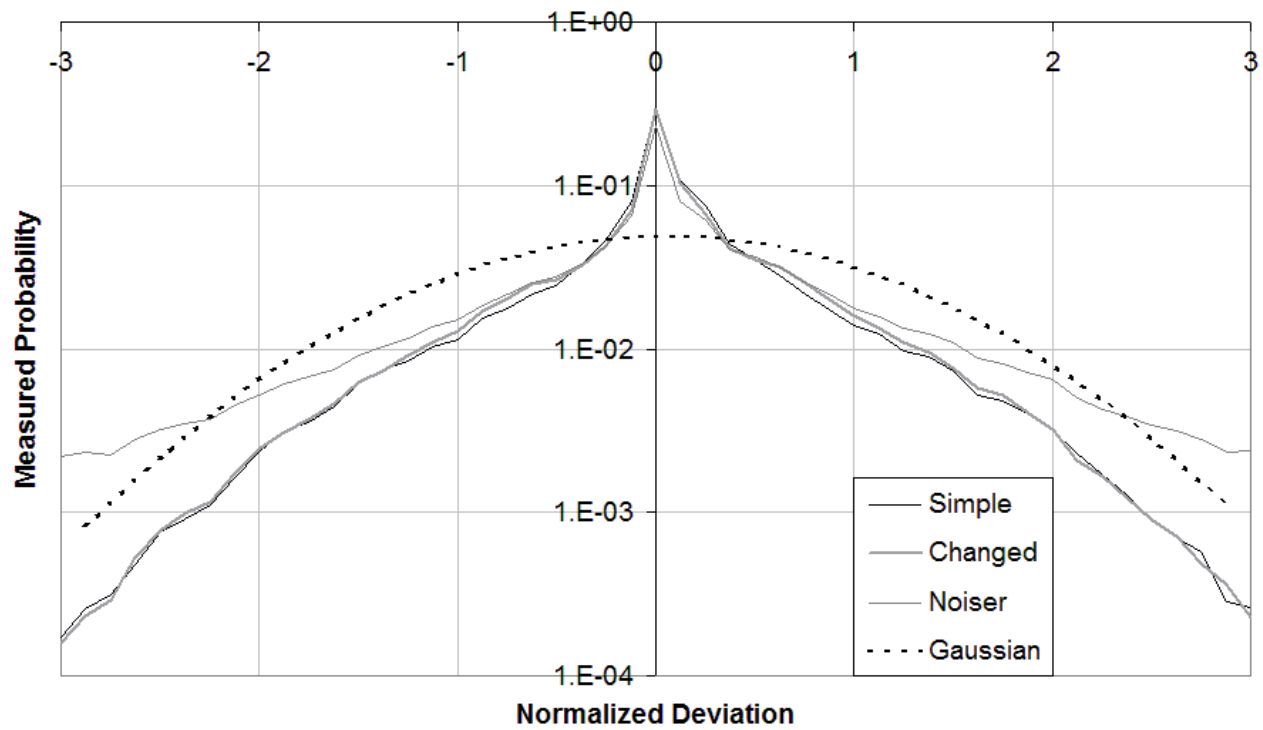


Figure 55: The measured normalized value error distributions using precision arithmetic with 4-bit calculated inside uncertainty for expressive linear regression. They are best fitted by a Gaussian distribution with mean of 0.06 and deviation of 0.73.

15 Appendix: Project Precision Arithmetic in Source Forge

The source codes using visual studio express 2008 are published as project Precision Arithmetic in source forge at <http://www.precisionarithm.sourceforge.com>. The source code can be divided into four parts:

15.1 RealTypes.lib

`RealTypes.lib` contains C++ template classes for precision arithmetic, independence arithmetic and interval arithmetic. Implicit interface `IReal` contains basic requirements for any uncertainty-bearing floating-point arithmetic. Each implementation of arithmetic has a short name.

Class `TInterval<unsigned t_Sigma=6>` and class `TIndp<unsigned t_Sigma=6>` provide implementations for independence arithmetic and interval arithmetic respectively, in which `t_Sigma` defines the σ rule of the relation between uncertainty deviation and bounding range of these two arithmetic. Their short names are “Intv<t_Sigma>” and “Indp<t_Sigma>” respectively, such as “Intv6” and “Indp6” for the 6σ - 10^{-9} rule. Class `TPrecision<unsigned t_ExtraSigBits=0, enum ECarry t_CarryWhenUncertain=CARRY_UNKNOWN>` implements precision arithmetic, in which `t_ExtraSigBits` is the extra significand bits calculated inside uncertainty, and `t_CarryWhenUncertain` defines the precision round up rule for a value of $(2S+1)?R@E$ as either `CARRY_UNKNOWN` or `CARRY_POSITIVE` or `CARRY_NEGATIVE`. Its short name is “Prec” followed by “<t_ExtraSigBits>” then followed by “+” or “-” when `t_CarryWhenUncertain` equals `CARRY_POSITIVE` or `t_CarryWhenUncertain` respectively. For example, the short name for a precision arithmetic with random precision round up rule and 2 bit calculation inside uncertainty is “Prec2”, while it is “Prec2+” when the precision round up rule is deterministic and positive.

In addition, `RealTypes.lib` contains some C++ function object for precision arithmetic, including (1) `TSquareMatrix` to hold a square matrix, and to calculate determinant and adjugate matrix according to Equation 6-1 and Equation 6-9; (2) `TFuncPolynom` to calculate a polynomial with constant coefficient and uncertain input according to Equation 3-31; and (3) `TFuncIntegral` to calculate numerical integral using depth-first binary-tree search for optimal integral intervals.

