Numerical Algorithms **00:** 1–13, 2003.

Affine Arithmetic: Concepts and Applications

Luiz Henrique de Figueiredo (lhf@impa.br)
IMPA–Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina 110, 22461-320 Rio de Janeiro, RJ, Brazil

Jorge Stolfi (stolfi@ic.unicamp.br)
Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6176, 13083-970 Campinas, SP, Brazil

Received: ......; Accepted: ......

Abstract. Affine arithmetic is a model for self-validated numerical computation that affine arithmetic keeps track of first-order correlations between computed and input quantities. We explain the main concepts in affine arithmetic and it handles the dependency problem in standard interval arithmetic. We also describe some of its applications.

Keywords: interval arithmetic; range analysis; dependency problem

AMS subject classification: 65G20, 65G30, 65D17, 65D18, 68U05

1. Introduction

One of the main obstacles in the wide-spread use of interval methods in numerical computing is that the range estimates computed with standard interval arithmetic (IA) tend to be too large, especially in complicated expressions or long iterative computations. This overestimation is mainly due to IA's underlying assumption that the (unknown) values of the arguments to primitive operations may vary *independently* over their given intervals. If this assumption is not valid — that is, if there are any mathematical constraints between those quantities — then not all combinations of values in the given intervals will be valid. In that case, the interval obtained by interval arithmetic may be much wider than the exact range of the result quantity. This problem is known as the "dependency problem".

Affine arithmetic (AA) is a model for self-validated numerical computation that aims to attack the dependency problem in interval computations. AA keeps track of *first-order correlations* between computed and input quantities; these correlations are automatically exploited in primitive operations, with the result that in many cases AA is able to produce interval estimates that are much better than the ones obtained with standard interval arithmetic. Moreover, AA also implicitly provides a geometric representation for the joint range of related quantities that can be exploited to increase the efficiency of interval methods.

In this paper, we explain the main concepts in affine arithmetic and how it handles the dependency problem. We also describe some of its applications. The paper is organized as follows: In Section 2 we review interval arithmetic and the dependency problem. In Section 3 we describe *affine forms*, which represent quantities in affine arithmetic, and explain how to compute with them. In Section 4 we discuss examples of how affine arithmetic handles the dependency problem.

Several other methods have been proposed to attack the dependency problem. The main class of methods is known as *centered forms*, in several incarnations and generalizations, such as the *mean-value form* and *slopes*. Closest to our approach are Hansen's generalized interval arithmetic [6], Tupper's linear interval arithmetic [10], and first-order Taylor models [1]. A brief comparison between generalized interval arithmetic and affine arithmetic is given in Section 5.

2. The dependency problem in standard interval arithmetic

In interval arithmetic, quantities are represented by *intervals* and primitive operations and functions are extended to operate on intervals. More precisely, a quantity $x \in \mathbf{R}$ is represented by an interval $[a,b]$, where a and b are floating-point numbers (including $\pm\infty$), with the understanding that $a \leq x \leq b$.[1]

Primitive arithmetic operations can be extended to intervals:

$$\begin{aligned}
[a,b]+[c,d] &= [a+c,b+d] \\
[a,b]-[c,d] &= [a-d,b-c] \\
[a,b]\times[c,d] &= [\min(ac,ad,bc,bd),\max(ac,ad,bc,bd)] \\
[a,b]\ /\ [c,d] &= [a,b]\times[1/d,1/c] \quad (\text{provided } 0 \notin [c,d])
\end{aligned}$$

with special care to round lower bounds downwards and upper bounds upwards. Similar formulas can be given for extending the elementary functions to intervals [9].

Once we have interval formulas for all primitive operations and functions, we then automatically have formulas for all functions that can be obtained

[1] One of the original motivations for the creation of interval arithmetic was automatic, a posteriori, roundoff error analysis: real numbers representing input quantities were represented as intervals of consecutive floating-point numbers; output quantities computed from this input would be represented by intervals, which would be wide when the computation was contaminated with roundoff errors. This would signal the need to use more precision or to rearrange the computation to try to avoid contaminating roundoff errors. Unfortunately, the dependency problem turned out to be a much more serious source of overestimation than roundoff errors.

by combining these primitives. As a consequence, it is easy to extend any function $f\colon \Omega \subseteq \mathbf{R}^n \to \mathbf{R}$ to an interval function F that operates on the interval boxes $X \subseteq \Omega$ and produces an interval $F(X) \subset \mathbf{R}$ such that

$$F(X) \supseteq f(X) = \{f(x) : x \in X\}.$$

Such a function F is called an *inclusion function* for f. The interval estimate $F(X)$ often *overestimates* the exact range $f(X)$, that is, $F(X)$ is often strictly larger than the smallest interval containing $f(X)$.

