

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 shown to be Gaussian. 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 available publicly at <http://precisionarithmetic.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 $10^{-2} \sim 10^{-4}$ of common measurements, to 10^{-14} of state-of-art measurements of basic physics constants [3].

Let x be a measured value, and Δx be the uncertainty for x . In scientific and engineering measurement x is usually sample mean and Δx is the sample deviation δx [1][2][4], though x

and Δx can also be raw reading¹ from an analog-to-digital converter (ADC) [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. $P \equiv \delta x/|x|$ is defined here as the (relative) precision² of the measurement, and its inverse is commonly known as the significance ϕ [1][2]. Precision represents reliable information content of a measurement. Finer precision means higher reliability thus better reproducibility of the 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. 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:

$$\begin{aligned} \text{Equation 2-2: } & (64919121 \times 205117922 - 159018721 \times 83739041) \\ & = 13316075197586562 - 13316075197586560 = 2.0000000000000000; \end{aligned}$$

1. The multiplication results exceed the maximal significance of the 64-bit IEEE floating-point representation, so they are rounded off, generating rounding errors;

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

² Precision here does not mean the maximal significand bit count as in common term “variable precision arithmetic”. It takes the traditional definition in measurement.

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 [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 [9][10], 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., faster in nonlinear algorithms than linear algorithms [11], propagation of rounding error in conventional floating-point arithmetic is very difficult to quantify generically [12] 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 [9][12][14], 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 floating-point results become insignificant. All existing uncertainty-bearing arithmetic is reviewed here.

2.3 Interval Arithmetic

Interval arithmetic [10][15][16] is currently a standard method to track calculation uncertainty. It assumes that a value is absolutely bounded within its bounding range $[x-R, x+R]$, using the following arithmetic equations [16]:

$$\text{Equation 2-3: } (x_1 \pm R_1) + (x_2 \pm R_2) = (x_1 + x_2) + (R_1 + R_2) ;$$

$$\text{Equation 2-4: } (x_1 \pm R_1) - (x_2 \pm R_2) = (x_1 - x_2) - (R_1 + R_2) ;$$

$$\text{Equation 2-5: } (x_1 \pm R_1) * (x_2 \pm R_2) = (x_1 * x_2) \pm |x_1 * x_2| \left(\frac{R_1}{|x_1|} + \frac{R_2}{|x_2|} + \frac{R_1}{|x_1|} \frac{R_2}{|x_2|} \right) ;$$

$$\text{Equation 2-6: } \frac{x_1 \pm R_1}{x_2 \pm R_2} = \frac{x_1}{x_2} \pm \left(\frac{R_1}{|x_2|} + \frac{R_2}{|x_2|} \left(\frac{|x_1|}{|x_2|} + \frac{R_2}{|x_2|} \right) \left(1 + 2 \frac{R_2}{|x_2|} \right) \right) / \left(1 - \frac{R_2}{|x_2|} \right) ;$$

If it is implemented using a floating-point representation with limited resolution, the result bounding range is widened further [16].

A basic problem is that the bounding range used by interval arithmetic is not compatible with usual scientific and engineering measurements, which use statistical deviation instead of absolute bounding range. Most measured values are well approximated by Gaussian 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 Gaussian 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. For example, in addition and subtraction, it gives the result when the two operands are +1 and -1 correlated respectively. However, if the two operands are -1 and +1 correlated respectively instead, the actually bounding range after addition and subtraction reduces. The vast overestimation of bounding ranges in these two cases prompts the developments of affine arithmetic [17][18], 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.

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 the statistical correlation between two operands is known, the result uncertainty is given by statistical propagation of uncertainty [19], 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 :

$$\text{Equation 2-7: } (x_1 \pm \sigma_1) + (x_2 \pm \sigma_2) = (x_1 + x_2) \pm \sqrt{\sigma_1^2 + \sigma_2^2 + 2\sigma_1\sigma_2\gamma} ;$$

$$\text{Equation 2-8: } (x_1 \pm \sigma_1) - (x_2 \pm \sigma_2) = (x_1 - x_2) \pm \sqrt{\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\gamma} ;$$

$$\text{Equation 2-9: } (x_1 \pm \sigma_1) * (x_2 \pm \sigma_2) = (x_1 * x_2) \pm |x_1 * x_2| \sqrt{P_1^2 + P_2^2 + 2P_1P_2\gamma} ;$$

$$\text{Equation 2-10: } \frac{x_1 \pm \sigma_1}{x_2 \pm \sigma_2} = \frac{x_1}{x_2} \pm \left| \frac{x_1}{x_2} \right| \sqrt{P_1^2 + P_2^2 - 2P_1P_2\gamma} ;$$

In practice the correlation between two operands is generally not precisely known, so direct use of statistical propagation of uncertainty is 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 2-7, Equation 2-8, Equation 2-9 and Equation 2-10 but with $\gamma=0$.

Another special case of statistical propagation of uncertainty is the arithmetic operation between an operation $x \pm \sigma$ and itself:

$$\text{Equation 2-11: } (x \pm \sigma) + (x \pm \sigma) = 2x \pm 2\sigma = 2 \cdot (x \pm \sigma) ;$$

$$\text{Equation 2-12: } (x \pm \sigma) - (x \pm \sigma) = 0 ;$$

$$\text{Equation 2-13: } (x \pm \sigma) * (x \pm \sigma) = x^2 \pm 2|x|\sigma = (x \pm \sigma)^2 ;$$

$$\text{Equation 2-14: } \frac{x \pm \sigma}{x \pm \sigma} = 1 ;$$

2.5 Significance Arithmetic

Significance arithmetic [20] tries to track reliable bits in an imprecise value during calculation. Except early attempts [21][22], 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 [20].

Also can be categorized as significance arithmetic, a stochastic arithmetic [12][23] 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 [24] 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 the other two uncertainty-bearing arithmetics in Section 5 using FFT algorithms. A brief discussion is provided in Section 6.

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 operands can be regarded as independent of each other. This assumption can be turned into a realistic statistical requirement on input data to precision arithmetic.

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 and precise constant. It is concluded from Equation 2-9 and Equation 2-10 for statistical propagation of uncertainty when $\sigma_2=0$ thus $P_2=0$.

3.2 The Independent Uncertainty Assumption

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 Cov() be covariance and γ be correlation:

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

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

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

Equation 3-2 states the correlation γ between two signals, each of which contains 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 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 signal is less at the 2nd digit than at the 3rd digit. Quantitatively, according to Equation 3-1:

- 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. The above conclusion is based on digit content of measured signals, so it is also applicable when the measured signals contain a constant fold of real signal.

Similarly, while the correlated portion between two operands has exactly same value at each bit of the two operands, the ratio η of uncorrelated portion to the correlated portion increases by 2-fold for each bit down from MSB of the two operands. If two significant operands are overall correlated by γ , at level of uncertainty the correlation between the two operands decreases to γ_ϕ :

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

In Equation 3-3, ϕ is the smaller significance of the two operands. Figure 2 plots the relation of γ vs. ϕ for each given γ_ϕ in Equation 3-3. If when γ_ϕ is less than a maximal threshold (e.g., 2%, 5% or 10%), the two operands can be deemed virtually independent of each other at the level of uncertainty, then there is a maximal allowed correlation between any two operands for each ϕ , as shown in Figure 2. If two operands 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 operands. Figure 2 shows that for two precisely measured operands, their correlation γ is allowed to be quite high. While to be acceptable in precision arithmetic, each of low-resolution operands 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 operands to be independent of each other, which is the assumption for independence arithmetic.

It is tempting to add white noise to otherwise unqualified operands to make their uncertainties independent of each other. As an extreme case of this approach, if two operands are constructed by adding noise to a same signal, they are 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. For a given γ_ϕ , there is an upper limit of correlation γ for each ϕ , beyond which adding white noises to operands will not make their correlation less than the γ_ϕ threshold.

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 significand 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 $S+@E+1$.

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 added to the existing rounding error. Since the newly generated rounding error always compensates for the existing rounding error, the rounding error range is limited to half bit of significand for the precision round up rule. Thus the rounded up significand is always the closest value to the original significand at each precision.

The precision arithmetic also tracks rounding error bounding range R , so that the precision representation becomes $S \sim R @ E$.

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 $S ? 1/2 @ E$.

A round down is applicable to a precise value $S @ E$. Immediately after a round down, the value becomes $2S @ E - 1$. A precise value should be rounded down repeatedly until it has largest possible value of $|S|$, in the same way as the normalization of a conventional floating-point value. An imprecise value can be decomposed as a precise value plus an imprecise zero:

$$\text{Equation 3-4: } S \sim R @ E = S @ E + 0 \sim R @ E;$$

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][10][12]. 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 $[-1/2, +1/2]$, 65536 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 uniform within range $(-1/2, +1/2)$. However, the precision round up rule changes such equal chance for few lowest bits of significand. For example, a value of $1+\epsilon$ is always rounded up to $1-\epsilon+1$ while a value $1-\epsilon$ is always rounded up to $0+\epsilon+1$. 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 16 and above.

3.6 Result Bounding For Addition and Subtraction

To a floating-point arithmetic, rounding errors are uncertainties [8][10][12]. 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 operands. 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:

$$\text{Equation 3-5: } S_1 \sim R_1 \epsilon \pm S_2 \sim R_2 \epsilon = (S_1 \pm S_2) \sim (R_1 + R_2) \epsilon;$$

Table 1 shows the result \sim for addition while Table 2 shows the result \sim for subtraction.

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

$$\text{Equation 3-6: } P_{1/2}(x) \equiv 1, \quad -1/2 \leq x \leq +1/2;$$

Then the density function $P_{n/2}(x)$ for rounding error distribution of bounding range $R \equiv n/2$ is obtained by convolution:

$$\text{Equation 3-7: } P_{n/2}(x) \equiv \int_{-\infty}^{+\infty} P_{1/2}(y) P_{(n-1)/2}(x-y) dy = \int_{-1/2}^{+1/2} P_{(n-1)/2}(x-y) dy, \quad n=2, 3, 4, \dots;$$

It is easy to prove that the deviation σ of $P_{n/2}(x)$ is determined by its bounding range:

$$\text{Equation 3-8: } \sigma^2 = R/6;$$

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

$$\text{Equation 3-9: } P_{(m+n)/2}(x) = \int_{-\infty}^{+\infty} P_{m/2}(y) P_{n/2}(x-y) dy ;$$

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 3-9. Addition and subtraction have slightly different result distribution due to uneven $P_{1/2}(x)$. 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 3-8. 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 ($y = a * x + b$) of a generic probability density function $D_X(x)$ is [4]:

$$\text{Equation 3-10: } D_Y(y) = D_X((y-b)/a) / a ;$$

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] and Equation 3-10:

$$\text{Equation 3-11: } P_{n/2}(x\sqrt{\frac{n}{12}})\sqrt{\frac{n}{12}} \xrightarrow{d} N(x);$$

After applying transformation of ($y = x/\sigma$) to both sides, in which σ is defined in Equation 3-8:

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

Equation 3-12 shows that $P_{n/2}(y)$ converges to a stable rounding error distribution of Gaussian distribution with larger n due to central limit theorem.

3.7 Result Bounding for Rounding Up and Rounding Down

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 3-8, 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 as a special application of the scaling principle, round up by deviation results in a much similar rounding error distribution. Round up by deviation is also required by the stable rounding error distribution of Equation 3-12.

Round up by deviation also introduces bounding leakage called round-up 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. Figure 7 actually shows that the convergence of Equation 3-12 is quite fast.

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 normalized if its R is in the range of $[R_{\max}/4, R_{\max})$. Otherwise it is called a nearly precise value. When $R_{\max}=16$, the normalization leakage is 10^{-6} , which is small enough for most applications.

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:

$$\text{Equation 3-13: } \delta x = \sigma 2^E;$$

$$\text{Equation 3-14: } \Delta x = R 2^E;$$

$$\text{Equation 3-15: } \delta x / \Delta x = \sigma / R = 1 / \sqrt{6R};$$

According to Equation 3-12 and Equation 3-13, the stable probability density function $\rho(y)$ of uncertainty distribution of a normalized precision value $x \pm \delta x$ is provided by Equation 3-16, in which y is a random variable, and $N(y)$ is the standard normal distribution:

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

The round down rule can be extended to an imprecise value $S \sim R @ E$ using the scaling principle. After round down once, $S \sim C @ E$ 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 is normalized afterward. If the operand with larger exponent is normalized, the result exponent is at least as large as the larger operand exponent. According to Equation 3-5, subtraction is just addition between two precision values of opposite signs. The commutative rule for addition is obviously satisfied.

3.8 Bounding Initiation

An integer S is first initialized as a precise value $S@0$ before it is normalized.

A conventional 64-bit floating-point value $S@E$ is 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 $S \sim R@E$ using Equation 3-13, while a median-range pair $(x \pm \Delta x)$ is initialized as an imprecise value $S \sim R@E$ using Equation 3-14. If the precision of the measured value is coarser than 10^{-16} , the result value is normalized. Practically all measured values are normalized.