15.2 Precision.exe

`Precision.exe` tests any floating-point arithmetic in `RealTypes.dll` specified by a short name for an algorithm on an input signal.

- For FFT algorithms, the input signal is among “Sin”, “Cos”, “SinCos” or “Linear”, with additional noise type “GN” for Gaussian noise.
- For regression calculation of sine table, the input signal is “Phase”.
- For linear regression, the input signal is “RegrLine”.
- For Taylor expansion, the input signal is “Taylor”.
- For the matrix calculation, the input signal is “Matrix” followed by an integer indicating matrix size.
- For integral calculation, the input signal is “Intg” followed by (1) an integer indicating power of the function to be integrated, (2) a fractional value indicating the lower limits for integration, and (3) a fractional value indicating the upper limits for integration.

`Precision.exe` is usually executed through a batch file `Precision.bat`, which accepts a short name and tests the arithmetic for all input signals and noise types. File “/log<short name>.txt” and “/err<short name>.txt” in folder “../Output/<short name>” capture standard output and error output of the batch process, with the former contains execution times. The batch file `Precision.bat` also contains other possible arguments for `Precision.exe`.

15.3 Parser.exe

Parser.exe either selects or normalizes the large amount of output data from Precision.exe, and outputs files named as “<short name>_<signal>_<noise type>_<algorithm>.txt” in folder “../Output/<short name>/Parser”. It is usually executed through a batch file Parser.bat embedded in Precision.bat. Thus, only Precision.bat needs to be executed for each arithmetic short name.

15.4 ErrorTracing.exe

It accepts a minimal significand threshold, use Monte Carlo method to generate the probability distribution for bounding range of $1/2$, calculates theoretical rounding error distributions and round up leakage for different bounding ranges, and outputs files in folder “../Output/Tracking/<minimal significand threshold>”. Files named as “Range_<bounding range>_<method>.txt” contains theoretical rounding error distributions for the bounding range, in which <method> can be “p” for repeated addition, “m” for repeated subtraction, and “pm” for repeated addition followed by subtraction. File “All.txt” contains round up leakage for different bounding ranges. File “MonteCarlo” contains precision round up process using random integer as original value.

15.5 Data processing

Each output file contains tab-separated tables with column heading, which can be analyzed directly by any math package such as R statistics software. To cater for wider audiences, the output files are imported into excel files which plot the data in folder “../Doc/<short name>”. The first line of each work sheet that contains data shows the path and name of the output data file. For example, for FFT algorithms, folder “../Doc/Prec2” contains the following files:

- File “Prec2 FFT Sin Profile.xls”, “Prec2 FFT Cos Profile.xls”, “Prec2 FFT SinCos Profile.xls” and “Prec2 FFT Linear Profile.xls” display FFT results for a particular input deviation and signal frequency. Figure 9 is taken directly from “Prec FFT Sin Profile.xls”.
- File “Prec2 FFT Dist Total.xls” displays error significand distribution for each bounding range, input deviation and FFT algorithm. Figure 11 and Figure 12 are taken directly from “Prec2 FFT Dist Total.xls”.
- File “Prec2 FFT Dist Norm.xls” displays normalized error significand distribution. Figure 14 is taken directly from “Prec2 FFT Dist Norm.xls”.
- File “Prec2 FFT Sin.xls”, “Prec2 FFT Cos.xls”, “Prec2 FFT SinCos.xls” and “Prec2 FFT Linear.xls” displays the results for Forward, Roundtrip and Reverse algorithms for different input signal respectively. Figure 15, Figure 16, Figure 17, Figure 18, Figure 19, Figure 20, Figure 21, Figure 22, Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28 and Figure 29 are taken directly from these four “xls” files.

The outputs of `ErrorTracing.exe` are also analyzed by excel. File “../Output/Tracking/Tracking <minimal significand threshold>.xls” displays rounding error distributions and round up leakage for a particular bounding range. For example, Figure 4 and Figure 6 are taken from file “Tracking 0.xls”. File “../Output/Tracking/Tracking All.xls” compares characteristics of different minimal significand threshold, which contains Figure 3 and Figure 5.

15.6 Improvements Needed

The current implementation of precision arithmetic can be improved in several ways:

- Function `TPrecision::Normalize()` and `TPrecision::RoundUpTo()` apply precision round-up rule repeatedly. Because the rounded up significand is always the closest value to the original significand at each precision, they could calculate the rounded up significand at targeted exponent and then sum up the rounding errors in one step.

- To calculate significand and bounding range strictly, class `UInt` provides an implementation of unsigned integer with variable significant bits. A 64-bit unsigned integer is shown to provide very good approximation but with a 2-fold improvement on speed.
- Function `UInt::Mul()` and `UInt::Div()` can be replaced by few assembly instructions respectively.
- In current approach, an instance of template is first identified from a short name, then the corresponding instantiated template class or function is called. All available instances of template class and associated friend functions need to be instantiated in the static library `RealTypes.lib`, as well as in the clients of the library, `Precision.exe` and `Parser.exe`. This leads to repeated and cumbersome codes. `RealTypes.lib` should contain a type factory which outputs a type from an inputted short name, then use this type to instantiate template classes and functions in run-time. However, unlike C#, C++ does not allow instantiating template from type info in run-time.