A Comparison of Range Arithmetic Methods in Symbolic-Numeric Circuit Analysis Applications

Balavelan Thanigaivelan *Tara Julia Hamilton* *Adam Postula*

Abstract:

In this paper we compare the performance of several range arithmetic methods in evaluating problems in symbolic-numeric circuit analysis. Symbolic-numeric circuit analysis involves solving systems of linear equations in order to obtain the values of currents and voltages in a given circuit. Symbolic circuit expressions contain a large number of product terms and long chains of arithmetic operations which make computations difficult. When analysis of circuit behaviour over a range of parameter values is required the symbolic expressions can be evaluated using one of the range arithmetic methods reported in literature. The range arithmetic methods considered in this paper are affine arithmetic (AA), Extensions to AA (EAA) suggested by Messine, and generalized interval arithmetic (GIA) suggested by Kolev. One important difference between these range arithmetic methods is the method by which two affine numbers, or two generalized interval numbers, are multiplied. In this paper, each range arithmetic method is evaluated based on the performance of the multiplication operation which is predominant in symbolic circuit expressions. We show that the performance of the multiplication rule in each method determines the accuracy of computations. We present the benefits and problems associated with using each of the range arithmetic methods. We also show that GIA gives tighter bounds and efficiently simulates the effects of uncertainties in circuit parameters when solving problems in circuit analysis.

I. Introduction

Symbolic Analysis of analog integrated circuits refers to the study of circuit behaviour by deriving mathematical transfer functions in which some or all of the circuit parameters are maintained in symbolic form. A fully expanded generic symbolic transfer function in the Laplace domain can be represented as shown in (1).

$$G(s) = \frac{a_0 + a_1 \cdot s + a_2 \cdot s^2 + \ldots + a_n \cdot s^n}{b_0 + b_1 \cdot s + b_2 \cdot s^2 + \ldots + b_n \cdot s^n} \tag{1}$$

Where, '*s*' is the complex frequency variable. The coefficients of *s*-powers (such as $\{a_0, a_1 \ldots a_n\}$ and $\{b_0, b_1, \ldots b_n\}$) are all polynomial functions which can be represented as a sum-of-product terms, consisting of circuit parameters in either symbolic or numeric form. Unlike numerical analysis, symbolic analysis provides an opportunity to study the behaviour of a circuit when exact values of some or all of its parameters are not known before hand. Therefore, symbolic analysis is useful in extracting the dominant parameters affecting the circuit behaviour and assists with making better decisions on choices of numerical values for unknown circuit parameters. The circuit designer can also study the influence of a parameter on circuit behaviour by simply evaluating the expression for a range of values for that parameter. The numerical post processing of symbolic expressions for interpretation and graphical representation purposes usually involves real arithmetic.

In modern semiconductor technologies, circuit parameters also vary due to variations in technology parameters, supply voltage fluctuations and temperature changes. In order to analyse the circuit performance for variations in parameter values, the circuit expressions need to be numerically evaluated repeatedly with parameter values drawn from their respective ranges. However, the size of the symbolic expression increases dramatically with increases in circuit size. The computational overhead, due to large size expressions, makes repeated numerical evaluation using real arithmetic much more difficult. In order to efficiently post process the expressions with a range of values for each circuit parameter, range arithmetic methods can be used. A number of range arithmetic methods have been reported in the literature [1-6]. The sum-of-product terms represented by coefficients of *s*-powers in (1) need to be efficiently evaluated by choosing a suitable range arithmetic method.

Interval arithmetic (IA) [1] is the fundamental range arithmetic method designed to quickly perform numerical computations involving interval values and assumes each interval value in the numerical computation to be independent of the others. Affine arithmetic (AA) [5, 6] is a more complex range arithmetic method, with the ability to preserve and keep track of dependencies between the parameters whilst retaining simplicity in its representation. AA has proven to be more accurate, yielding results with tighter bounds than IA in many applications [5, 7, 8]. The Extensions to Affine Arithmetic (EAA) introduced by Messine [9] attempts to overcome the inefficiencies associated with the standard AA. The Generalized Interval Arithmetic, suggested by Kolev [2, 4], is another range arithmetic method with results represented in affine form. This method attempts to provide optimal results for affine arithmetic computations.

The arithmetic operations in the coefficients of *s*-powers in the symbolic transfer functions, such as shown in (1), are predominantly addition, subtraction, multiplication and exponentiation. Since the exponentiation operation is achieved by repeated multiplication, multiplication can be regarded as the predominant operation in the s-power coefficients. In this paper, different range arithmetic methods are compared based on their performance in multiplication operations. The different range arithmetic methods compared in the paper are standard Affine Arithmetic (AA) [6], Extensions to Affine Arithmetic (EAA) [9] and Generalized Interval Arithmetic (GIA) [2, 4]. We show that in a long chain of computations, such as the examples

considered in this paper, the multiplication operation determines the accuracy of the computed results. This is because the bounds on AA results can be wider than the IA results whenever the error terms compensating the result of a multiplication operation are overestimated during a long chain of computations. Although, the result of a multiplication operation, as suggested by Kolev, is optimal, the size of the results in affine form grows with the number of multiplication operations. This is because in the result of each multiplication operation a new error term is introduced. In this paper we adapt the optimal multiplication formulae to suit numerical evaluation of symbolic circuit expressions using GIA and show that the over estimation in the results are reduced.

This paper is organised as follows. Section II presents a background on symbolic circuit analysis and the range arithmetic methods. The GIA method for obtaining the maxima of symbolic network functions with parameter variations is presented in section III. This section also presents experimental results and discusses the benefits of using GIA. Section IV concludes the paper.

II. Background

I. Symbolic Circuit Analysis

Symbolic circuit analysis can be defined as the process of analysing circuit behaviour by deriving mathematical relationships between circuit parameters in symbolic form. The expressions are derived for the purpose of analysing the performance of parameters such as voltage gain, current gain, input impedance, output impedance, and so on. A fully expanded symbolic expression in its generic form is shown in (1). Depending on whether all or some of the circuit parameters are represented as symbols, the symbolic expressions can be classified into three types – fully symbolic, partially symbolic and rational functions in 's' with all other parameters as real numbers.

i. Fully symbolic expressions consist of all the parameters represented in symbolic form. For example consider the fully expanded exact expression in the Laplace domain shown in (2).

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{G_1 \cdot G_2}{\left(G_1 \cdot G_2 + \left(G_1 \cdot C_2 + G_2 \cdot C_1 + G_2 \cdot C_2\right) \cdot s + C_1 \cdot C_2 \cdot s^2\right)} \tag{2}$$

The expression, representing voltage gain, contains circuit parameters G_1, G_2, C_1 and C_2 that are preserved as symbols during its derivation. As can be seen in the denominator in (2), the coefficients of the *s*-power terms can be represented in the sum-of-product form.

ii. Partially symbolic expressions consist of some of the parameters retained as symbols and rest of them as numbers. For example, the conductance elements G_1, G_2 and capacitance C_1 in (2) are substituted with 1×10^{-4}, 2×10^{-4} and 2×10^{-9} respectively during derivation of the expression. The capacitance C_2 is retained as symbol and the partial symbolic expression is shown in (3).