Independence arithmetic and interval arithmetic both use conventional floating-point arithmetic to calculate their values, which assumes that each mean value is completely precise without uncertainty. In contrast, when $R_{\max}=16$, precision arithmetic does not calculate inside its uncertainty, and the resolution of its significand is determined by its precision. For example:

- $0.5 \pm 0.001 = +512?6.29@-10$;
- $1 \pm 0.001 = +1024?6.29@-10$;
- $1 \pm 0.002 = +512?6.29@-9$;

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 of significands:

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

3.9 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 $S_1 \sim_1 R_1 @ E_1$ by $S_2 @ E_2$ is defined as:

$$\text{Equation 3-17: } S_1 \sim_1 R_1 @ E_1 \times S_2 @ E_2 \equiv S_1 S_2 \sim_1 S_2^2 R_1 @ (E_1 + E_2) ;$$

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:

$$\text{Equation 3-18: } 0 \sim_1 R_1 @ E_1 \times 0 \sim_2 R_2 @ E_2 \equiv 0 (\sim_1 \sim_2) R_1 R_2 @ (E_1 + E_2) ;$$

$$\text{Equation 3-19: } S_1 \sim_1 R_1 @ E_1 \times S_2 \sim_2 R_2 @ E_2 = (S_1 S_2) \sim R @ (E_1 + E_2) ;$$

$$\sim R = \sim_1 R_1 S_2^2 + \sim_2 R_2 S_1^2 + (\sim_1 \sim_2) R_1 R_2 ;$$

Equation 3-18 extends the result distribution of the cross term to be ρ -distributed. The non-cross terms in Equation 3-19 are ρ -distributed as result of round by deviation, so that the result of multiplication is also ρ -distributed.

The result precision P of multiplication is:

$$\text{Equation 3-20: } P^2 = \frac{1}{6} \frac{|\sim_1 R_1 S_2^2 + \sim_2 R_2 S_1^2 + (\sim_1 \sim_2) R_1 R_2|}{(S_1 S_2)^2} = |\sim_1 P_1^2 + \sim_2 P_2^2 + (\sim_1 \sim_2) \frac{1}{6} P_1^2 P_2^2| ;$$

Equation 3-20 is different from Equation 2-9 due to difference in their statistical requirements:

- When $\sim_1 = \sim_2$, the result precision using precision arithmetic is coarser than that using independence arithmetic due to the accumulation of operand rounding errors..
- When $\sim_1 \neq \sim_2$, the result precision using precision arithmetic is finer than that using independence arithmetic due to the canceling of operand rounding errors.

The commutative rule for multiplication is obviously satisfied. The associative rule between multiplication and addition is also satisfied.

3.10 Result Bounding For Division

The reverse of Equation 3-17 defines:

$$\text{Equation 3-21: } S_1 \sim_1 R_1 @ E_1 / S_2 @ E_2 \equiv (S_1 / S_2) \sim_1 (R_1 / S_2^2) @ (E_1 - E_2) ;$$

In Equation 3-21, 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 times before the division.

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

$$\text{Equation 3-22: } S_1 @ E_1 / S_2 \sim R_2 @ E_2 \equiv (S_1 / S_2) (-\sim_2) (R_2 S_1^2 / S_2^4) @ (E_1 - E_2) ;$$

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

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

$$\begin{aligned} \text{Equation 3-23: } S_1 \sim R_1 @ E_1 / S_2 \sim R_2 @ E_2 &= (S_1 / S_2) \sim R @ (E_1 - E_2) ; \\ \sim R &= \sim_1 R_1 / S_2^2 - \sim_2 R_2 S_1^2 / S_2^4 - (\sim_1 \sim_2) R_1 R_2 / S_2^4 ; \end{aligned}$$

The result precision P of division is:

$$\text{Equation 3-24: } P^2 = \frac{1}{6} \frac{|\sim_1 R_1 / S_2^2 - \sim_2 R_2 S_1^2 / S_2^4 - (\sim_1 \sim_2) R_1 R_2 / S_2^4|}{(S_1 / S_2)^2} = |\sim_1 P_1^2 - \sim_2 P_2^2 - (\sim_1 \sim_2) \frac{1}{6} P_1^2 P_2^2| ;$$

Equation 3-24 is different from Equation 2-10 due to difference in their statistical requirements:

- When $\sim_1 \neq \sim_2$, the result precision using precision arithmetic is coarser than that using independence arithmetic due to the accumulation of operand rounding errors..
- When $\sim_1 = \sim_2$, the result precision using precision arithmetic is finer than that using independence arithmetic due to the canceling of operand rounding errors.

3.11 Function Evaluation

The uncertainty of function $f(x)$ at x is evaluated by the set $S \equiv \{f(x+y) - f(x) : y \in \rho(x)\}$, in which y is a random variable whose distribution is defined by Equation 3-16:

- If $f(x)$ is monotonic or can be divided into limited monotonic sections within range $[x-\Delta x, x+\Delta x]$, Δf is evaluated directly.
- If function $f(x)$ can be expressed analytically in x , δf may be able to be expressed analytically in x and δx . Even if x appears in $f(x)$ multiple times, the similar dependency problem when using interval arithmetic [25] will not present because no dependency assumption is required.
- If function $f(x)$ is a generic function, both Δf and δf can be found by sampling, because it is very easy to divide range $[x-\Delta x, x+\Delta x]$ into any number of quantiles [4]. For example, a κ -point monotonic sampling assumes that a monotonic region contains at least consecutive κ sampling points. First, the input range is divided into κ quantiles, to see if sampled $f(x)$ contains only monotonic region(s). If not, each quantile is further divided into κ quantiles, and so on. δf is then calculated as sample deviation, and Δf is searched among extreme values of all monotonic region(s).
- Other mathematical methods such as Taylor expansion may be used to obtain Δf and δf . It is a wish that such analysis may give a meaningful definition of uncertainty of $f(x)$ when range $[x-\Delta x, x+\Delta x]$ is actually outside the definition range of $f(x)$.

It is assumed that the result uncertainty of $f(x)$ is also ρ -distributed. One necessary check for this assumption is Equation 3-25:

$$\text{Equation 3-25: } \sqrt{3} \delta f \leq \Delta f \leq \sqrt{6R_{\max}} \delta f;$$

3.12 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 with a fixed denominator of 2^9 so that $R_{\max}=16$.

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 for interval arithmetic. With code weight-trimming and optimization, its speed is expected to be comparable to that of interval arithmetic. Unlike conventional floating-point arithmetic, it only calculates limited number of significand bits, e.g., 10 bits instead of all 53 bits when the precision is 10^{-3} . 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]. Thus, a hardware implementation of precision arithmetic should have comparable speed to that of a conventional floating-point arithmetic.

When the addresses of the two operands are identical, the arithmetic equation for precision arithmetic is Equation 2-11, Equation 2-12, Equation 2-13 and Equation 2-14.

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 less than but 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 for a data set 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.

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.

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 , and vice versa. Such 6σ rule has comparable bounding leakage of 10^{-9} as that of precision arithmetic with $R_{\max}=16$.

The statistical assumption of precision arithmetic is weaker than that of independence arithmetic but stronger than that of interval arithmetic, so the 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 3-8 and Equation 3-5, 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 3-20 and Equation 3-24, the result precision of multiplication or division using precision arithmetic is comparable to that of independence arithmetic. Thus, the result of precision arithmetic should be much closer to that of independence arithmetic.

4.2 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 with an index frequency of 2 has 2^4 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 error propagation. An alternative to Gaussian noise is white noise, which has the same bounding range as the input signal.

5 Validating Uncertainty-Bearing Arithmetics Using FFT

5.1 FFT (Fast Fourier Transformation)

When $N = 2^L$, in which L is a positive integer, the generalized Danielson-Lanczos lemma [9] can be applied in Fourier transformation as FFT (Fast Fourier Transformation), to decompose a discrete signal $h[k]$ in time domain to components $H[n]$ in frequency domain [9]:

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

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

$$\text{Equation 5-3: } H[n] = H[n, \frac{k}{2^0}];$$

Thus, each output value is obtained after applying Equation 5-2 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 5-2 can be simplified. Let << denotes bit left-shift operation and let & denote bitwise AND operation:

$$\text{Equation 5-4: } \phi[n] \equiv e^{i2\pi \cdot n \cdot \frac{1}{2^L}}; \quad e^{i2\pi \cdot n \cdot \frac{1}{2^{L-m}}} = \phi[(n \ll m) \& ((1 \ll L) - 1)];$$

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 error is introduced from trigonometric functions.

All the test input signals are real, so that:

$$\text{Equation 5-5: } H[n] = H[N - n]^*, \quad n=1, 2, \dots N/2;$$

If signal is periodic, the period count of the signal is called index frequency in this paper. The Fourier spectrum of a sine signal with index frequency f is ($H[n] = \delta_{f,n} - i f N/2$), while it is ($H[n] = \delta_{f,n} f N/2$) for a cosine signal. For arithmetic test using periodic signals, only integer index frequencies up to less than the Nyquist frequency [9] are used.

A linear signal with slope s ($h[j] = s j$) is also used in the test, whose Fourier spectrum is

$$(H[n] = -s \frac{N}{2} (1 + \frac{i}{\tan(\pi \cdot n / N)})), \text{ in which } n=1, 2, \dots N/2.$$

5.2 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 index 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 [9]:

$$\text{Equation 5-6: } H[n] = \sum_{k=0}^{N-1} h[k] \cdot e^{i2\pi \cdot n \cdot \frac{k}{N}}$$

In Equation 5-6, n is index frequency. The corresponding reverse transformation is [9]:

$$\text{Equation 5-7: } h[k] = \frac{1}{N} \sum_{n=0}^{N-1} H[n] \cdot e^{-i2\pi \cdot k \cdot \frac{n}{N}}$$

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

$$\text{Equation 5-8: } H[n] = \frac{\sum_{k=0}^{N-1} e^{i2\pi \cdot (n+f) \cdot \frac{k}{N}} - \sum_{k=0}^{N-1} e^{i2\pi \cdot (n-f) \cdot \frac{k}{N}}}{2i}$$

When f is an integer F , $H[n] = i\delta_{n,F}N/2$. Otherwise:

$$\text{Equation 5-9: } H[n] = \frac{1}{2} \frac{\sin(2\pi f - 2\pi \frac{f}{N}) + \sin(2\pi \frac{f}{N}) - \sin(2\pi f) \cdot e^{-i2\pi \frac{n}{N}}}{\cos(2\pi \frac{n}{N}) - \cos(2\pi \frac{f}{N})}$$

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

$$\text{Equation 5-10: } |\Delta f| \rightarrow 0: H[n] \rightarrow i\delta_{n,F}N/2;$$

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

Discrete Fourier transformation $H[n]$ of a signal $h[k]$ is the digital implementation of continuous Fourier transformation $H(s)$ of signal $h(t)$, in which $H(s) = i\delta(s-f)$ for $h[k] = \sin(2\pi \cdot f \cdot k)$. From the above equations, when the frequency of the original signal falls between two frequency indexes of the transformation, the peak is lower and wider, depending on the fractional frequency. Thus, a discrete Fourier transformation is only faithful for signal

components with exactly one of the index frequency 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 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. 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.

Due to its width, an unfaithful representation 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 [9][12][14] may generate artifacts due to its uniform treatment of faithful and unfaithful signal components, which probably coexist in reality.

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

$$\text{Equation 5-11: } h(t) \Leftrightarrow H(s): \quad h(t-\tau) \Leftrightarrow H(s) \exp(i2\pi s \tau);$$

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 $h[k]$, $k = 0 \dots N-1$:

$$\text{Equation 5-12: } G[n] = (H[n] + h[N] - h[0]) \exp(i2\pi n/N);$$

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 a shift in phase. Unlike aliasing [9][12][14], unfaithful representation of discrete Fourier transformation has equal effect in whole frequency range.

5.3 Evaluating Uncertainty-Tracking

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. 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. The result deviation of Equation 5-2 only depends on the deviations of the two operands. Thus, 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 5-2 are no longer a constant. One interesting consequence is that only in precision arithmetic the output deviations for a noisy input signal are noticeably larger than those for a clean input signal.

The occurrence frequencies of output value error significand are counted for different bounding ranges and normalized as measured probability densities. The theoretical rounding error distributions for different bounding ranges obtained from Equation 3-12 are integrated around each integer value of rounding error, to result in theoretical probability densities of value error significand for different bounding ranges. Figure 10 compares such three sets of probability densities. It shows that the theoretical and measured probability densities agree well with other in all cases, confirming that the rounding error distribution of precision arithmetic is well defined and actually simple. Because precision arithmetic does not calculate extensively inside uncertainty, the error significand distributions in Figure 10 have largest population at 0, which corresponds to 0 value error, while using conventional floating-point arithmetic the value

errors are usually very small but non-zero. Thus, even precision arithmetic may have larger maximal error value, it may still have comparable average error value as that of conventional floating-point arithmetic.

Figure 11 shows that for a same input deviation, the output deviations of forward FFT increases exponentially with FFT order using all three arithmetics. Figure 12 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 index frequency of input signal, so that all data of a same input deviation and FFT order but different frequencies can be pooled together during analysis. The trends in Figure 11 and Figure 12 is modeled by Equation 5-13, in which L is FFT order, δx is input deviation, δy is output average deviation, α and β are empirical fitting constants:

$$\text{Equation 5-13: } \delta y = \alpha \beta^L \delta x;$$

β measures propagation speed of deviation with increased amount of calculation in Equation 5-13. It is called propagation base rate. Unless β is close to 1, β dominates α in fitting thus determines characteristics of Equation 5-13.