Overestimation is a frequent event when f is given by an expression that contains multiple instances of one or more variables because the formulas for the primitive arithmetic operations given above assume that the operands are independent. When the operands partially depend on each other, not all combinations of values in the given intervals will be valid and the exact result interval will probably be smaller than the one produced by the formulas. This is the "dependency problem". The more complex the function f is, the worse the dependency problem becomes. In particular, for long iterative computations, we often see a rapid growth of the result interval at each stage.

As an extreme example of the dependency problem, take $f(x) = x - x$. Then of course $f(X) = \{0\}$ for every interval X. However, the interval subtraction formula gives

$$[a,b] - [a,b] = [a-b, b-a],$$

whose diameter is twice the diameter of $[a,b]$, instead of being zero!

For a less extreme example, take $f(x) = (10+x)(10-x)$ for $x \in X = [-2,2]$. Using the basic formulas, we get:

$$\begin{aligned} 10+X &= [8,12] \\ 10-X &= [8,12] \\ (10+X)(10-X) &= [64,144] \end{aligned}$$

So the interval formulas give an interval whose diameter is 80, whereas the exact interval $f(X) = [96,100]$ has diameter only 4. Note that when one operand in the product $(10+x)(10-x)$ is at the maximum value 12, the other must be at the minimum value 8; the combinations $8 \cdot 8$ and $12 \cdot 12$, which gave the extreme values of $F(X)$, do not happen.

A simple remedy for this example is to rewrite $(10+x)(10-x) = 100 - x^2$, which has only one occurrence of the variable x. An interval computation of this new expression will give the exact result. Unfortunately, this remedy is often impossible to apply in practice.

Here is another example that hints to what can happen in complicated expressions and long iterative computations. Take $g(x) = \sqrt{x^2 - x + 1/2}/\sqrt{x^2 + 1/2}$. The iterates of g converge to a constant function (the fixed point of g); however, as shown in Figure 1, the intervals computed for them with standard

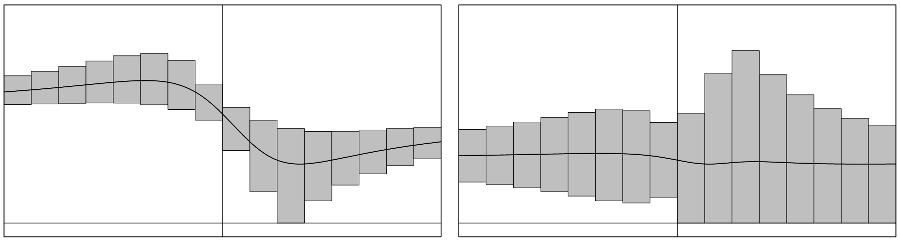


Figure 1. Catastrophic interval overestimation due to dependency.

interval arithmetic actually diverge, in the sense that their diameters get larger and larger.

3. Affine arithmetic: concepts

We have seen that interval arithmetic represents quantities with intervals. In affine arithmetic a quantity x is represented by an expression of the form

$$\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n,$$

which is an affine expression (i.e., first-degree polynomial) on the *noise symbols* ε_i with floating-point coefficients x_i — hence the name *affine form*. Each noise symbol ε_i is a symbolic real variable whose value is unknown except that it is restricted to $\mathbf{U} = [-1, +1]$ and is independent from the other noise symbols. The coefficient x_0 is called the *central value* of the affine form $\hat{x}$. The coefficients $x_1, \ldots, x_n$ are called the *partial deviations* associated with the noise symbols $\varepsilon_1, \ldots, \varepsilon_n$ in $\hat{x}$. The number n of noise symbols depends on the affine form: different affine forms can use a different number of noise symbols, some of which may be shared with other affine forms. As we shall see below, new noise symbols are created during computation.