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{\left(2\times10^{-4}\right)}{\left(2\times10^{-8} + \left(3\times10^{-3}\times C_2 + 2\times10^{-13}\right)\cdot s + \left(10^{-9}\times C_2\right)\cdot s^2\right)} \tag{3}$$

The expression shown in (3) can be repeatedly evaluated with a range of values for C_2 and its influence on the voltage gain can be analysed with numerical or graphical representations.

iii. Rational function in '*s*' consist of all the parameters except the complex frequency variable, '*s*', retained as numbers and is shown in (4).

$$\frac{V_{out}}{V_{in}} = 2\times10^{10}\,\frac{1}{\left(s^2 + \left(5\times10^5\right)\cdot s + 2\times10^{10}\right)} \tag{4}$$

Symbolic expressions can be obtained for various parameters of interest such as voltage gain, current gain, input impedance and output impedance, etc. A number of methodologies and techniques for symbolic analysis of analog circuits have been reported in the literature [10-16]. Symbolic circuit analysis methods can be broadly classified into two types, based on mathematical procedures, – topology or graph based methods – and matrix based methods [17]. In this paper we adopt a simple matrix based method that uses Cramer's rule to obtain symbolic expressions from a given circuit schematic. The symbolic circuit analysis procedure starts by formulating linear equations in matrix form with parameters drawn from the given circuit schematic. The linear equations are solved for desired unknown variables using Cramer's rule to obtain necessary symbolic expressions.

Despite several advanced symbolic analysis methods reported in the literature [11-16], the size of the symbolic expressions increase dramatically with increase in circuit size. This is a computational overhead when the expressions are numerically evaluated using range arithmetic. In order to generate compact symbolic expressions some useful strategies and models were suggested in [10, 18]. The results of symbolic analysis of different circuits, obtained using the strategies and

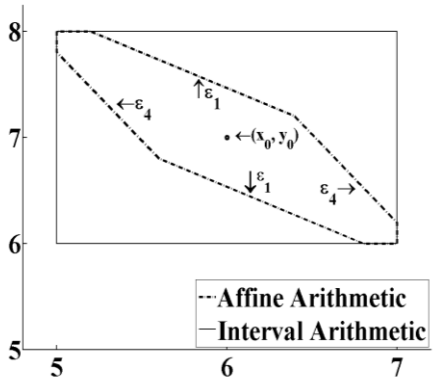

Fig 1. Joint range of 2 partially dependent quantities shown in equation (6)

TABLE I. SYMBOLIC EXPRESSION COMPARISON FOR CIRCUITS WITH LESS THAN 10 TRANSISTORS

Circuit	Product terms in coefficient of						N^a or D^b	No. of symbolic variables
	s^5	s^4	s^3	s^2	s^1	s^0		
Cascode Current Mirror #mosfet: 4	-	37	107	109	45	6	N	22
	-	113	288	239	70	5	D	
Differential Amplifier #mosfet: 6	-	128	393	428	189	26	N	31
	-	179	564	619	272	38	D	
Two Stage Opamp #mosfet: 8	240	849	1159	757	233	26	N	**42**
	1334	4269	5181	2966	790	76	D	

a. N = Numerator terms.
b. D = Denominator terms

models suggested in [10, 18] is shown in table I. The numbers in Table I represent the number of product terms in the coefficient of *s*-powers in the numerator and denominator of fully symbolic expressions. The last column shows the number of symbolic variables in the expressions. The number of product terms in the expression increases dramatically with the size of the circuit. Therefore in order to efficiently use range arithmetic to evaluate the symbolic expressions, a suitable framework and method has been suggested in [19]. The range arithmetic method used in [19] is presented in detail in this paper. The different range arithmetic methods compared in this paper are discussed in detail in the next sub-section.

II. Range Arithmetic Methods.

In this sub-section, different range arithmetic methods such as standard AA as suggested by Stolfi [6], EAA suggested by Messine [9] and GIA in Affine form suggested by Kolev [2, 4] and their application to evaluating symbolic expressions are discussed in detail. The methods are compared against each other using a quantitative measure for the overestimation in their results, called relative accuracy. The relative accuracy is defined as the ratio between the size of the actual range and that of the computed range [6]. The actual range is the smallest interval that contains only the possible values of result of the evaluated expression, when the value of each variable in the expression is set vary over its given range. If the width of the computed range is w_c and the width of the actual range is w_c, then the relative accuracy is given by (5).

$$\text{Relative Accuracy} = \frac{w_a}{w_c} \tag{5}$$

According to (5), the relative accuracy is 1 when the computed width is the same as the actual width. The relative accuracy is zero, when the computed result is infinitely wide. The width of the actual range is not always a known quantity, especially when a number of variables dependent on each other are involved in the computation.. Therefore, in this paper, the relative accuracy measure is computed using IA results as a basis. The ratio of width of IA result to the width of the AA result represents the relative accuracy of AA when compared against IA. In this case, the relative accuracy is greater than 1, whenever AA is more accurate than IA and less than 1 otherwise. The IA results are used as the base, because IA is the fundamental range arithmetic method and assumes that all the variables in the expression are independent of each other. The other methods were introduced in order to use the dependency between the variables to achieve less conservative results. The following sub-sections present the range arithmetic methods in detail.

a) Standard Affine Arithmetic (AA)

Affine arithmetic was first introduced by Stolfi et al in the early 90s. The standard affine number as defined in [6] is given in (6).

$$\tilde{x} = x_0 + \varepsilon_1 x_1 + \varepsilon_2 x_2 \cdots + \varepsilon_n x_n = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \tag{6}$$

Where x_0 and x_i are known real numbers and ε_i is a symbolic variable whose value lies in the interval [-1…+1]. The value x_0 is the central value of $\tilde{x}$ while each x_i multiplied with its corresponding noise symbol ε_i quantifies an uncertainty which when appropriately added to or subtracted from x_0 yields $\tilde{x}$. Two affine numbers are said to be dependent on each other when they share common noise symbols. For example, consider the affine numbers in (7). The numbers $\tilde{x}$ and $\tilde{y}$ share common noise symbols ε_1 and ε_4. So they are said to be dependent on each other or correlated with each other. The principle of AA is to keep track of the dependencies between operands during computation of the bounds of the final result.

$$\tilde{x} = 6 + 0.6\varepsilon_1 + 0.1\varepsilon_2 + 0.3\varepsilon_4; \quad \tilde{y} = 7 - 0.4\varepsilon_1 + 0.1\varepsilon_3 - 0.5\varepsilon_4 \tag{7}$$

Sometimes effects of common noise variables (such as ε_1 and ε_4) cancel out if their coefficients are the same. The common noise symbols also determine the size of the joint range of the two numbers, which is a convex polygon symmetric around the central point x_0 and y_0, as shown by the dotted line in Fig 1. The value of the coefficients of the dependencies ε_1 and ε_4 determine the lengths of two edges of the polygon as indicated in the figure. In this example, coefficients of ε_1 and ε_4 are significantly larger than the coefficients of other noise terms such as ε_2 and ε_3, resulting in a polygon of smaller size. This dependency information and its benefits are completely lost when the two affine numbers are converted to interval form. The joint range of two intervals is a rectangle as shown by the solid line in Fig 1. Therefore, by preserving the dependency between parameters, all those combinations of values in the given interval that do not yield a valid result are eliminated. The tighter bounds in the results are achieved at the cost of preserving and keeping track of dependencies in a chain of computation.