It turns out that Equation 5-13 is very good fits for both average output deviations and value errors for all three arithmetics, such as demonstrated in Figure 13. 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 5-14 and Equation 5-15, 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 5-13 for average output deviations and value errors respectively:

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

$$\text{Equation 5-15: } \alpha = (\alpha_{\text{err}} / \alpha_{\text{dev}}); \quad \beta = (\beta_{\text{err}} / \beta_{\text{dev}});$$

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 14, 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 FFT order. Equation 5-13 and Equation 5-14 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 5-2.
- 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 5-13 and Equation 5-16, in which $(\alpha_{\text{for}}, \beta_{\text{for}})$ are corresponding fitting parameters of Equation 5-13 for forward FFT, while the output average error significands is expected to obey Equation 5-14 with same α and β as those of forward FFT.

Equation 5-16: $\alpha = \alpha_{\text{for}}; \quad \beta = \beta_{\text{for}}/2;$

Roundtrip FFT is forward FFT followed by reverse FFT, with the output deviation of forward FFT as input deviation to reverse FFT. Thus, both its average output deviations and value errors are expected to fit Equation 5-13 and Equation 5-17, in which $(\alpha_{\text{for}}, \beta_{\text{for}})$ and $(\alpha_{\text{rev}}, \beta_{\text{rev}})$ are corresponding fitting parameters of Equation 5-13 for forward FFT and reverse FFT respectively. And its error significands are expected to fit Equation 5-14 and Equation 5-17, in which $(\alpha_{\text{for}}, \beta_{\text{for}})$ and $(\alpha_{\text{rev}}, \beta_{\text{rev}})$ are corresponding fitting parameters of Equation 5-14 for forward FFT and reverse FFT respectively.

Equation 5-17: $\alpha = (\alpha_{\text{for}} \quad \alpha_{\text{rev}}); \quad \beta = (\beta_{\text{for}} \quad \beta_{\text{rev}});$

Figure 15, Figure 16 and Figure 17 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 5-15. The estimated β for average uncertainty deviations and value errors for Reverse and Roundtrip algorithms are obtained from Equation 5-16 and Equation 5-17 respectively. The measured β and the estimated β agree well with each other in all cases, confirming the notion that uncertainty-tracking is a rational competition between error propagation and uncertainty propagation:

- Equation 5-2 sums up two mutually independent operands, so the error propagation in a FFT algorithm is precisely tracked by independence arithmetic. Figure 15 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.
- 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 both average output deviations and value errors is a few percents larger than the corresponding β s of independence arithmetic, so both its average output deviations and error values propagate slightly faster with increased FFT order than those of independent arithmetic.
- Interval arithmetic has β for average output deviations always much larger than β for average output value errors, resulting in β for average output significands of about 0.62 for forward and reverse FFT, and about 0.39 for roundtrip FFT. So its average output deviations propagate much faster with the amount of calculations than its value error does.

Figure 18 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. Precision arithmetic thus provides better uncertainty tracking than interval arithmetic in this case.

5.4 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 5-13 well experimentally using any arithmetic. Equation 5-14 and Equation 5-15 can be used to estimate maximal bounding ratio as well. For example, Figure 19 shows that the maximal output bounding ratios using precision arithmetic fits Equation 5-14 well. Unlike average output error significands in Figure 14, using precision arithmetic, the maximal output bounding ratios increases slowly with FFT order.

To characterize trend of maximal output bounding ratios with FFT order, β for measured average output deviation vs. 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 20 shows the characterization plot for independence arithmetic while Figure 21 shows the characterization plot for precision arithmetic. In both characterization plots, β for maximal value error exceeds β for maximal uncertainty deviation, suggesting that maximal bound ratio should increase with FFT order. Figure 23 shows that the output maximal bounding ratios indeed increase with FFT order of forward FFT for both arithmetics. Independence arithmetic has a constant β of 1.05 for maximal bounding ratio, with a bounding leakage about 10^{-9} by statistical estimation. Because β s for maximal bounding ratio of precision arithmetic exceeds those of independence arithmetic by about 10%, the maximal bounding ratios of

precision arithmetic exceeds those of independence arithmetic with increased FFT order. When maximal bounding ratios are larger than 1, the empirical bounding leakages using precision arithmetic are approximately a constant about 10^{-4} , which is independent of algorithm, input deviation, input signal type, FFT order, and the value of maximal bounding ratio. The reason for this empirical invariant is still not clear at this moment.

As shown in Figure 22 and Figure 23, 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. In contrast, precision arithmetic has approximate fixed bounding leakages about 10^{-4} with much tighter bounding ranges and stable maximal bounding ratios. Which arithmetic is better for uncertainty-bounding depends on the statistical requirements on calculation bounding leakage. Since most input data already contain bounding leakage, precision arithmetic should be more suitable for normal usages.

5.5 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 $S+@E+1$, although both have identical result in error tracking as the random precision round-up rule. Figure 24 shows the characterization plot of precision arithmetic using deterministic precision round up rule. Compared with Figure 21, it shows that these two deterministic precision round-up rules are worse in error bounding than the random precision round-up rule, and they distinguish type of input data. For forward FFT on a sine input signal, β for maximal output value errors has distinctive larger increase at it imaginary part than real part. In contrast, for forward FFT on a cosine input signal, such distinctive larger increase of β is at it real part instead its imaginary part, as shown in Figure 25. Such difference can not be found for reverse FFT. Forward FFT condenses a sine signal into only two non-zero imaginary values by mutual cancellation of signal components, while reverse

FFT spread only two non-zero imaginary values to construct a sine signal. Thus forward FFT is more sensitive to calculation errors than reverse FFT. Such difference can also not be found using conventional floating-point arithmetic, suggesting that precision arithmetic has larger worst-case value errors than conventional floating-point arithmetic. In retrospect, the value errors are quantum using precision arithmetic while they are continuous using conventional floating-point arithmetic in Figure 9. So the behavior of no calculation inside uncertainty using precision arithmetic could be a source for such worse maximal bounding ratios.

5.6 Calculation inside Uncertainty using Precision Arithmetic

Table 3 summarizes the characteristics of precision arithmetic using different R_{\max} . With larger R_{\max} , a uncertainty deviation corresponds to a larger bounding range according to Equation 3-15. Table 3 lists the bounding ranges of normalized precision values and their ratios to uncertainty deviation for each R_{\max} . Assuming rounding error is Gaussian distributed, Table 3 also lists the corresponding bounding leakage, which is comparable to the theoretical maximal normalization bounding leakage for each R_{\max} according to Figure 7. Larger R_{\max} also increase S in precision representation $S \sim R @ E$ for the same precision. When $R_{\max}=256$, precision arithmetic calculates 2 bits inside uncertainty, as shown in the following example:

- $0.5 \pm 0.001 = +2048?101@-12;$
- $1 \pm 0.001 = +4096?101@-12;$
- $1 \pm 0.002 = +2048?101@-11;$

Figure 26 shows the characterization plot of precision arithmetic when $R_{\max}=256$. Figure 26 is very similar to Figure 20, suggesting that precision arithmetic with $R_{\max}=256$ is approaching the performance of the ideal uncertainty-bearing arithmetic in this case. This similarity is remarkable because these two arithmetics have very different arithmetic equations. Compared with Figure 21, it also suggests that the quantum effect of value error due to not calculating inside uncertainty has largely gone after calculating 2-bit inside uncertainty. The normalization

bounding leakage is 10^{-92} according to Figure 7, which can be regarded as virtually 0 in most applications. Thus, precision arithmetic with $R_{\max}=256$ should be the arithmetic of choice whenever independent uncertainty assumption holds while the independence between two operands is questionable.

6 Conclusion and Discussion

Equation 5-13 is expected to describe general value error propagation for linear algorithms in which L is the amount of calculations [11]. For all three arithmetics, the average and maximal output deviation and value errors are shown to obey Equation 5-13, while the average output error significands and maximal bounding ratios are shown to obey Equation 5-14 and Equation 5-15. FFT algorithms provide a good linear platform to test any uncertainty-bearing arithmetic, with a clearly defined L as FFT order and a known error propagation mechanism. For FFT algorithms, precision arithmetic seems more suitable for uncertainty-tracking as well as uncertainty-bounding than interval arithmetic in normal usages. Precision arithmetic with 2-bit calculation inside uncertainty gives almost ideal performance for both uncertainty tracking and uncertainty bounding.

The deterministic precision round up rules have worse result in uncertainty bounding than that of random precision round up rule, suggesting that they can be optimized further.

Before applying it generally, precision arithmetic still needs more ground works and testing. However, it seems a worthwhile alternative to interval arithmetic. It should be tested further in other problems, such as improper integrations and solutions to differential equations.

As an independent researcher, the author of this paper feels indebted to encouragements and valuable discussions with his wife Dr. Yingxia Wang, 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.

7 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] William H. Press, Saul A Teukolsky, Willaim T. Vetterling, and Brian P. Flannery, Numerical Recipes in C (Cambridge University Press, 1992)
- [10] Oliver Aberth, Precise Numerical Methods using C++, Academic press, 1998
- [11] Gregory L. Baker and Jerry P. Gollub, Chaotic Dynamics: An Introduction (Cambridge University Press, 1990)
- [12] J. Vignes, A stochastic arithmetic for reliable scientific computation, Mathematics and Computers in Simulation vol. 35, pp 233-261, 1993
- [13] B. Liu, T. Kaneko, Error analysis of digital filters realized with floating-point arithmetic, Proc. IEEE, vol. 57, pp 1735-1747, Oct.1969
- [14] B. D. Rao, Floating-point arithmetic and digital filters, IEEE, Trans. Signal Processing, vol 40, pp 85-95, Jan, 1992

- [15] R.E. Moore, Interval Analysis, Prentice Hall, 1966
- [16] W. Kramer, A prior worst case error bounds for floating-point computations, IEEE Trans. Computers, vol. 47, pp 750-756, Jul. 1998
- [17] J. Stolfi, L. H. de Figueiredo, An introduction to affine arithmetic, TEMA Tend. Mat. Apl. Comput., 4, No. 3 (2003), 297-312.
- [18] http://amp.ece.cmu.edu/Publication/Fang/icassp2003_fang.pdf
- [19] Propagation of uncertainty, http://en.wikipedia.org/wiki/Propagation_of_uncertainty.
- [20] Significance Arithmetic, http://en.wikipedia.org/wiki/Significance_arithmetic.
- [21] Max Goldstein, Significance Arithmetic on a Digital Computer, Communications of the ACM, Volume 6 (1963), Issue 3, Page 111-117
- [22] R. L. Ashenhurst and N. Metropolis, Unnormalized Floating-point Arithmetic, Journal of the ACM, Volume 6 (1959), Issue 3, Page 415-428
- [23] R. Alt, and J.-L. Lamotte, and Markov, S, Some experiments on the evaluation of functional ranges using a random interval arithmetic, Mathematics and Computers in Simulation, 2001, Vol. 56, pp. 17-34.
- [24] Chengpu Wang, Error Estimation of Floating-point Calculations by a New Floating-point Type That Tracks the Errors, AMCS'05
- [25] W. Krämer, Generalized Intervals and the Dependency Problem, Proceedings in Applied Mathematics and Mechanics, Vol. 6, Nr. 1, p. 685-686, Wiley-InterScience (2006).

8 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 $S1 \sim 1R1@E + S2 \sim 2R2@E = (S1+S2) \sim (R1+R2)@E$.

\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 $S1 \sim 1R1@E - S2 \sim 2R2@E = (S1-S2) \sim (R1+R2)@E$.

R_{\max}	16	64	256
Normalized bounding range / R_{\max}	[4, 16)	[16, 64)	[64, 256)
Normalized bounding range / deviation	[4.9, 9.8)	[9.8, 19.6)	[19.6, 39.2)
Normalized bounding leakage by assuming Gaussian distribution	$[10^{-6}, 10^{-22})$	$[10^{-22}, 10^{-85})$	$[10^{-85}, \sim 0)$
Measured maximal normalization bounding leakage	10^{-6}	10^{-23}	10^{-92}
Bit calculation inside uncertainty	0	1	2

Table 3: Characterization of precision arithmetic with different R_{\max}

9 Figures

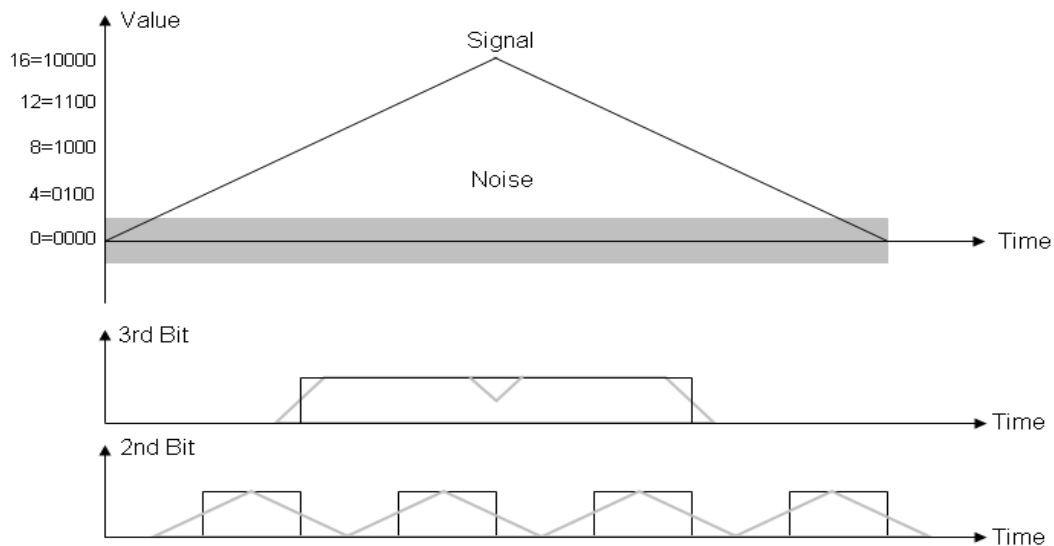


Figure 1: Effect of noise on bit values of a measured value.