Affine forms provide interval bounds for the corresponding quantities: If a quantity x is represented with the affine form $\hat{x}$ as above, then $x \in [x_0 - r_x, x_0 + r_x]$, where $r_x = |x_1| + \cdots + |x_n|$ is called the *total deviation* of $\hat{x}$. Conversely, if $x \in [a, b]$, then x can be represented with the affine form

$$\hat{x} = x_0 + x_i\varepsilon_i$$

where $x_0 = (b+a)/2$ and $x_1 = (b-a)/2$. In other words, AA algorithms can input and output intervals, and so AA can be use as a replacement for IA. However, as we shall see presently, affine forms give additional information that can be exploited to further bound the joint range of quantities.

3.1. The geometry of affine forms

The key feature of AA is that two or more affine forms can share noise symbols. (A noise symbol is *shared* when it appears with a non-zero coefficient in all affine forms in question.) When this happens, the quantities represented by those affine forms are not completely independent: they have a partial dependency for each noise symbol shared by their affine forms. This dependency can be measured by the corresponding partial deviations.

For concreteness, consider two quantities x and y represented by the affine forms

$$\begin{aligned}\hat{x} &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\ \hat{y} &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n.\end{aligned}$$

Whenever x_i and y_i are both non-zero, the values of x are y are partially dependent. While it is true that the affine forms above imply that $x \in X = [x_0 - r_x, x_0 + r_x]$ and $y \in Y = [y_0 - r_y, y_0 + r_y]$, the partial dependency implied by the shared noise symbols implies that the *joint range* Z of x and y is not simply the rectangle $R = X \times Y$. In fact, the joint range is

$$Z = \{(x,y) : \varepsilon_i \in \mathbf{U}, i = 1, \ldots, n\}.$$

In other words, Z is the image of the hypercube $\mathbf{U}^n$ under the affine transformation $\mathbf{R}^n \to \mathbf{R}^2$ given by

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix} \cdot (\varepsilon_1, \ldots, \varepsilon_n)^T.$$

Therefore, Z is *zonotope*, that is, a convex polygon that is centrally symmetric with respect to the point (x_0, y_0), the image of the origin $(0, \ldots, 0) \in \mathbf{R}^n$ [11]. In general, the joint range of m affine forms is a zonotope in $\mathbf{R}^m$, that is, a centrally symmetric convex polytope in $\mathbf{R}^m$.

This geometric information provided by AA has immediate applications in geometric modeling. Let $\mathcal{C}$ be a parametric curve given by $\gamma: I \subseteq \mathbf{R} \to \mathbf{R}^2$ and let $\mathcal{P} = \gamma(T) \subseteq \mathcal{C}$, where T is a subinterval of I. By representing $t \in T$ as an affine form and computing $\gamma(t) = (x(t), y(t))$ for $t \in T$ with AA, we obtain as explained above a zonotope Z that contains $\mathcal{P}$. To get a simpler enclosure for $\mathcal{P}$, we compute the rectangle of minimal width that contains Z. If we do this for a covering of the domain I, we get a covering of $\mathcal{C}$ by rectangles; see Figure 2. When these coverings are organized hierarchically, we get a multiresolution representation that can be used to solve several geometric problems on $\mathcal{C}$ efficiently [5]. Interval arithmetic can also provide guaranteed bounding rectangles for pieces of curves, but they are always aligned with the axes, and so converge more slowly to the curve. Compare the two approximations in Figure 3.

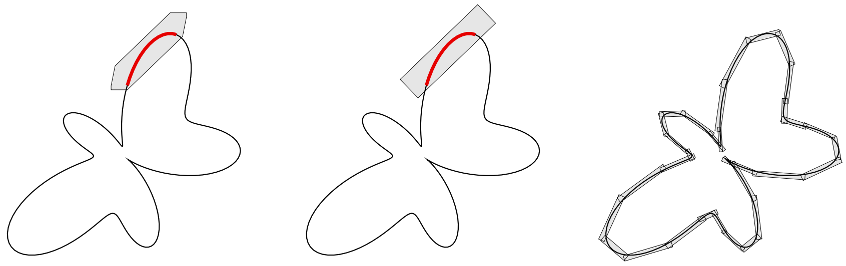


Figure 2. Approximating a parametric curve with rectangles.