The AA values can be converted to their interval form and vice versa. Consider the affine number in (6). This can be converted to interval form using equation (8). Similarly, an interval value $\bar{x}$ given by $[a \ldots b]$, can be converted to affine form using (9).

$$\bar{x} = [x_0 - rad(\tilde{x}),\ \ x_0 + rad(\tilde{x})]; \quad \text{where} \quad rad(\tilde{x}) = \sum_{i=1}^{n} |x_i| \tag{8}$$

$$\tilde{x} = x_0 + x_1\varepsilon_1; \quad \text{where } x_0 = \frac{a+b}{2};\ x_1 = \frac{b-a}{2};\ \varepsilon_1 = [-1 \ldots +1] \tag{9}$$

The arithmetic operations in affine analysis can be classified, depending on the result of the operation, into two types – affine operations and non-affine operations. The affine operations yield results that can be represented in the affine form as shown in (6). Some examples of affine operations are shown in the following equations. The addition or subtraction between two affine numbers is shown in (10). The multiplication by a scalar with an affine number is shown in (11) and addition or subtraction of a scalar to an affine number is shown in (12). As can be seen from these equations the results of the operations are in the affine form.

$$\tilde{x} \pm \tilde{y} = (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n \tag{10}$$

$$\alpha \cdot \tilde{x} = (\alpha \cdot x_0) + (\alpha \cdot x_1)\varepsilon_1 + \cdots + (\alpha \cdot x_n)\varepsilon_n \tag{11}$$

$$\tilde{x} \pm \zeta = (x_0 \pm \zeta) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \tag{12}$$

The non-affine operations yield results that cannot be expressed in the affine form shown in (6). Operations between affine numbers, such as multiplication, division, square and square root, are examples of non-affine operations. The results of these operations are approximated in order to express them in affine form. The most commonly used and hence most important operation is multiplication. The multiplication rule in its exact form is shown in (13).

$$\tilde{x} \times \tilde{y} = \left(x_0 + \sum_{i=1}^{n} x_i\varepsilon_i\right) \times \left(y_0 + \sum_{i=1}^{n} y_i\varepsilon_i\right) = x_0y_0 + \sum_{i=1}^{n} (x_0y_i + y_0x_i) + \left(\sum_{i=1}^{n} x_i\varepsilon_i\right)\left(\sum_{i=1}^{n} y_i\varepsilon_i\right) \tag{13}$$

The above equation cannot be represented in affine form as in (6) and is approximated with an affine form shown in (14).

$$\begin{aligned} &\tilde{z}_{exact} = \tilde{x} \times \tilde{y} \approx \tilde{z}_{approx} + z_{n+1}\varepsilon_{n+1} \\ &\text{where} \quad \tilde{z}_{approx} = x_0y_0 + \sum_{i=1}^{n} (x_0y_i + y_0x_i) \end{aligned} \tag{14}$$

The term $z_{n+1} \cdot \varepsilon_{n+1}$ is added to $\tilde{z}_{approx}$ to model the error or the difference between $\tilde{z}_{exact}$ and $\tilde{z}_{approx}$. However, in standard AA, $\cdot\varepsilon_{n+1}$ is an additional noise symbol added to the result after each non-affine multiplication operation. Therefore, the number of noise symbols in the result grows by one additional symbol for each non-affine multiplication operation. In a long chain of computation this can lead to the generation of up to *m* distinct additional terms for *m* non-affine operations. This poses an overhead on the number of symbolic variables ε_i that need to be handled during numerical evaluation of symbolic expressions. A simple conservative estimate for z_{n+1} suggested in [6] is shown in (15).

$$z_{n+1} = \sum_{i=1}^{n} |x_i| \times \sum_{i=1}^{n} |y_i| \tag{15}$$

The product of the affine numbers in (7) obtained using equations (14) and (15) is shown in (16). The affine product converted to interval form (bottom), along with the result obtained using IA (top) is also shown in (16) for the purpose of comparison.

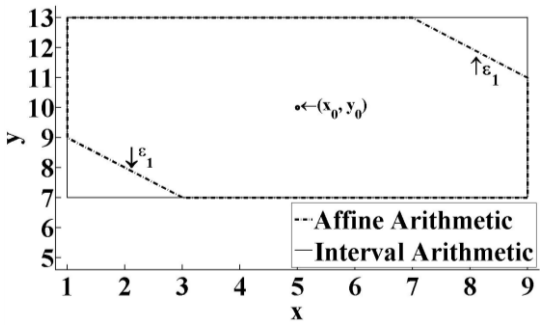

Fig 2. Joint range of 2 partially dependent quantities shown in equation (17)

$$\begin{aligned} \bar{x} \times \bar{y} &= [5 \ldots 7] \times [6 \ldots 8] = [30 \ldots 56] \\ \tilde{x} \times \tilde{y} &= 42 + 1.8\varepsilon_1 + 0.7\varepsilon_2 + 0.6\varepsilon_3 - 0.9\varepsilon_4 + \varepsilon_5 = [37 \ldots 47] \end{aligned} \tag{16}$$

The results in (16), compared using the relative accuracy measure ((56 – 30) / (47 – 37)) gives 2.6. In this case, IA result is 2.6 times wider than the AA result. The AA yields tighter bounds because it preserves the dependencies between the two numbers. The result supports the inference obtained from the joint range plot shown in Fig 1. In the case of IA, all the dependency information is lost and therefore bounds are wider than necessary.

Although preserving the dependencies between parameters help achieve tighter bounds, the error estimate for z_{n+1} shown in (15) is conservative. Therefore, the AA results sometimes can be overestimated. Consider the multiplication operation using the example shown in (17). The joint range plot of the two variables is shown in Fig 2. According to this plot, some of the values for the pair (x, y) are eliminated because of the dependency due to ε_1. The AA result is expected to be narrower than the IA result. However, it is not only overestimated than the IA result, but also contains a negative lower bound. This is not consistent with the joint range plot shown in Fig 2. The intervals of both $\bar{x}$ and $\bar{y}$ contain only positive values, whereas the AA result contains negative values as well due to overestimation in z_{n+1}

$$\begin{aligned} &\tilde{x} = 5 + \varepsilon_1 - 3\varepsilon_3; \quad \tilde{y} = 10 - \varepsilon_1 + 2\varepsilon_2 \\ &\bar{x} \times \bar{y} = [1 \ldots 9] \times [7 \ldots 13] = [7 \ldots 117] \\ &\tilde{x} \times \tilde{y} = 50 + 5\varepsilon_1 + 10\varepsilon_2 - 30\varepsilon_3 + 12\varepsilon_4 = [-7 \ldots 107] \\ &z_{approx} = 50 + 5\varepsilon_1 + 10\varepsilon_2 - 30\varepsilon_3 = [5 \ldots 95] \\ &\text{Relative Accuracy} = 110 / 114 = 0.9649 \text{ (less than 1)} \end{aligned} \tag{17}$$

The interval bounds of the affine arithmetic results can be wider than necessary, even when the numbers are independent. In the case of two independent affine numbers, the interval bounds of the AA results can be expected to be the same as that of IA. Consider another multiplication example shown in (18). The AA result in (18) is also wider than the IA result. The overestimation can be attributed to both z_{approx} and conservative error estimate ε_3 because the width of z_{approx} alone is equal to the width of the IA result as shown in (18).