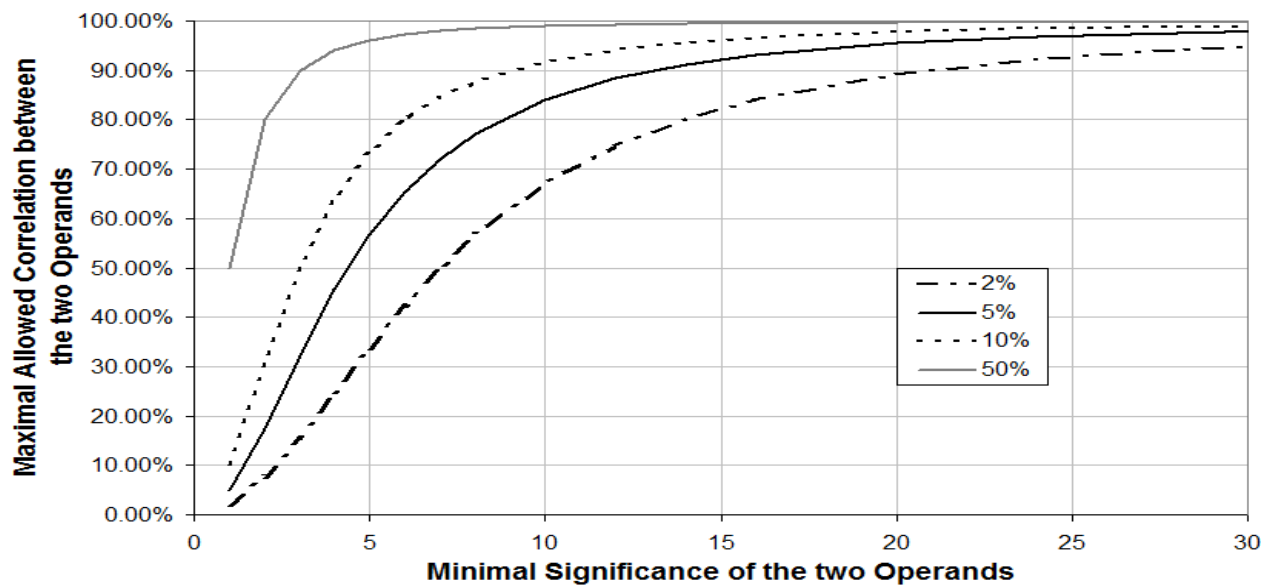


Figure 2: Allowed maximal correlation between two operands for different significance for precision arithmetic

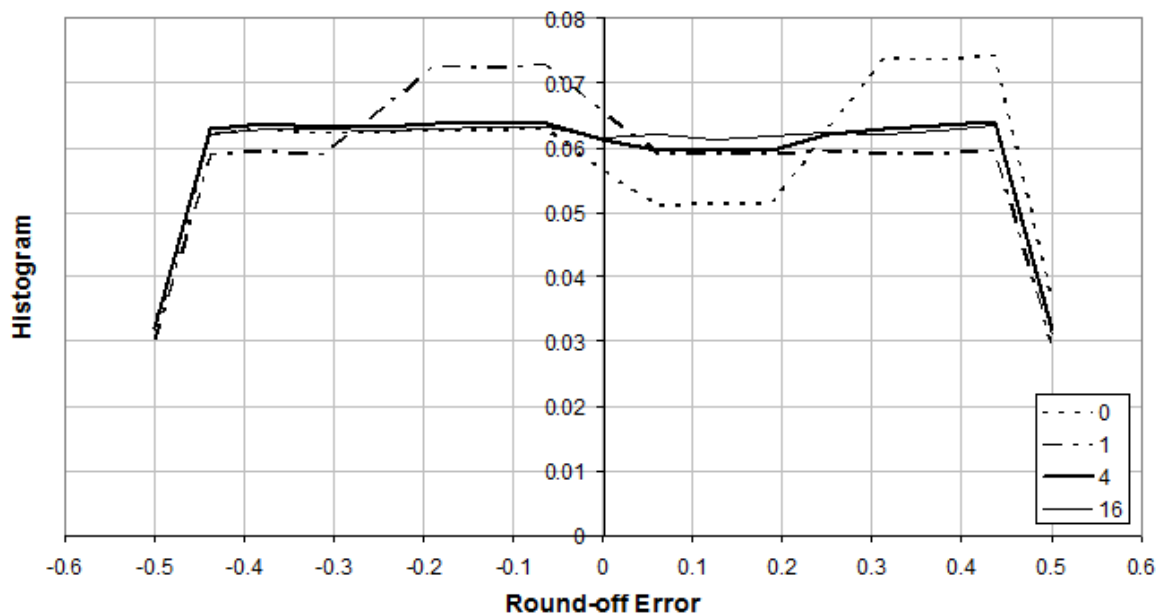


Figure 3: Measured probability distribution of rounding errors of precision round-up rule for the minimal significand thresholds 0, 1, 4, and 16 respectively.

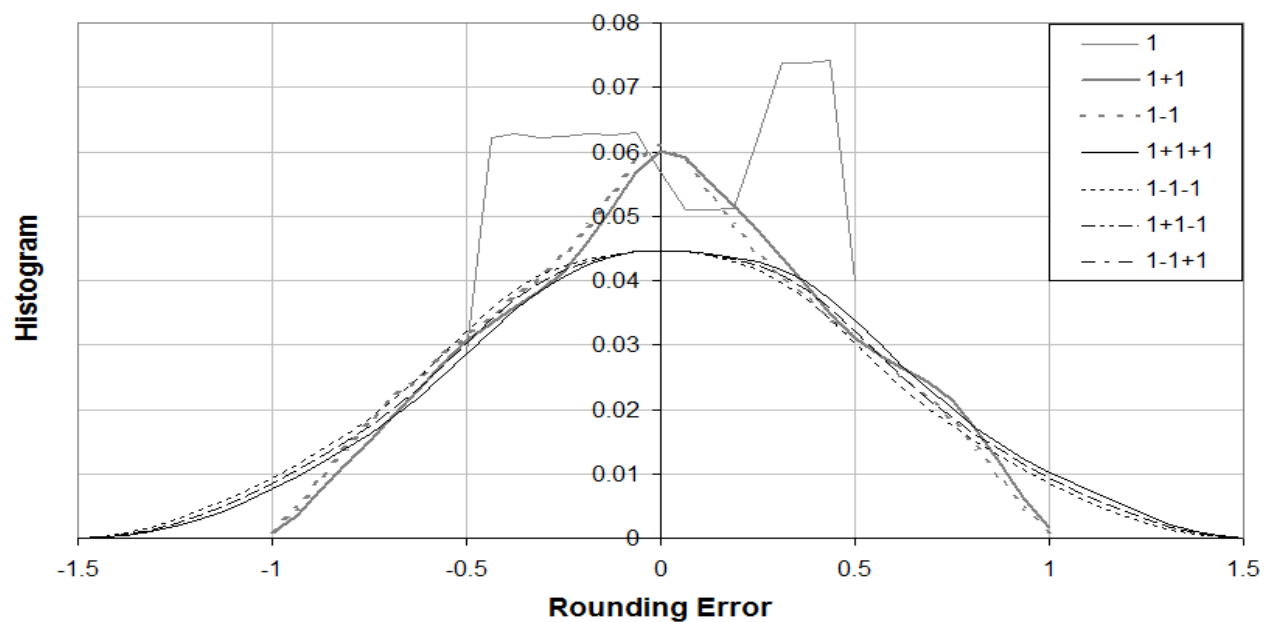


Figure 4: Measured probability distribution of rounding error after addition and subtraction.

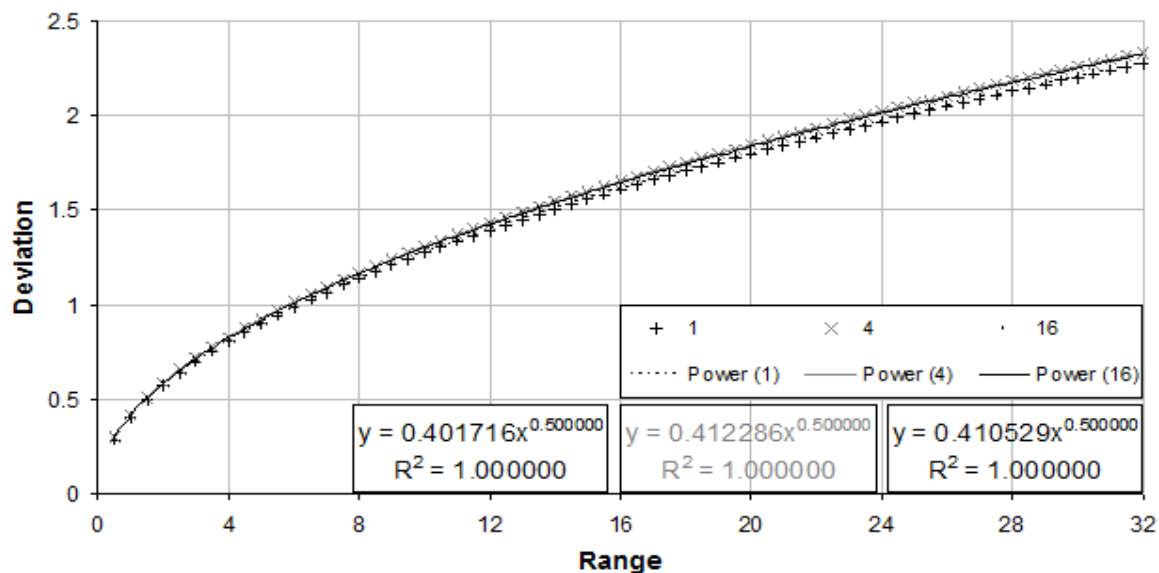


Figure 5: Deviations vs. bounding ranges of measured rounding error distributions of different minimal significant thresholds.

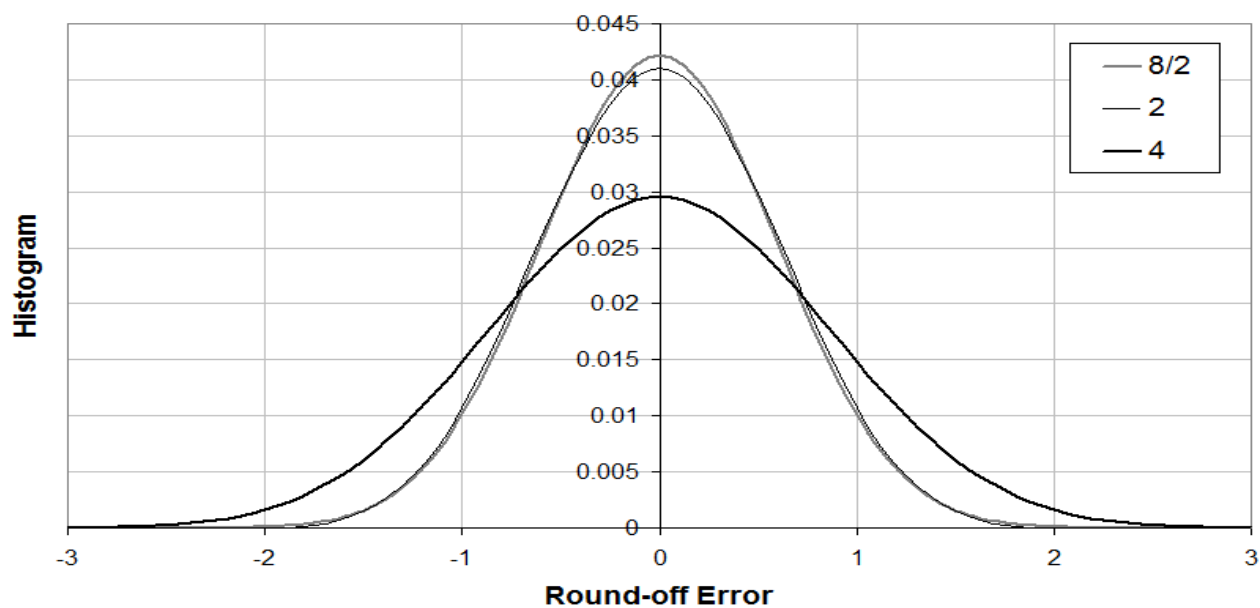


Figure 6: The result rounding error distribution $R=8/2$ after the original error distribution $R=8$ is rounded up once. The $R=8/2$ distribution is compared with the $R=4$ distribution and the $R=2$ distribution, which have same bounding range and deviation respectively.