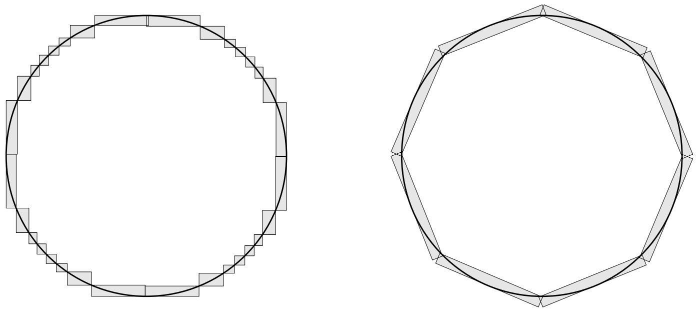


Figure 3. Rectangle approximation of a circle computed with interval arithmetic (left) and with affine arithmetic (right), using the same tolerance.

3.2. Computing with affine arithmetic

As done in interval arithmetic, computations in affine arithmetic are performed by first extending primitive operations and functions to operate on affine forms, and then combining these primitives to compute arbitrarily complex functions.

Affine operations are fairly straightforward: Given two affine forms

$$\begin{aligned}\hat{x} &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\ \hat{y} &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n\end{aligned}$$

and three real numbers α, β, and γ, we have

$$\alpha\hat{x} + \beta\hat{y} + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + (\alpha x_1 + \beta y_1)\varepsilon_1 + \cdots + (\alpha x_n + \beta y_n)\varepsilon_n.$$

Extending non-affine operations requires that we use good affine approximation to the exact result and append an extra term to bound the error of this approximation.

For concreteness (but without loss of generality), suppose we have to compute a non-affine operation of two variables, $z = f(x,y)$. Given affine forms $\hat{x}$

and $\hat{y}$ representing x and y respectively, we want to compute an affine form $\hat{z}$ representing z that is consistent with $\hat{x}$ and $\hat{y}$ and that preserves the information provided by them as much as possible. First, we see z as a function of the noise symbols in $\varepsilon_1, \ldots, \varepsilon_n$:

$$\begin{aligned} z &= f(x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n) \\ &= f^*(\varepsilon_1, \ldots, \varepsilon_n), \end{aligned}$$

where f^* is a function $\mathbf{U}^n \to \mathbf{R}$. In general, f^* is not an affine function of ε_1, $\ldots$, ε_n. So, we approximate f^* over $\mathbf{U}^n$ by an affine function f^{a} with error bound δ:

$$|f^*(\varepsilon_1, \ldots, \varepsilon_n) - f^{\mathrm{a}}(\varepsilon_1, \ldots, \varepsilon_n)| \leq \delta \text{ for all } \varepsilon_1, \ldots, \varepsilon_n \in \mathbf{U}.$$

Writing

$$f^{\mathrm{a}}(\varepsilon_1, \ldots, \varepsilon_n) = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n,$$

we obtain that $z = f(x, y)$ is represented by the affine form

$$\hat{z} = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n + z_k\varepsilon_k,$$

where $z_k = \delta$ and ε_k is a new noise symbol. The challenge is to find an affine approximation f^{a} that is easy to compute but which has a small approximation error δ.

Note that the introduction of the term $z_k\varepsilon_k$ to represent the non-affine part of $f^*(\varepsilon_1, \ldots, \varepsilon_n)$ implies a loss of information: from this point on, the noise symbol ε_k will be implicitly assumed to be independent from $\varepsilon_1, \ldots, \varepsilon_n$, when in fact it is a (non-affine) function of them. Any subsequent operation that takes $\hat{z}$ as input will not be aware of this constraint between ε_k and $\varepsilon_1, \ldots, \varepsilon_n$, and therefore may return an affine form that is less precise than necessary.

In most contexts, the error bound δ will depend quadratically on the widths of the ranges of the input variables $\hat{x}$ and $\hat{y}$, so that its magnitude will decrease (even in the relative sense) as those ranges become smaller.

For simplicity, the formulas above, for both affine and non-affine operations, do not take into account rounding errors. In practice, rounding errors are handled by adding a new noise symbol to carry them or by absorbing them into the approximation error term $z_k\varepsilon_k$, in the case of non-affine operations. For details, see [9].