$$\begin{aligned} &\tilde{x} = 4.5 + 0.5\varepsilon_1; \quad \tilde{y} = 10 + 2\varepsilon_2 \\ &\bar{x} \times \bar{y} = [4 \ldots 5] \times [8 \ldots 12] = [32 \ldots 60] \text{ (width = 28)} \\ &\tilde{x} \times \tilde{y} = 45 + 5\varepsilon_1 + 9\varepsilon_2 + \varepsilon_3 = [30 \ldots 60] \\ &z_{approx} = 45 + 5\varepsilon_1 + 9\varepsilon_2 = [31 \ldots 59] \text{ (width = 28)} \\ &\text{Relative Accuracy} = 28 / 30 = 0.9333 \text{ (less than 1)} \end{aligned} \tag{18}$$

In addition, the affine arithmetic results can be overestimated during the exponentiation operation. Consider the affine number shown in (19).

$$\begin{aligned} &\tilde{x} = 0 + 3 \cdot \varepsilon_1 \\ &\tilde{x}^2 = \tilde{x} \times \tilde{x} = 0 + 0 \cdot \varepsilon_1 + 9 \cdot \varepsilon_2 = [-9 \ldots 9] \\ &\bar{x}^2 = [-3 \ldots 3] \times [-3 \ldots 3] = [-9 \ldots 9] \end{aligned} \tag{19}$$

The result of a square operation is expected to be an interval with no negative values. Instead, the affine arithmetic computation overestimates the result with negative numbers included in the result.

In summary, the size of the affine form of the result of multiplication operation grows by an extra term in the case of AA. The conservative error estimate used in AA can lead to overestimation in the multiplication of both independent and dependent affine numbers. The AA result is overestimated in case of exponentiation operation. Therefore, the standard AA is

not suitable for evaluating the symbolic expressions involving long computation chains due to overestimation in the results. The next section suggests other forms of affine arithmetic as suggested by Messine [9].

b) Extensions to Affine Arithmetic (EAA)

The Extensions to Affine Arithmetic (EAA) [9] were introduced by Messine in order to address the issues and difficulties with standard AA. The EAA consists of two new representations for the affine number shown in (6). The Affine Form 1 (AF1) attempts to contain the growth of the affine form by introducing a common error noise symbol for all the multiplication operation in a chain. The affine operations are the same as AA, except the multiplication rule has a different error estimate. Although, the AF1 overcomes the issues with growing noise symbols, it also suffers from overestimation for the examples described in the previous sub-section. In this sub-section, the second affine form, AF2, is discussed in detail. The AF2 is an extension of AF1 as it consists of two more new noise symbols added to the affine number, defined in (20).

$$\tilde{x} = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i + x_{n+1}\varepsilon_{n+1} + x_{n+2}\varepsilon_{n+2} + x_{n+3}\varepsilon_{n+3} \tag{20}$$

The partial deviation ε_{n+1} is introduced to represent the error from a non-affine operation similar to the AF1 or AA. The two other terms ε_{n+2} and ε_{n+3} are introduced to overcome the issues with overestimation in AF1 [9]. The values of new partial deviation terms ε_{n+2} and ε_{n+3} lie in the intervals [0…+1] and [-1…0] respectively. The values of x_{n+1}, x_{n+2} and x_{n+3} are always retained as positive numbers. The basic affine operations are shown in (21) – (23).

$$\tilde{x} \pm \tilde{y} = (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\cdot\varepsilon_i + (x_{n+1} + y_{n+1})\cdot\varepsilon_{n+1} + (x_{n+2} + y_{n+2})\cdot\varepsilon_{n+2} + (x_{n+3} + y_{n+3})\cdot\varepsilon_{n+3} \tag{21}$$

$$\alpha\cdot\tilde{x} = (\alpha\cdot x_0) + (\alpha\cdot x_1)\varepsilon_1 + \cdots + (\alpha\cdot x_n)\varepsilon_n + (|\alpha|\cdot x_{n+1})\varepsilon_{n+1} + (|\alpha|\cdot x_{n+2})\varepsilon_{n+2} + (|\alpha|\cdot x_{n+3})\varepsilon_{n+3} \tag{22}$$

$$\zeta \pm \tilde{x} = (\zeta \pm x_0) \pm x_1\varepsilon_1 \pm \cdots \pm x_n\varepsilon_n + x_{n+1}\varepsilon_{n+1} + x_{n+2}\varepsilon_{n+2} + x_{n+3}\varepsilon_{n+3} \tag{23}$$

The values of x_{n+1}, x_{n+2} and x_{n+3} are determined during a non affine operation. The non-affine operations are more involved than the other affine arithmetic rules presented before. The multiplication operation is shown in (24).

$$\tilde{x}\times\tilde{y} = x_0y_0 + \sum_{i=1}^{n}(x_0y_i + y_0x_i)\cdot\varepsilon_i + K_1\varepsilon_{n+1} + K_2\varepsilon_{n+2} + K_3\varepsilon_{n+3} \tag{24}$$

The values for K_1, K_2 and K_3 are carefully computed depending on certain conditions. These values are initialized according to (25). The final values are computed according to the equations in (26).

$$\begin{aligned}
K_1^0 &= |x_0|\,y_{n+1} + |y_0|\,x_{n+1} \\
K_2^0 &= \begin{cases} x_0y_{n+2} + y_0x_{n+2} & \text{if } x_0 > 0 \text{ and } y_0 > 0 \\ x_0y_{n+2} - y_0x_{n+3} & \text{if } x_0 > 0 \text{ and } y_0 < 0 \\ -x_0y_{n+3} + y_0x_{n+2} & \text{if } x_0 < 0 \text{ and } y_0 > 0 \\ -x_0y_{n+3} - y_0x_{n+3} & \text{if } x_0 < 0 \text{ and } y_0 < 0 \end{cases} \\
K_3^0 &= \begin{cases} x_0y_{n+3} + y_0x_{n+3} & \text{if } x_0 > 0 \text{ and } y_0 > 0 \\ x_0y_{n+3} - y_0x_{n+2} & \text{if } x_0 > 0 \text{ and } y_0 < 0 \\ -x_0y_{n+3} + y_0x_{n+2} & \text{if } x_0 < 0 \text{ and } y_0 > 0 \\ -x_0y_{n+2} - y_0x_{n+2} & \text{if } x_0 < 0 \text{ and } y_0 < 0 \end{cases}
\end{aligned} \tag{25}$$

$$K_1 = K_1^0 + \sum_{i=1}^{n+3}\sum_{j=1,j\neq i}^{n+3}|x_iy_j|;\;\; K_2 = K_2^0 + \sum_{i=1,x_iy_i>0}^{n+3} x_iy_i;\;\; K_3 = K_3^0 + \sum_{i=1,x_iy_i<0}^{n+3}|x_iy_i| \tag{26}$$

The conversions from AF2 to interval form and vice versa are shown in (27) and (28).