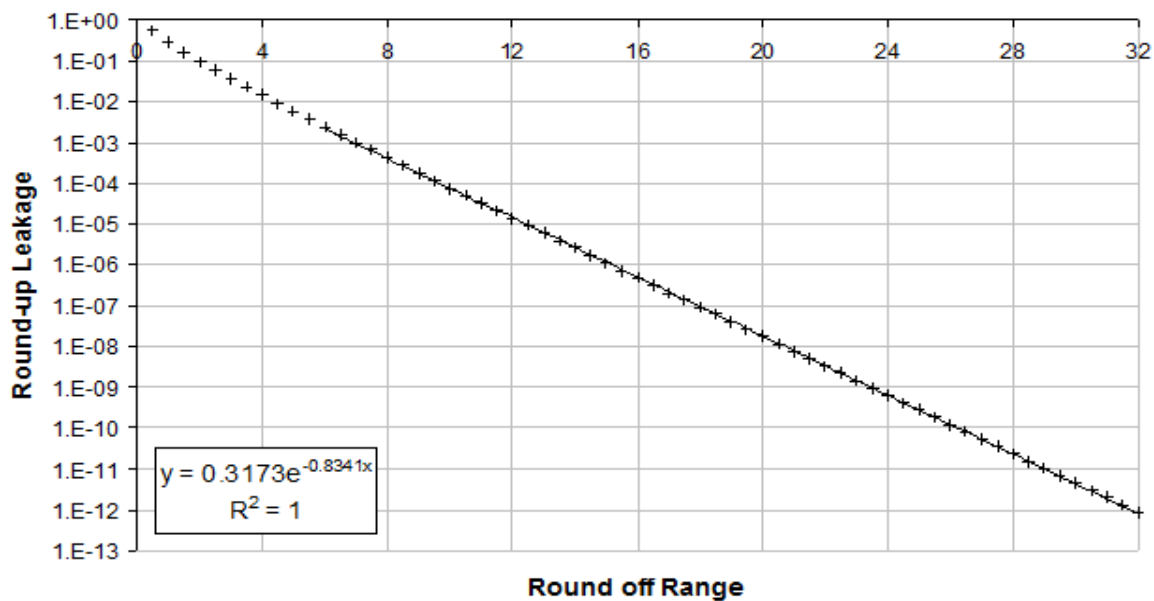


Figure 7: Round-up leakage vs. bounding ranges after rounding up by deviation once.

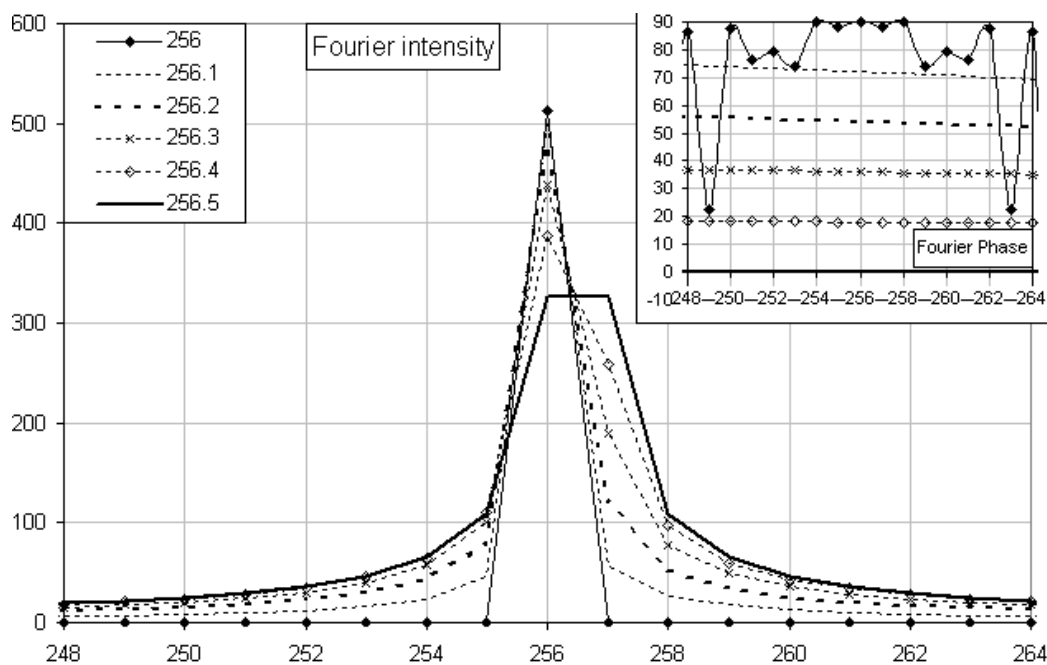


Figure 8: Unfaithful representations of perfect sine signals in discrete Fourier transformation.

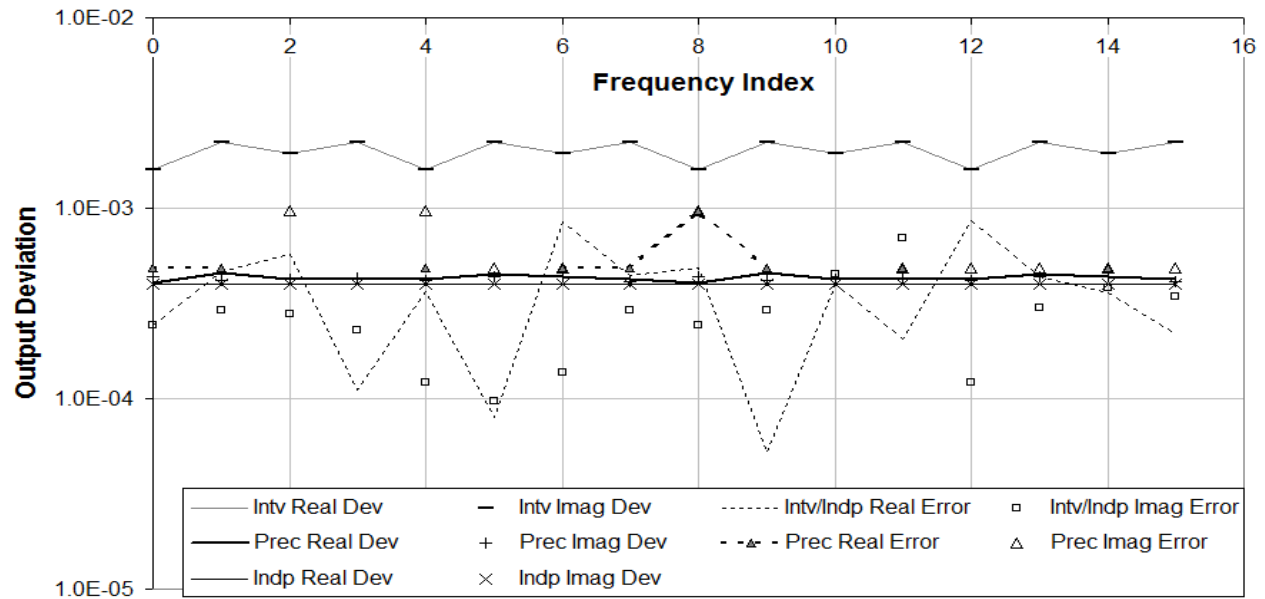


Figure 9: The output deviations and value errors of forward FFT on a noisy signal of index frequency 1 and input deviation 10^{-4} .

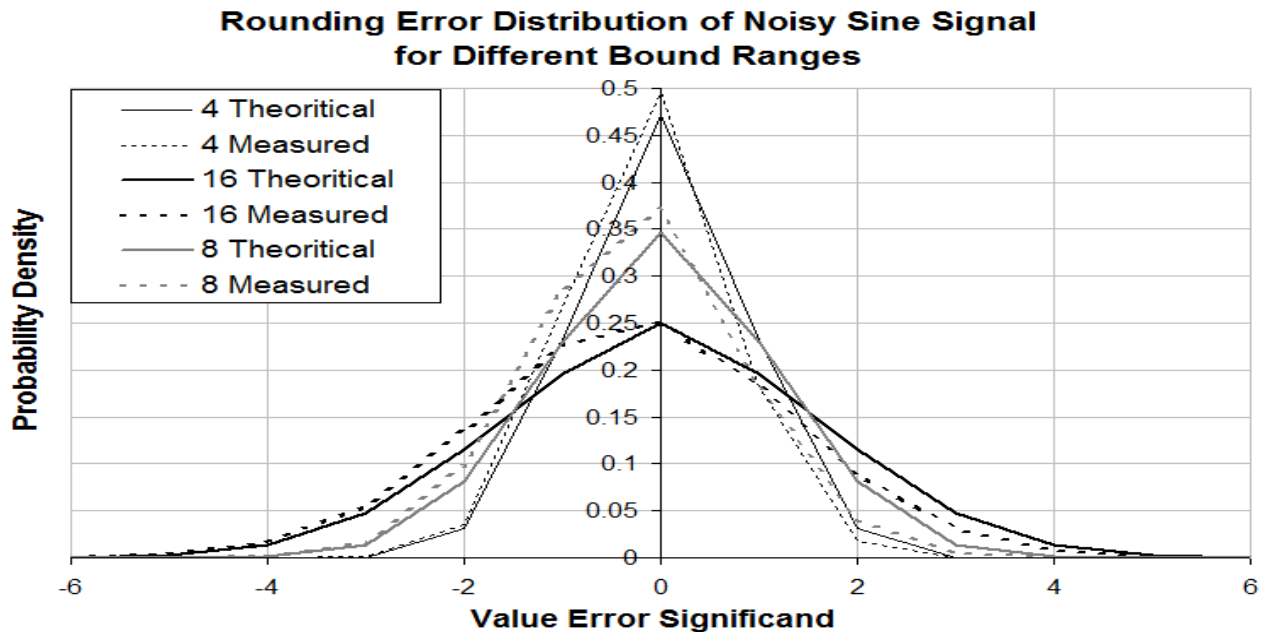


Figure 10: For different bounding ranges, the measured output probability distributions of error significands agree with the theoretical ones respectively.

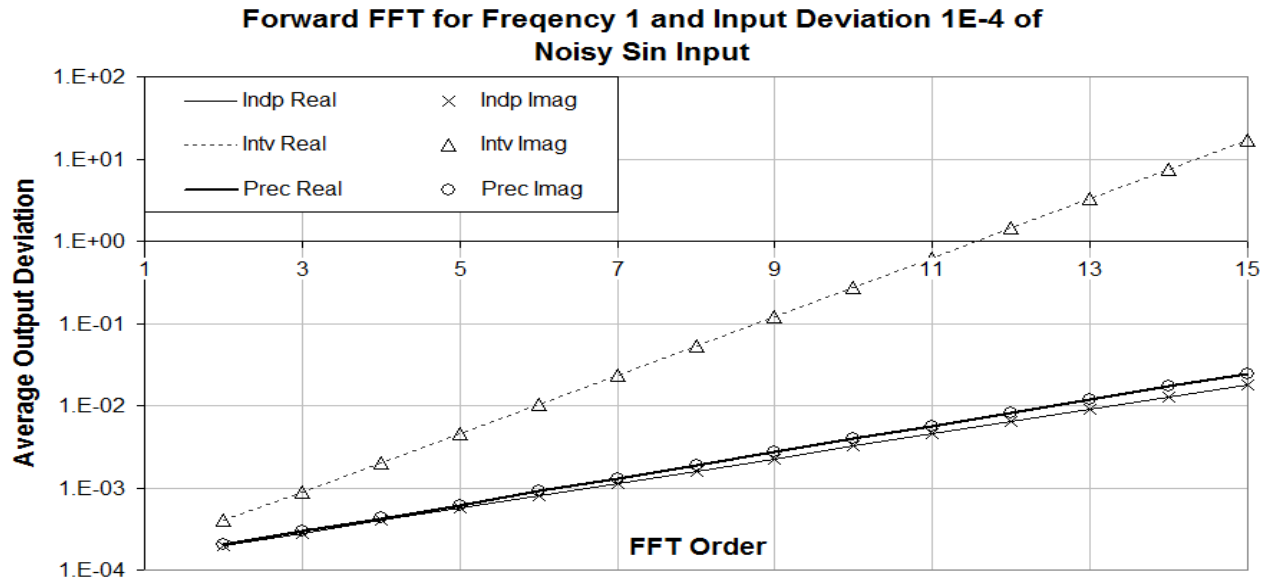


Figure 11: For a same input deviation, the average output deviations of forward FFT increase exponentially with FFT order.