3.3. Selecting a good affine approximation

As discussed above, to compute with affine arithmetic we must find a good affine approximation f^{a} for each primitive non-affine operation f. When we write f^{a} as a function of the noise symbols in the input forms $\hat{x}$ and $\hat{y}$,

$$f^{\mathrm{a}}(\varepsilon_1, \ldots, \varepsilon_n) = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n,$$

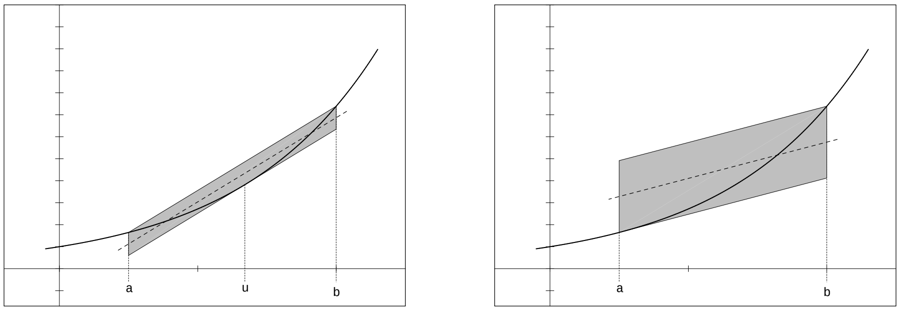


Figure 4. Chebyshev $\times$ minimum range approximations.

we see that there are $n+1$ degrees of freedom for the choice of f^{a}, corresponding to the choices of $z_0, \ldots, z_n$. For simplicity and efficiency, we consider only approximations f^{a} that are themselves affine combinations of the input forms $\hat{x}$ and $\hat{y}$:

$$f^{\mathrm{a}} = \alpha\hat{x} + \beta\hat{y} + \gamma.$$

This simplification reduces the number of parameters to determine from $n+1$ to just three. For univariate functions $f(x)$, the restriction is in fact harmless, because the best affine approximation f^{a} is indeed of the form $\alpha\hat{x}+\gamma$. In any case, the error term for the restricted approximation is still quadratic on the size of the input ranges.

There are two basic approaches to finding a good affine approximation to a non-affine univariate function: minimize the error or minimize the range. In the first approach, we select the affine approximation that has the smallest possible error over the given input interval. This affine approximation is then the best approximation in the Chebyshev (or minimax) sense. This approximation is optimal in the sense that it minimizes the measure of the polytope defined by the affine forms $\hat{x}$, $\hat{y}$, and $\hat{z}$ — i.e., it preserves the maximum information about the triple (x, y, z). However, the best approximation does not necessarily have the smallest range for $\hat{z}$ alone. For example, Figure 4 (left) shows the Chebyshev approximation for the exponential function. The approximation appears as a dotted line. The gray region is the zonotope corresponding to the error term. Note that this approximation underestimates the lower bound in the given interval. Figure 4 (right) shows the affine approximation that has the smallest range, that is, the exact range of the exponential function on that interval. Note that its zonotope is larger than the Chebyshev zonotope. Nevertheless, the diameters of both zonotopes still depend quadratically on the diameter of the input interval.

The choice of which affine approximation to use — Chebyshev or minimum range — depends on the application and on the function: in some applications it is important to compute interval bounds that contain only positive numbers if the function itself is always positive, as is the case with the exponential function. (The Chebyshev approximation of the exponential function over a relatively large interval around zero will contain negative numbers.) Moreover, the minimum range approximation is usually easier to compute. On the other hand, the use of the Chebyshev approximation is of course recommended because it gives the best approximation, that is, the smallest zonotope.

3.4. Multiplication in affine arithmetic

The most important primitive operations that are not univariate are multiplication and division. If we write $x/y = x \times (1/y)$, then we can concentrate on the multiplication.