$$\bar{x} = [x_0 - rad\_l(\tilde{x}),\;\; x_0 + rad\_h(\tilde{x})];\quad \text{where}\quad rad\_l(\tilde{x}) = \sum_{i=1,i\neq n+2}^{n+3}|x_i|, rad\_h(\tilde{x}) = \sum_{i=1}^{n+2}|x_i| \tag{27}$$

$$\tilde{x} = x_0 + x_1\varepsilon_1;\quad \text{where } x_0 = \frac{a+b}{2};\;\; x_1 = \frac{b-a}{2};\;\; \varepsilon_1 = [-1\ldots+1];\; x_2 = x_3 = 0 \tag{28}$$

The difference between the multiplication rules in standard AA and EAA is the error term in a non-affine operation. The error terms in AF2 help achieve tighter interval bounds in the result of non-affine operation, such as multiplication. For example the result of exponentiation operation for the example shown in (19) obtained using the rules of EAA is shown in (29).

$$\tilde{x} = 0 + 3 \cdot \varepsilon_1$$
$$\tilde{x}^2 = \tilde{x} \times \tilde{x} = 0 + 0 \cdot \varepsilon_1 + 0 \cdot \varepsilon_2 + 9 \cdot \varepsilon_3 + 0 \cdot \varepsilon_4 = [0 \ldots 9] \tag{29}$$

Despite achieving tighter bounds than standard AA and other advantages as mentioned in [9], the EAA also suffers from overestimation in the results. Consider the examples in (17) and (18), the results of EAA are shown in (30) and (31).

$$\tilde{x} = 5 + \varepsilon_1 - 3\varepsilon_3; \quad \tilde{y} = 10 - \varepsilon_1 + 2\varepsilon_2$$
$$\tilde{x} \times \tilde{y} = 50 + 5\varepsilon_1 + 10\varepsilon_2 - 30\varepsilon_3 + 11\varepsilon_4 + \varepsilon_6 = [-7 \ldots 107] \tag{30}$$
$$\text{Relative Accuracy} = 110/114 = 0.9649 \text{ (less than 1)}$$

$$\tilde{x} = 4.5 + 0.5\varepsilon_1; \quad \tilde{y} = 10 + 2\varepsilon_2$$
$$\tilde{x} \times \tilde{y} = 45 + 5\varepsilon_1 + 9\varepsilon_2 + \varepsilon_3 = [30 \ldots 60] \tag{31}$$
$$\text{Relative Accuracy} = 28/30 = 0.9333 \text{ (less than 1)}$$

Moreover, the *n+1*, *n+2* and *n+3* terms in the result are always positive and do not always compensate for the error in the non-affine operation appropriately. In the next sub-section an optimal range arithmetic method that has no overestimation in the multiplication results given by Kolev in [2, 4] is described in detail.

c) Kolev's Generalized Interval Arithmetic (GIA)

The Generalized Interval Arithmetic was first presented as a modification to standard AA in [20]. The GIA number is defined in affine form as shown in (6). The affine operations are the same as those shown in (10) – (12). In order to achieve tighter bounds in non-affine operations two new multiplication rules were suggested in [2, 4]. The two rules are defined separately each for dependent and independent affine numbers.

Whenever the product of two affine numbers $\tilde{x}$ and $\tilde{y}$ is to be computed, the following general cases arise.

$$\tilde{x} \geq 0, \tilde{y} \geq 0; \quad \tilde{x} \leq 0, \tilde{y} \leq 0; \quad \tilde{x} \geq 0, \tilde{y} \leq 0; \quad \tilde{x} \leq 0, \tilde{y} \geq 0 \tag{32}$$

An affine number $\tilde{x}$ is said to be positive ($\tilde{x} > 0$), if the corresponding interval contains all positive numbers. Similarly, an affine number is negative when the corresponding interval contains only negative numbers. The multiplication rules in [2, 4] are defined only for the first case and therefore, all the other cases in (32) are transformed to the first case similar to that shown in (34).The optimal multiplication rules suggested in [2, 4] can be used only when one of the above four conditions are satisfied. A fifth general case arises when one of the intervals ($\bar{x}$ or $\bar{y}$) contains a zero properly, that is when the lower bound (x_l) and the upper bound (x_h) are of different signs. In this case, the multiplication rule suggested by AA or EAA can be used. The last three of the four cases shown in (32) can be transformed to the first case during a multiplication operation. For example, the fourth case shows that *x* is negative and y is positive and their multiplication is shown in (33). This can be transformed to first case as shown in (34).

$$\tilde{z} = \tilde{x} \times \tilde{y} \text{ where } \tilde{x} \geq 0, \tilde{y} \leq 0 \tag{33}$$

$$\tilde{z} = -\tilde{z}' = -\tilde{x} \times \tilde{y}' \text{ where } \tilde{x} \geq 0, \tilde{y}' \geq 0 \text{ and } \tilde{y}' = -\tilde{y} \tag{34}$$

The optimal multiplication formula for independent affine numbers as presented in [4] is summarized as follows. Consider two independent affine numbers and their interval form shown in (35),

$$\tilde{x} = x_0 + \sum_{i=1}^{m} x_i \varepsilon_i; \quad \tilde{y} = y_0 + \sum_{i=m+1}^{n} y_i \varepsilon_i$$
$$\bar{x} = [x_l \ldots x_h]; \; \bar{y} = [y_l \ldots y_h] \tag{35}$$

The product of the two affine numbers in (35) is shown in (36)

$$\tilde{z} = \tilde{x} \times \tilde{y} = z_0 + \sum_{i=1}^{n+1} z_i \varepsilon_i; \text{ where } z_0 = -y_l x_l + y_l x_0 + x_l y_0 + b_0$$
$$z_i = y_l x_i \text{ for } i = 1, \ldots, m$$
$$z_i = x_l y_i \text{ for } i = m+1, \ldots, n \tag{36}$$
$$z_{n+1} = b_1$$

Here b_0 and b_1 are the centre and radius of error term *b*, whose interval bounds are computed using equation (37).

$$b_l = 0; b_h = x_l y_l + x_h y_h - x_l y_h - x_h y_l \tag{37}$$

The above optimal multiplication formula for independent affine numbers and its proof is presented in detail in [4]. The result for the optimal multiplication of independent affine numbers in (18) is shown in (38).

$$\begin{aligned} &\tilde{x} = 4.5 + 0.5\varepsilon_1; \quad \tilde{y} = 10 + 2\varepsilon_2 \\ &z_{approx} = 46 + 4\varepsilon_1 + 8\varepsilon_2 = [34 \ldots 58] \text{ (width = 24)} \\ &\tilde{x} \times \tilde{y} = 46 + 4\varepsilon_1 + 8\varepsilon_2 + 2\varepsilon_3 = [32 \ldots 60] \\ &\text{Relative Accuracy} = 28/28 = 1 \end{aligned} \tag{38}$$

The relative accuracy is 1 as opposed to 0.9333 in the case of standard AA and EAA. It can be seen that the width of z_{approx} is less than the final interval result and the error term compensates for it. Therefore the overestimation in multiplication of independent affine numbers is avoided (refer to [4] for proof). Since the numbers are independent, the relative accuracy is equal to 1.

In the case of affine numbers sharing common noise symbols, the optimal multiplication formula presented in [2] is used. The computation of the optimal multiplication result of dependent affine numbers requires that the first condition in (32) is satisfied. The important part of the multiplication formula involves computation of an interval Z^*, which is an optimal result of multiplication (in interval form $[z_l^* \ldots z_h^*]$) of two dependent affine numbers. The conditions and formula for computing Z^* is more involved and its presentation is therefore not covered here. The optimal multiplication formula for dependent affine numbers is described as follows.