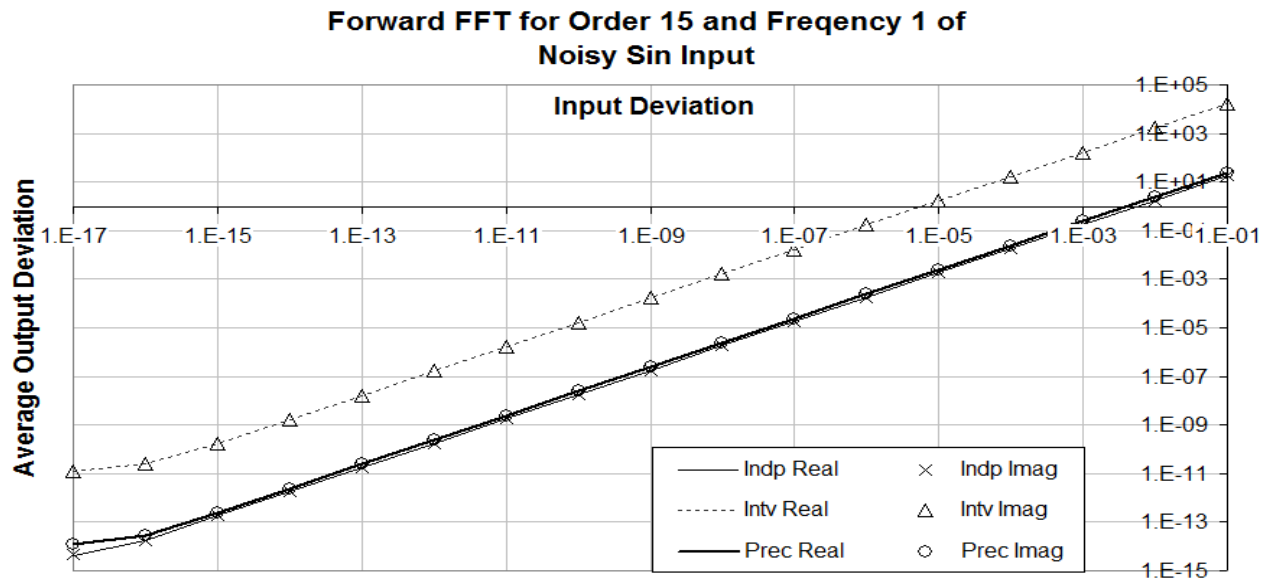


Figure 12: For a same order of FFT calculation, the average output deviations of forward FFT increases linearly with input deviation.

Output Real Average Value Error for Noisy Sin Signal

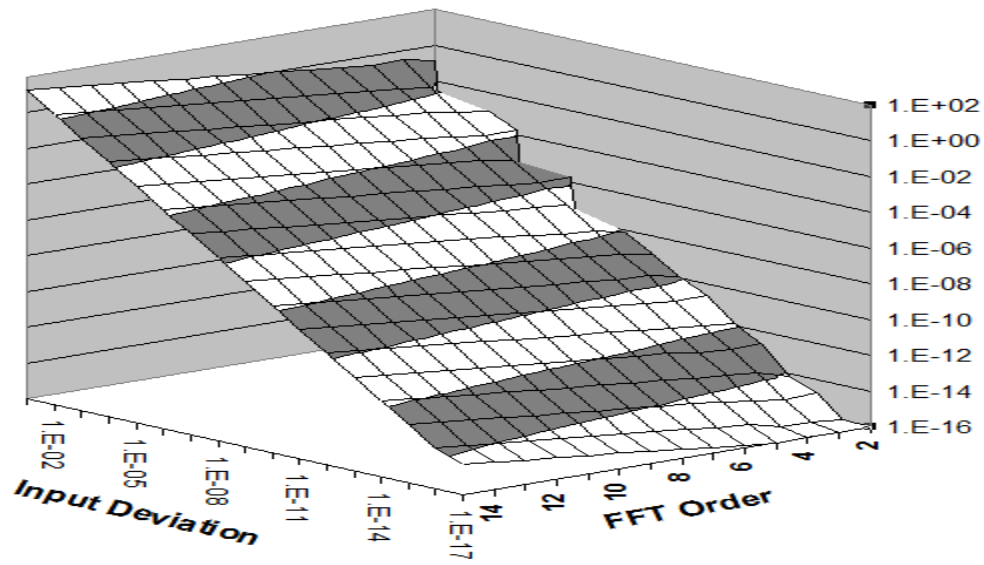


Figure 13: The average output value errors using precision arithmetic increase exponentially with FFT order and linearly with input deviation respectively.

Output Real Average Error Significand for Noisy Sin Signal

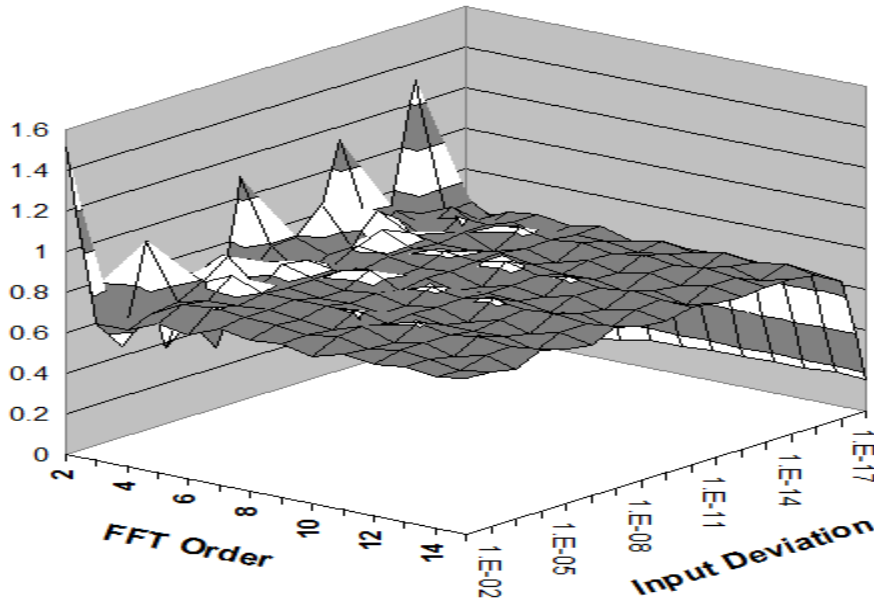


Figure 14: The average output error significands using precision arithmetic is a constant when the input deviation is larger than 10^{-15} and the FFT order is more than 4.

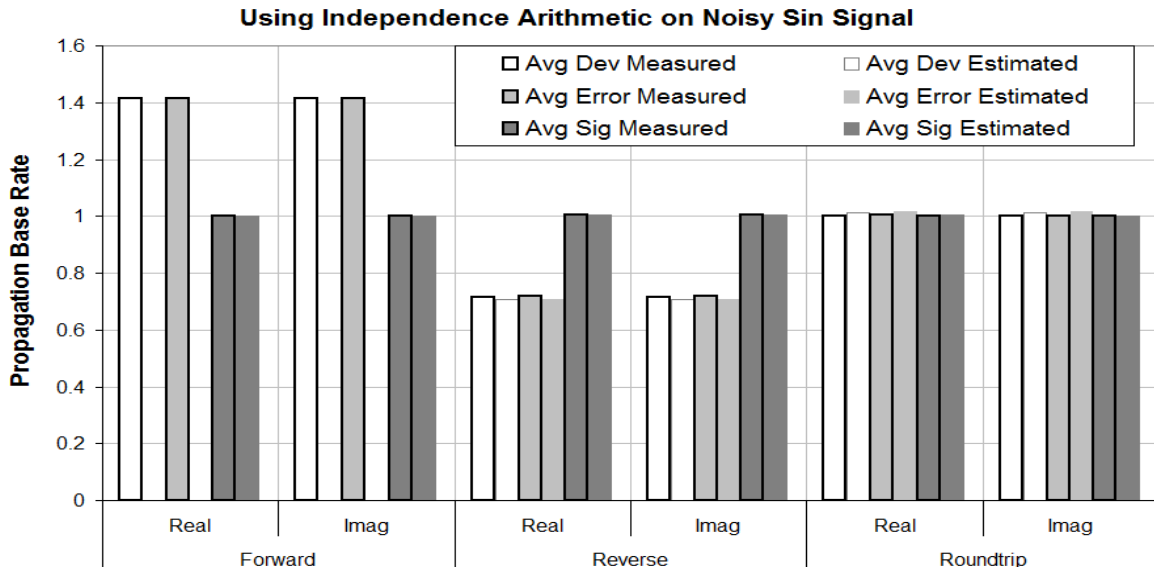


Figure 15: β for fitting average output deviations, value errors and error significands for forward, reverse and roundtrip FFT using independence arithmetic on noisy sine signals.

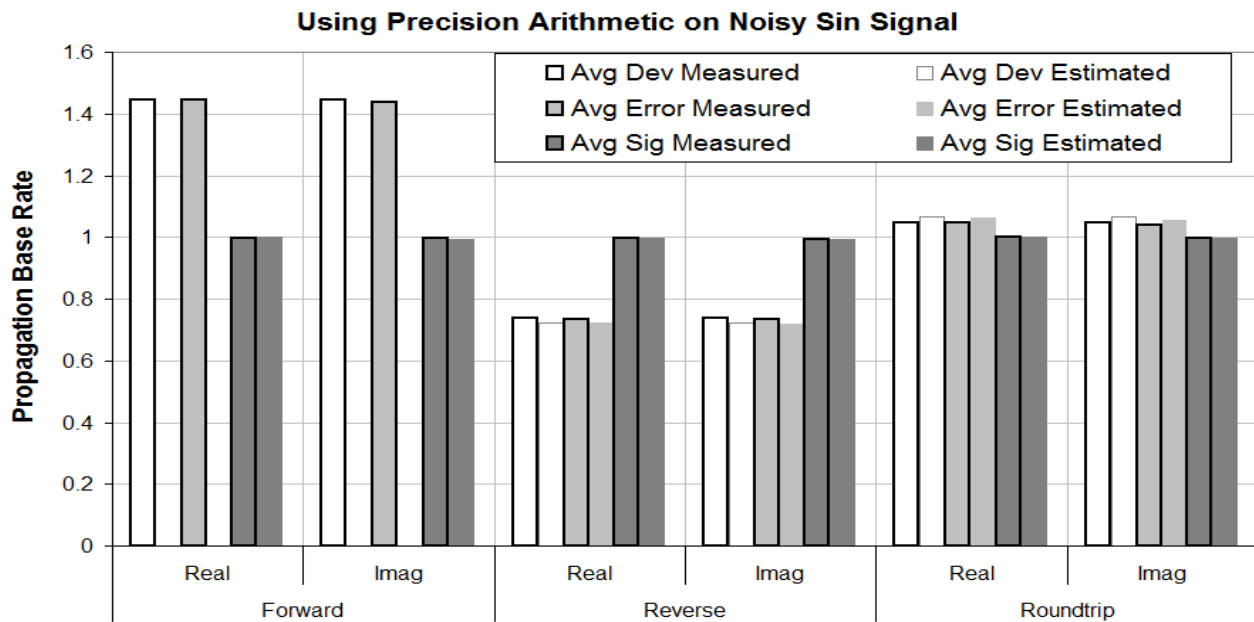


Figure 16: β for fitting average output deviations, value errors and error significands for forward, reverse and roundtrip FFT using precision arithmetic on noisy sine signals.

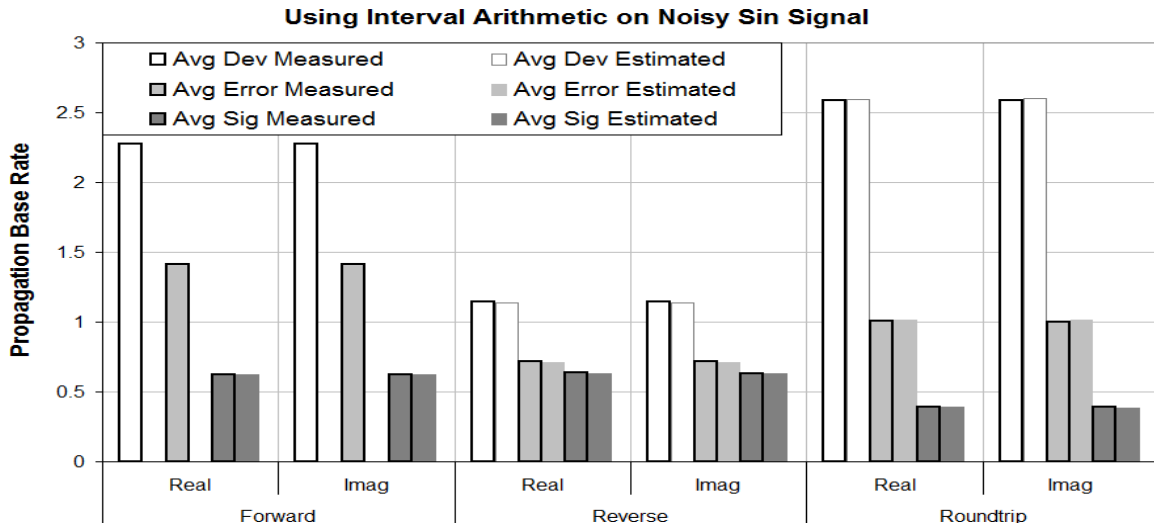


Figure 17: β for fitting average output deviations, value errors and error significands for forward, reverse and roundtrip FFT using interval arithmetic on noisy sine signals.

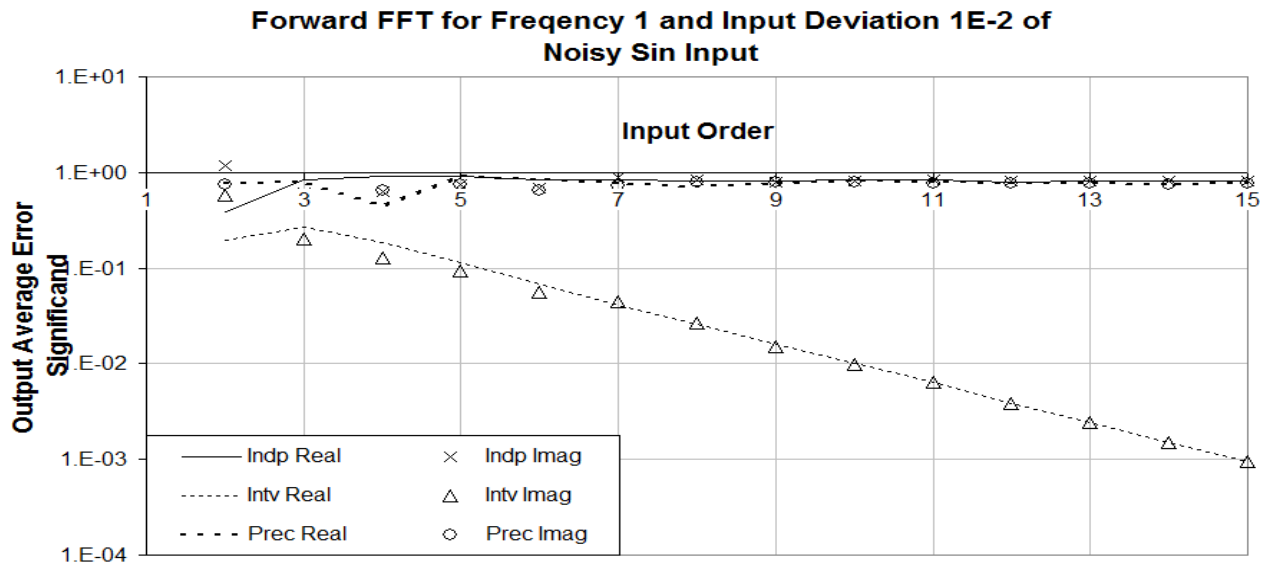


Figure 18: The output average error significands vs. FFT order of forward FFT for all three arithmetics.

Output Imag Max Error Bounding Ratio for Noisy Sin Signal

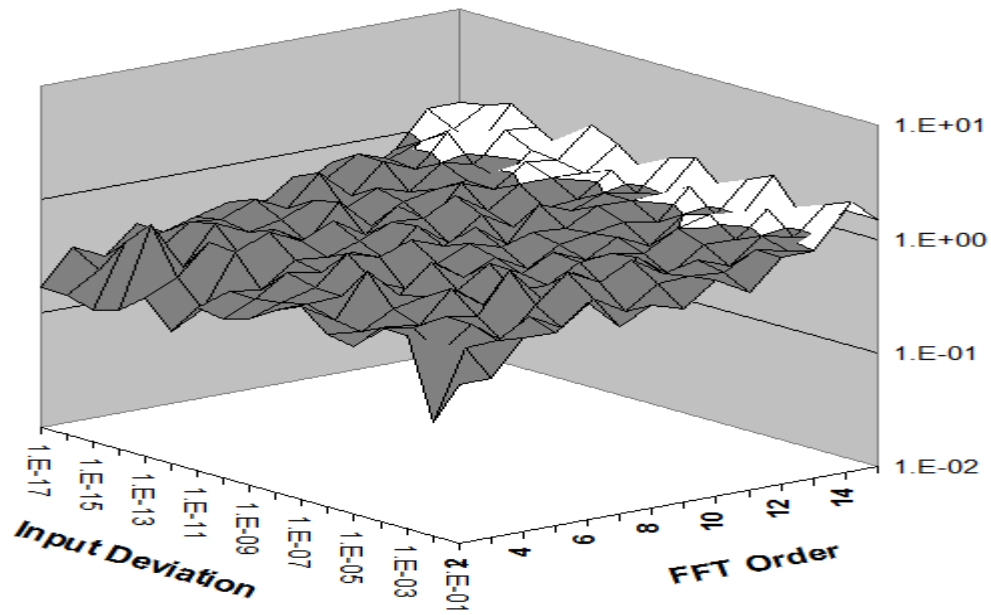


Figure 19: The maximal output bounding ratios using precision arithmetic increases slowly with FFT order.

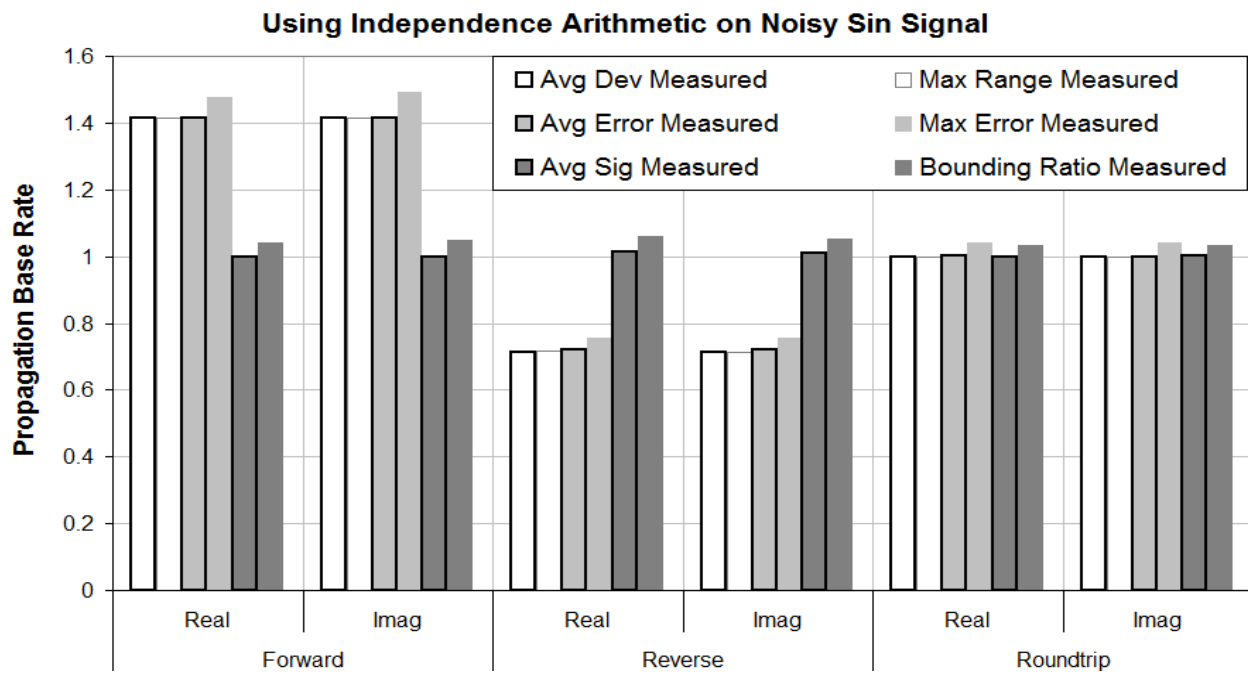


Figure 20: The characterization plot of independence arithmetic on noisy sine signal.

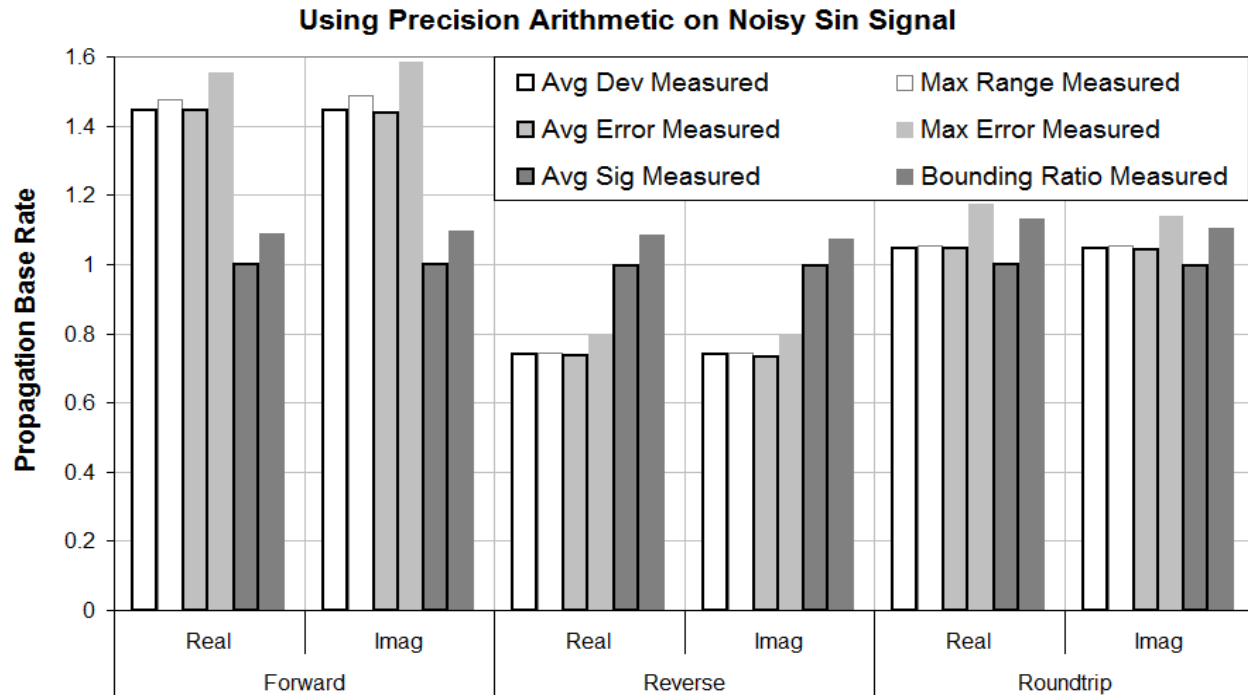


Figure 21: The characterization plot of precision arithmetic on noisy sine signal.

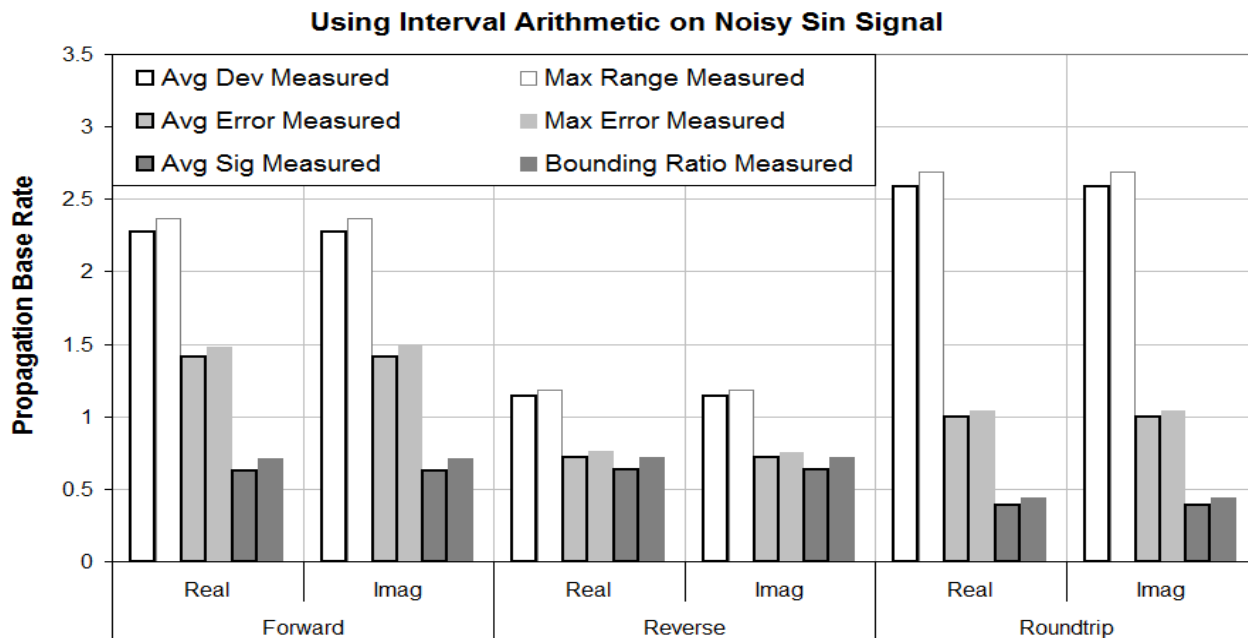


Figure 22: The characterization plot of interval arithmetic on noisy sine signal.