Consider the two dependent affine numbers $\tilde{x}$ and $\tilde{y}$ and the corresponding interval numbers $\tilde{x}$ and $\tilde{y}$ shown in (39).

$$\begin{aligned} &\tilde{x} = x_0 + \sum_{i=1}^{m} x_i \varepsilon_i; \ \ \tilde{y} = y_0 + \sum_{i=1}^{m} y_i \varepsilon_i \\ &\overline{x} = [x_l \ldots x_h]; \ \overline{y} = [y_l \ldots y_h] \end{aligned} \tag{39}$$

Let M_x denote a set of indices of those noise symbols that have a non-zero coefficient in $\tilde{x}$ and zero coefficient in $\tilde{y}$. Similarly, let M_y denote the set of indices of independent noise symbols with non-zero coefficients in $\tilde{y}$ and M_c denote the indices of those noise symbols shared by $\tilde{x}$ and $\tilde{y}$. The product of the affine numbers in (39) is shown in (40). The optimal result Z^* in interval form, $[z_l^* \ldots z_h^*]$, is computed using the method illustrated in [2] and values are used in (40).

$$\begin{aligned} &\tilde{z} = \tilde{x} \times \tilde{y} = z_0 + \sum_{i=1}^{n+1} z_i \varepsilon_i; \text{ where } z_0 = \frac{z_h^* + z_l^*}{2} \\ &z_i = y_l x_i \text{ for } i \in M_x; \qquad z_i = x_l y_i \text{ for } i \in M_y; \\ &z_i = x_l y_i + y_l x_i \text{ for } i \in M_c; \\ &z_{n+1} = \frac{z_h^* - z_l^*}{2} - R, \quad R = \sum_{i=1}^{n} |z_i| \end{aligned} \tag{40}$$

The result for multiplication of dependent affine numbers in (17) is shown in (41).

$$\begin{aligned} &\tilde{x} = 5 + \varepsilon_1 - 3\varepsilon_3; \quad \tilde{y} = 10 - \varepsilon_1 + 2\varepsilon_2 \\ &\overline{x} \times \overline{y} = [1 \ldots 9] \times [7 \ldots 13] = [7 \ldots 117] \\ &\tilde{x} \times \tilde{y} = 54 + 6\varepsilon_1 + 2\varepsilon_2 - 21\varepsilon_3 + 16\varepsilon_4 = [9 \ldots 99] \\ &\text{Relative Accuracy} = 110/90 = 1.222 \text{ (greater than 1!)} \end{aligned} \tag{41}$$

The relative accuracy in (41) is greater than 1, showing tighter bounds in the affine arithmetic result than the interval arithmetic result by a small margin. This result is consistent with the joint range of the two numbers shown in Fig 2. It can be expected that in a long chain of computation, the use of Kolev's GIA will yield tighter bounds than interval arithmetic results. The implementation of the optimal multiplication formulae in (36) and (40) is not computationally expensive and only involves checking the conditions as mentioned in [2, 4] and computing the result.

Although optimal results can be obtained, the GIA always generates an extra term during multiplication. In a long chain of computation, such as in the evaluation of symbolic circuit expressions, the number of error terms grows dramatically. Therefore, the optimal multiplication rules suggested in [2, 4] need to be modified such that the error is always stored in a single error term when used to evaluate the symbolic expressions. The modified multiplication rules and experimental results are presented in the following section.

III. Symbolic-Numeric Circuit Analysis using GIA

I. GIA in long computation chain

The size of the multiplication result in affine form grows by one extra term for every multiplication operation in a chain of computations. For example, consider an arbitrary sum-of-product computation with three product terms. Let the number of multiplication operations in each product term be two. Therefore for each product term in the sum-of-product, the result will have two extra noise terms. Since the noise terms are treated as distinct from each other, the sum of the three product terms will have six noise terms in total. In a symbolic-numeric analysis, the circuit expressions have a large number of product terms shown in Table I. Therefore, it is essential to restrict the size of the result of final computation in affine form. The GIA method has to be modified for this purpose, so that the symbolic expressions can be numerically evaluated with ease.

The multiplication rules in (36) and (40) are modified as follows. Let $\tilde{x}$ and $\tilde{y}$ be two independent affine numbers as shown in (42) along with their interval form.

$$\begin{aligned} &\tilde{x} = x_0 + \sum_{i=1}^{m} x_i \varepsilon_i + x_{err}\varepsilon_{err};\ \ \tilde{y} = y_0 + \sum_{i=m+1}^{n} y_i \varepsilon_i + y_{err}\varepsilon_{err} \\ &\overline{x} = \left[x_l \ldots x_h\right];\ \overline{y} = \left[y_l \ldots y_h\right] \end{aligned} \tag{42}$$

According to the multiplication rule in (36) the central value and the noise terms are shown in (43).

$$\begin{aligned} &\tilde{z} = \tilde{x} \times \tilde{y} = z_0 + \sum_{i=1}^{n} z_i \varepsilon_i + z_{err}\varepsilon_{err};\ \text{where } z_0 = -y_l x_l + y_l x_0 + x_l y_0 + b_0 \\ &z_i = y_l x_i \ \text{ for } i = 1,\ldots,m \\ &z_i = x_l y_i \ \text{ for } i = m+1,\ldots,n \\ &z_{n+1} = x_l y_{err};\ \ z_{n+2} = y_l x_{err} \\ &z_{err} = b_1 \end{aligned} \tag{43}$$

The terms z_{n+1} and z_{n+2} shown in (43) are extra terms that increase the size of the multiplication result. These two extra terms need to be accumulated and absorbed into the error term z_{err}. The absolute values of these two terms contribute to the radius of $\tilde{z}$ as shown in (44).

$$rad\left(\tilde{z}\right) = \sum_{i=1}^{n} \left|z_i\right| + \left|z_{n+1}\right| + \left|z_{n+2}\right| + \left|z_{err}\right| \tag{44}$$

In order to contain the growth of the result in terms of extra terms generated during a non-affine operation, the extra terms need to be accumulated into a single error term z_{err}. Therefore the noise terms ε_{n+1} and ε_{n+2}, in the result of the multiplication operation, are regarded as the same source of uncertainty as ε_{err} as shown in (45).

$$\tilde{z} = z_0 + \sum_{i=1}^{n} z_i \varepsilon_i + z_{n+1}\varepsilon_{err} + z_{n+2}\varepsilon_{err} + z_{err}\varepsilon_{err} \tag{45}$$

The error term in (43) is now recalculated as the algebraic sum shown in (46)

$$z_{err} = b_1 + x_{err} y_l + y_{err} x_l \tag{46}$$

The term, ε_{err}, is now the only extra term generated during a multiplication operation of independent affine numbers in long chain of computation. In the case of dependent affine numbers, consider $\tilde{x}$ and $\tilde{y}$ as shown in (47) along with their interval form.

$$\begin{aligned} &\tilde{x} = x_0 + \sum_{i=1}^{m} x_i \varepsilon_i + x_{err}\varepsilon_{err};\ \ \tilde{y} = y_0 + \sum_{i=1}^{m} y_i \varepsilon_i + y_{err}\varepsilon_{err} \\ &\overline{x} = \left[x_l \ldots x_h\right];\ \ \overline{y} = \left[y_l \ldots y_h\right] \end{aligned} \tag{47}$$