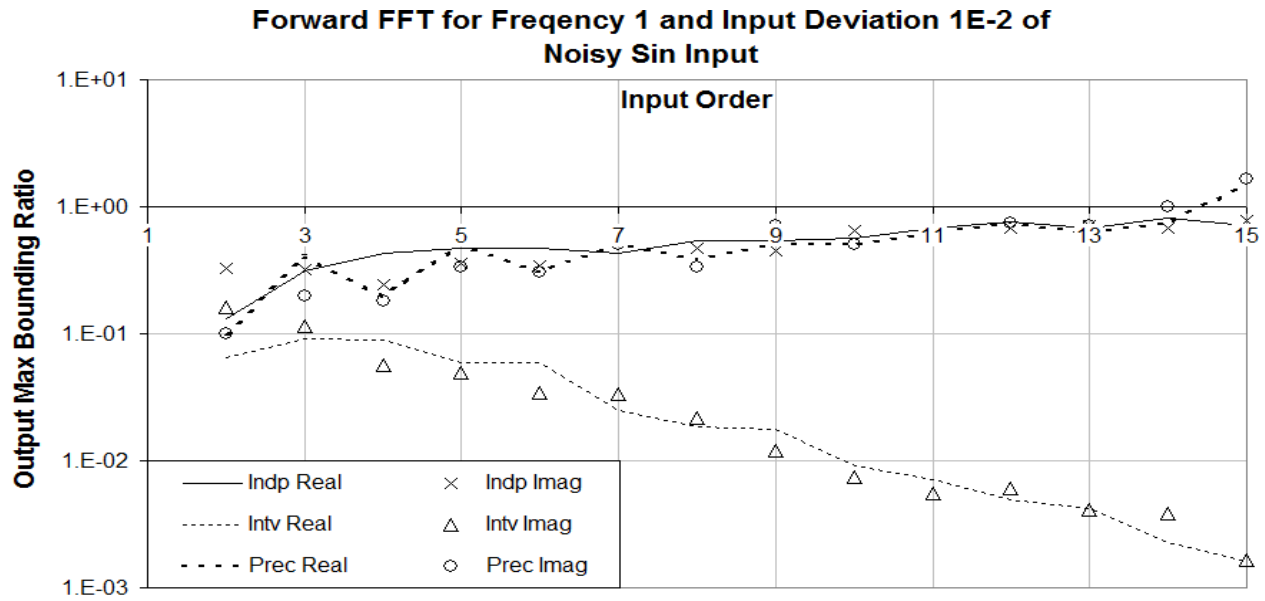


Figure 23: The output maximal bounding ratios vs. FFT order of forward FFT for all three arithmetics.

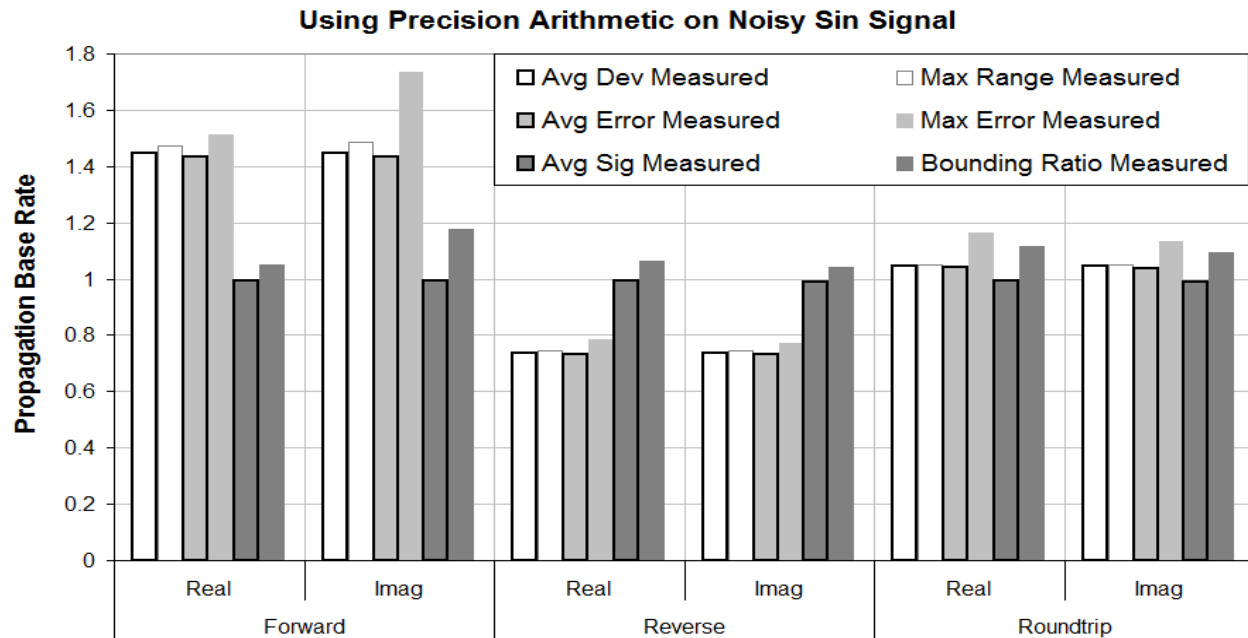


Figure 24: The characterization plot of precision arithmetic with a deterministic precision round up rule on noisy sine signal.

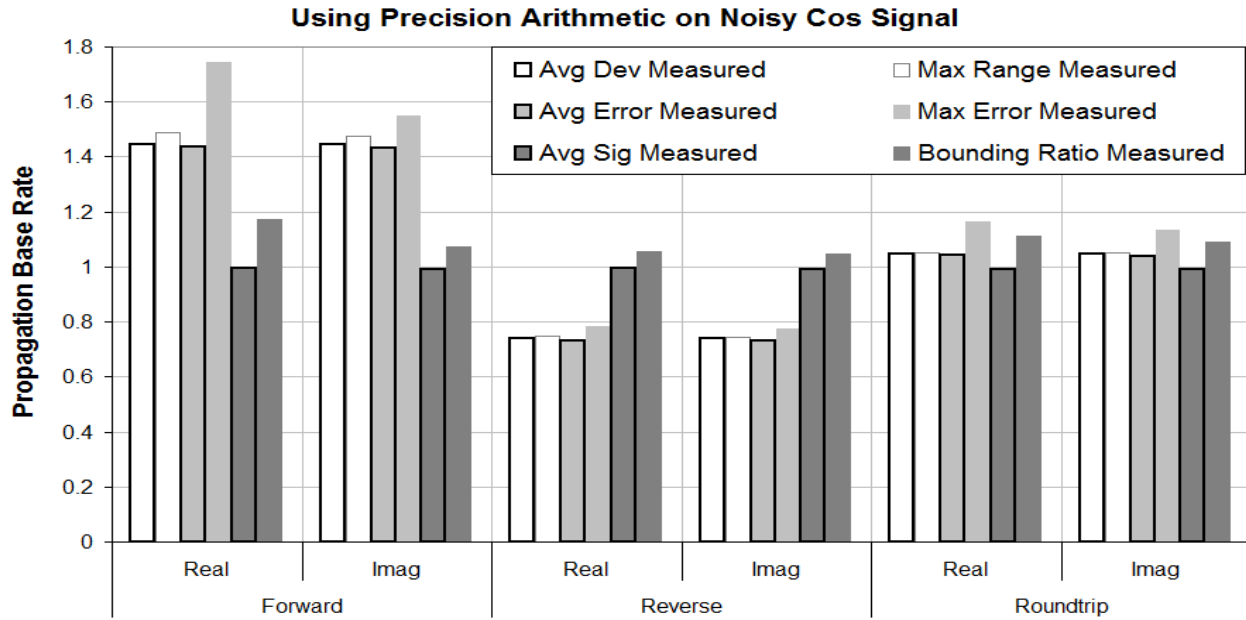


Figure 25: The characterization plot of precision arithmetic with a deterministic precision round up rule on noisy cosine signal.

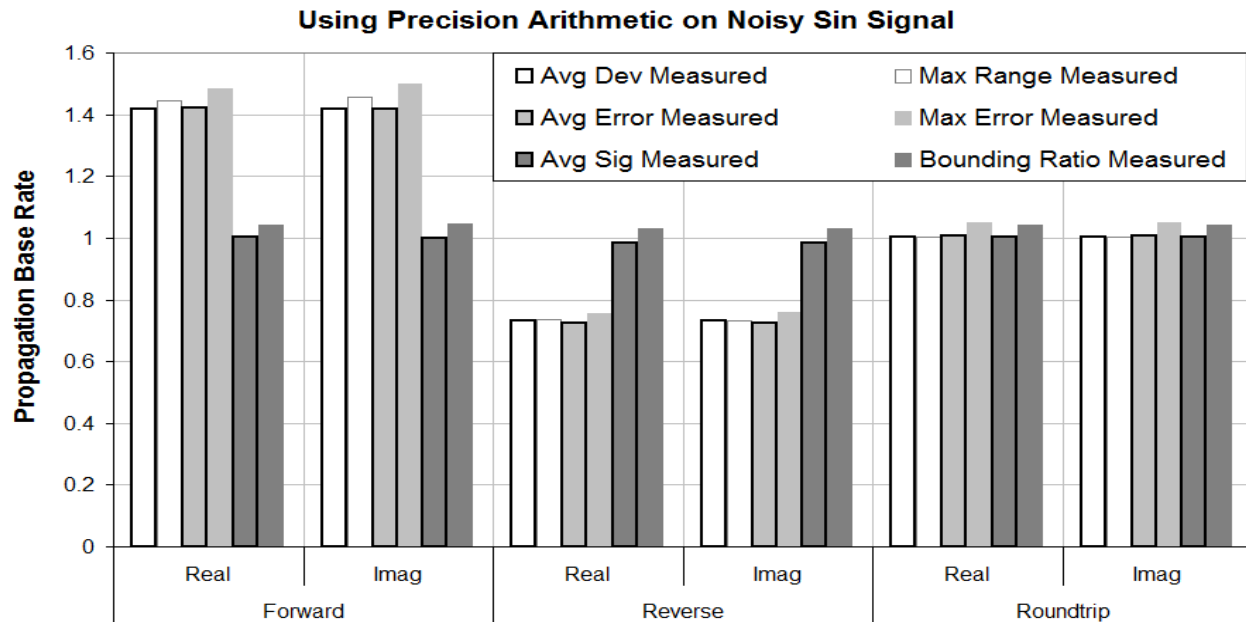


Figure 26: The characterization plot of precision arithmetic with two-bit calculation inside the uncertainty on noisy sine signal.

10 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:

- `RealTypes.lib`: It 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 to be 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.
- `Precision.exe`: It tests any floating-point arithmetic in `RealTypes.dll` specified by a short name on an input signal (among “Sin”, “Cos”, “SinCos”, “Linear”) with noise type

("" for none, "GN" for Gaussian noise) for all three FFT algorithms ("For"--Forward, "Rnd"--Round-trip and "Inv"--Reverse). It generates output files named as "<short name>_<algorithm>_<signal>_<noise type>.txt" in folder "../Output/<short name>". It outputs detailed FFT calculation in files named as "<short name>_<signal>_<FFT order>_<frequency>_<input deviation>_<noise type>.txt" in folder "../Output/<short name>/FFT", and measured error significand distribution in files named as "<short name>_<signal>_<input deviation>_<noise type>.txt" in folder "../Output/<short name>/Dist". It also demonstrates basic math using the floating-point arithmetic and outputs the result in file "../Output/<short name>/DemoMath.txt". It 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.

- `Parser.exe`: It 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.
- `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.

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 fit and plot the data in folder "../Doc/<short name>". For example, in "../Doc/Prec0":

- File "Prec Sin.xls", "Prec Cos.xls", "Prec SinCos.xls" and "Prec Linear.xls" displays FFT results for a particular input deviation and signal frequency. Figure 9 is taken directly from "Prec Sin.xls".
- File "Prec Dist 0.xls", "Prec Dist 1e-1.xls", "Prec Dist 1e-2.xls", "Prec Dist 1e-3.xls", "Prec Dist 1e-4.xls", "Prec Dist 1e-5.xls" and "Prec Dist Total.xls" displays error significant distribution for each bounding range, input deviation and FFT algorithm. Figure 10 is taken directly from "Prec Dist 1e-4.xls".
- File "For Charts.xls", "Rnd Charts.xls" and "Inv Charts.xls" compare results for Forward, Roundtrip and Reverse algorithms with the corresponding results of "Indp6" and "Intv6" respectively. Figure 11, Figure 12, Figure 18 and Figure 23 are taken directly from "For Charts.xls".
- File "Prec For.xls", "Prec Rnd.xls" and "Prec Inv.xls" contain fitting results for Forward, Roundtrip and Reverse FFT algorithms respectively. Figure 13, Figure 14 and Figure 19 are taken directly from file "Prec For.xls".
- File "Prec FFT.xls" compares the fitting results from "Prec For.xls", "Prec Rnd.xls" and "Prec Inv.xls". Figure 16 and Figure 20 are taken directly from file "Prec FFT.xls".

Thus, xls files under "../Doc/Prec0" gives a complete picture of the "Prec0" arithmetic. 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.

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.