According to the multiplication rule in (41), the result is computed as shown in (48).

$$\begin{aligned}
&\tilde{z} = \tilde{x} \times \tilde{y} = z_0 + \sum_{i=1}^{n+2} z_i \varepsilon_i + z_{err} \varepsilon_{err}; \text{ where } z_0 = \frac{z_h^* + z_l^*}{2} \\
&z_i = y_l x_i \text{ for } i \in M_x, i \neq err; \qquad z_i = x_l y_i \text{ for } i \in M_y, i \neq err; \\
&z_i = x_l y_i + y_l x_i \text{ for } i \in M_c; \\
&z_{n+1} = y_l x_{err}; \quad z_{n+2} = x_l y_{err} \\
&z_{err} = \frac{z_h^* - z_l^*}{2} - R, \quad R = \sum_{i=1}^{n+2} |z_i|
\end{aligned} \tag{48}$$

The coefficients of error terms, ε_{err} in $\tilde{x}$ and $\tilde{y}$, were treated independent of each other during the computation of optimal interval Z^*, using the method described in [2]. The extra terms z_{n+1} and z_{n+2} are treated in the same way as in the case of independent numbers shown in (45) and (46). The error term in the non-affine multiplication operation is recalculated as shown in (49).

$$z_{err} = \left[\frac{z_h^* - z_l^*}{2} - R \right] + y_l x_{err} + x_l y_{err} \tag{49}$$

Thus, there is only one additional term, z_{err}, generated as a result of non-affine multiplication operations even in a long chain of computation. The affine operations with the error terms included are shown in (50) – (51).

$$\tilde{x} \pm \tilde{y} = (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i) \cdot \varepsilon_i + (x_{err} \pm y_{err}) \cdot \varepsilon_{err} \tag{50}$$

$$\alpha \cdot \tilde{x} = (\alpha \cdot x_0) + (\alpha \cdot x_1) \varepsilon_1 + \cdots + (\alpha \cdot x_n) \varepsilon_n + (\alpha \cdot x_{err}) \varepsilon_{err} \tag{51}$$

$$\zeta \pm \tilde{x} = (\zeta \pm x_0) \pm x_1 \varepsilon_1 \pm \cdots \pm x_n \varepsilon_n \pm x_{err} \varepsilon_{err} \tag{52}$$

The modifications to the GIA described in this sub-section were implemented in our MATLAB toolbox and used to evaluate symbolic expressions of analog circuits. The experimental details are discussed in the next sub-section.

II. Experimental Results and Discussion

A MATLAB toolbox defined with basic arithmetic operations such as addition, subtraction, multiplication, division and exponentiation was developed for each method (AA, EAA and GIA) described in this paper. The GIA was implemented in this toolbox as described in section III (a). The symbolic expressions were generated using our symbolic circuit analysis toolbox also developed in MATLAB based on the framework and method described in [19].

The symbolic expressions for various circuit examples including those described in Table I were numerically evaluated using different range arithmetic methods implemented in the toolbox. The results for the cascode current mirror circuit and Operational Transconductance Amplifier circuit in Table I are discussed in this sub-section. The symbolic expressions for the circuits are generated using the symbolic circuit analysis toolbox. Each parameter in the circuit expressions is governed by an equation extracted and logged during the symbolic analysis procedure. In our experiments, nominal values for those variables in the extracted equations were obtained from numerical circuit analysis software SPICE. Each value obtained from SPICE was allowed to vary +/-2.5% in order to obtain its interval value. The interval values were then converted to affine numbers and substituted in the equations governing the parameters in the symbolic expressions. The affine arithmetic values for all the symbolic parameters in the circuit expressions were obtained in this manner. The symbolic expressions can now be substituted with affine values for the symbolic parameters and evaluated using one of the range arithmetic methods implemented in our toolbox. For the purpose of illustration, a product term and affine values of the symbolic parameters in the product term are shown in (53).

$$\begin{aligned}
&\text{product_term} = G_{dsN4} \times C_{dsN3} \times C_{gsN2} \\
&\text{where,} \quad G_{dsN4} = 0.00025 + 0.0000053 \varepsilon_{gds} \\
&\qquad C_{dsN3} = 3.4 \times 10^{-15} + 1.2 \times 10^{-16} \varepsilon_{cds} + 1.4 \times 10^{-18} \varepsilon_{cgs} + 2.3 \times 10^{-18} \varepsilon_{cgd} \\
&\qquad C_{gsN2} = 1.53 \times 10^{-15} + 7.2 \times 10^{-16} \varepsilon_{cgs} + 4.3 \times 10^{-18} \varepsilon_{cgd}
\end{aligned} \tag{53}$$

As can be seen from (53), the symbolic parameters sometimes share common noise symbols (as in case of C_{dsN3} and C_{gsN2}) and sometimes are independent (as in case of G_{dsN4} and C_{dsN3}). The equations governing the parameters were obtained using circuit analysis techniques in order to bring out the dependencies or correlation between circuit parameters [19]. The dependencies between the symbolic parameters will then help to achieve tighter bounds in the final computed result when the expressions are evaluated using one of the range arithmetic methods discussed in this paper.

TABLE II A comparison between different Range Arithmetic methods for Cascode Current Mirror example

Numerator Or Denominator	Relative Accuracy of coefficient of					Average Percentage	Natural Interval Extension compared against
	s^4	s^3	s^2	s^1	s^0		
Numerator	4.1	6.1	12.2	1.6	1	3.51	GIA
Denominator	4.3	2.3	1.4	1.1	1		
Numerator	3.8	5.3	7.1	1.5	0.97	2.81	AA
Denominator	3.9	2.1	1.35	1.1	0.97		
Numerator	3.6	4.9	8.1	1.5	0.97	2.83	EAA
Denominator	3.7	2.09	1.3	1.1	0.97		

In order to compare the different range arithmetic methods described in Section II, interval arithmetic was used as a base for comparison. The interval arithmetic computations were performed using the INTLAB toolbox available for use in MATLAB [21]. The results obtained using INTLAB were used to compute the relative accuracies of the AA, EAA and GIA methods. The relative accuracies for each coefficient computed using different range arithmetic methods for the cascode current mirror example is shown in Table II. As can be seen from this table, the GIA method has the highest overall relative accuracy among the three methods, as expected from the discussions in this paper.

The AA and EAA methods can sometimes be over conservative especially when the numbers are independent or share less noise symbols. For example, the relative accuracy for coefficients of s^0 in both numerator and denominator are less than 1 for both EAA and AA. This is because the coefficient for the s-power term consists of only conductance elements. The conductance elements are either independent or rarely share a noise symbol. The conservative error estimates during multiplication leads to the overestimation in the individual coefficient bounds, although the overall relative accuracy is greater than 1. The GIA method scores high relative accuracy for each coefficient in addition to the overall accuracy, indicating that it is suitable for circuit analysis applications.

The relative accuracy for all the three methods increases in case of higher powers of 's', because of increased dependency or correlation between the parameters at higher powers of '*s*'. In other words, the interval arithmetic produces overestimated results because it assumes parameters to vary independent of each other. The overestimation increases as the dependency between parameters increases. Therefore, interval arithmetic can be regarded as unsuitable for analysing circuits of large size. The CPU computation time between interval arithmetic and other range arithmetic are comparable, that is the computational expense of keeping track of dependencies is negligible, compared to the quality of results.

The frequency response plots for cascode current mirror and OTA are shown in Fig 3 and Fig 4 respectively. The plots compare the GIA method shown in green against Interval Arithmetic shown in black. As can be seen from the plots the interval arithmetic over estimates the results and is overly conservative at high frequencies. This is because the sum-of-products terms in the expression is dominated by capacitance components in the high frequency region. The capacitance parameters in the symbolic expressions are dependent on each other. This is consistent with the concepts of circuit analysis where at high frequencies, capacitors dominate and influence the circuit performance.

IV. Conclusion

Range arithmetic methods have useful applications in symbolic-numeric circuit analysis. They can efficiently handle variations in parameters which is very important in modern semiconductor technologies. In this paper we compared different

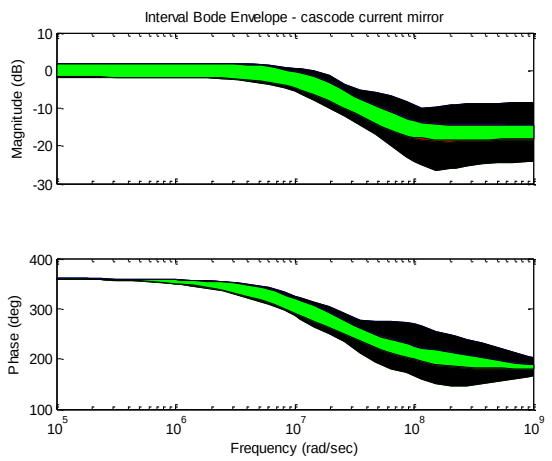

Figure 3 Frequency Response plot for Cascode Current Mirror Circuit

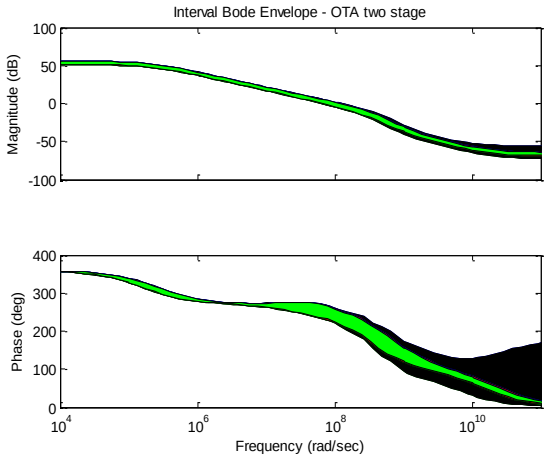

Figure 4 Frequency Response plot for OTA circuit

range arithmetic methods in applications to the evaluation of symbolic-numeric circuit expressions. The use of interval arithmetic leads to over estimation in the results and therefore can be regarded as unsuitable for this application. Although, affine arithmetic provides more information on dependencies between parameters, it also suffers from over estimation problems when the error in the non-affine approximation becomes large. The optimal bounds on the final results can be obtained by using a combination of Kolev's Method and Messine's multiplication rules. The bounds on the results of affine arithmetic computations will be narrower than the results from use of interval arithmetic. In addition affine arithmetic provides a facility to include and simulate uncertainties in a parameter easily.

References

[1] R. E. Moore and F. Bierbaum, *Methods and Applications of Interval Analysis (SIAM Studies in Applied and Numerical Mathematics) (Siam Studies in Applied Mathematics, 2.)*: Soc for Industrial \& Applied Math, 1979.

[2] L. Kolev, "Optimal Multiplication of G-intervals," *Reliable Computing*, vol. 13, pp. 399-408, 2007.

[3] F. Messine and A. Touhami, "A General Reliable Quadratic Form: An Extension of Affine Arithmetic," *Reliable Computing*, vol. 12, pp. 171-192, 2006.

[4] L. Kolev, "New Formulae for Multiplication of Intervals," *Reliable Computing*, vol. 12, pp. 281-292, 2006.

[5] G. Manson, "Calculating frequency response functions for uncertain systems using complex affine analysis," *Journal of Sound and Vibration*, vol. 288, pp. 487-521, 2005.

[6] L. H. d. Figueiredo and J. Stolfi, "Self-Validated Numerical Methods and Applications," in *IMPA, Rio de Janeiro, Brazil; July 1997. [fig-sto-97-iaaa]*, ed. Rio de Janeiro, Brazil, 1997.

[7] D. Degrauwe, *et al.*, "Improving interval analysis in finite element calculations by means of affine arithmetic," *Computers & Structures*, vol. 88, pp. 247-254, 2010.

[8] R. Martin, *et al.*, "Comparison of interval methods for plotting algebraic curves," *Computer Aided Geometric Design*, vol. 19, pp. 553-587, 2002.

[9] F. Messine, "Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization," *Journal of Universal Computer Science*, vol. 8, pp. 992-1015, 2002.

[10] B. Thanigaivelan, *et al.*, "A modified MOSFET small-signal model based on Affine Arithmetic concepts," presented at the PRIME Asia, Shangai, China, 2009.

[11] M. Pierzchala and B. Rodanski, "Road map representation of s-expanded symbolic network functions," in *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, 2007, pp. 858-861.

[12] S. X. D. Tan, "Symbolic Analysis of Analog Circuits By Boolean Logic Operations," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 53, pp. 1313-1317, 2006.

[13] S. X.-D. Tan and C.-J. R. Shi, "Efficient approximation of symbolic expressions for analog behavioral modeling and analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, pp. 907-918, 2004.

[14] S. X.-D. Tan, *et al.*, "Hierarchical approach to exact symbolic analysis of large analog circuits," in *Design Automation Conference, 2004. Proceedings. 41st*, 2004, pp. 860-863.

[15] G. G. E. Gielen, "Techniques and Applications of Symbolic Analysis for Analog Integrated Circuits: A Tutorial Overview," in *Computer-Aided Design of Analog Integrated Circuits and Systems*, G. G. E. Gielen, *et al.*, Eds., ed: IEEE Press, 2002, pp. pp. 245-261.

[16] F. V. Fernandez, *et al.*, Eds., *Symbolic analysis techniques: applications to analog design automation*. New York: IEEE Press, 1998, p.^pp. Pages.

[17] P. Wambacq, *et al.*, "Symbolic network analysis methods for practical analog integrated circuits: a survey," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 45, pp. 1331-1341, 1998.

[18] B. Thanigaivelan, *et al.*, "Efficient simplification strategies for symbolic circuit expressions of linear analog integrated circuits," in *SPIE International Symposium on Microelectronics, MEMS and Nanotechnology*, Brisbane, Australia, 2005.

[19] B. Thanigaivelan, *et al.*, "Symbolic-Numeric Circuit Analysis using Affine Arithmetic *(in review)*," *IEEE Transcastions on Computer Aided Design*, p. 10, 2011.

[20] L. V. Kolev, "An Improved Interval Linearization for Solving Nonlinear Problems," *Numerical Algorithms*, vol. 37, pp. 213-224, 2004.

[21] S. M. Rump. (2010, July 2010). *INTerval LABoratory (INTLAB)*. Available: http://www.ti3.tu-harburg.de/rump/intlab/