Given two affine forms

$$\begin{aligned}\hat{x} &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\ \hat{y} &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n,\end{aligned}$$

their product is

$$\begin{aligned}\hat{x}\cdot\hat{y} &= (x_0 + \sum_{i=1}^{n} x_i\varepsilon_i)\cdot(y_0 + \sum_{i=1}^{n} y_i\varepsilon_i) \\ &= x_0y_0 + \sum_{i=1}^{n}(x_0y_i + y_0x_i)\,\varepsilon_i + \sum_{i=1}^{n} x_i\varepsilon_i \cdot \sum_{i=1}^{n} y_i\varepsilon_i\end{aligned}$$

So we can write the following affine form for the product:

$$\hat{x}\cdot\hat{y} = x_0y_0 + \sum_{i=1}^{n}(x_0y_i + y_0x_i)\,\varepsilon_i + z_k\varepsilon_k,$$

where

$$|z_k| \geq \left|\sum_{i=1}^{n} x_i\varepsilon_i \cdot \sum_{i=1}^{n} y_i\varepsilon_i\right|, \qquad \varepsilon_i \in \mathbf{U},$$

is an upper bound for the approximation error, as before. The simplest bound is

$$z_k = \sum_{i=1}^{n} |x_i| \cdot \sum_{i=1}^{n} |y_i|,$$

which is at most four times the error of the best affine approximation, but is very easily computed (as far as we know, it takes $\Theta(m \log m)$ time to compute the best affine approximation to the product, where m is the number of nonzero noise terms in $\hat{x}$ and $\hat{y}$).

4. Affine arithmetic and the dependency problem

We now return to the examples in Section 2 to check how affine arithmetic works on the dependency problem.

We start with the extreme example, $f(x) = x - x$. According to the formulas in Section 3.2, the difference $\hat{x} - \hat{x}$ between an affine form and itself is identically zero. The subtraction formula "knows" that, in this case, the operands are actually the same quantity — because they share the same noise symbols with the same coefficients — and not just two quantities that happen to have the same range of possible values.

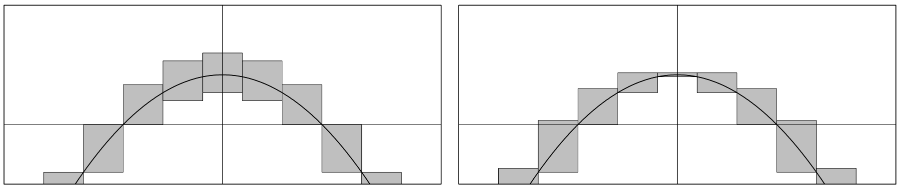
For the same reason, linear identities such as $(\hat{x}+\hat{y}) - \hat{x} = \hat{y}$ or $(3\hat{x}) - \hat{x} = 2\hat{x}$, which do not hold in standard interval arithmetic, do hold in affine arithmetic (except for floating-point roundoff errors). More generally, computations that consist only of affine operations with numeric coefficients will have relative accuracy near unity.

Consider now the quadratic example, $f(x) = (10+x)(10-x)$, for $x \in [-u, u]$ with $u > 0$. In Section 2 we used $u = 2$, but here we want to show what happens when $u \to 0$. Writing $\hat{x} = 0 + u\varepsilon$, we get

$$\begin{aligned} 10+\hat{x} &= 10 - u\varepsilon \\ 10-\hat{x} &= 10 + u\varepsilon \\ (10+\hat{x})(10-\hat{x}) &= 100 - u^2\,\varepsilon \end{aligned}$$

which implies that $f(x) \in [100 - u^2, 100 + u^2]$. The exact range is $[100 - u^2, 100]$ and so the interval computed with affine arithmetic is only twice as large as the exact result. Standard interval arithmetic gives $[(10-u)^2, (10+u)^2]$, whose diameter is $40u$. Contrast this with $2u^2$, which is the diameter of the interval computed with affine arithmetic, and u^2, which is the diameter of the exact result. Thus, affine arithmetic gives an interval that shrinks to a point as $u \to 0$ at the same speed as the exact result, whereas standard interval arithmetic gives an interval that converges much more slowly. See Figure 5.

Next, consider $g(x) = \sqrt{x^2 - x + 1/2}/\sqrt{x^2 + 1/2}$ and its iterates. Recall that the iterates of g converge to a constant function but the intervals computed for them with standard interval arithmetic diverge. When affine arithmetic is used we see a rapid convergence; contrast Figure 6 with Figure 1.



IA AA

Figure 5. Quadratic convergence.

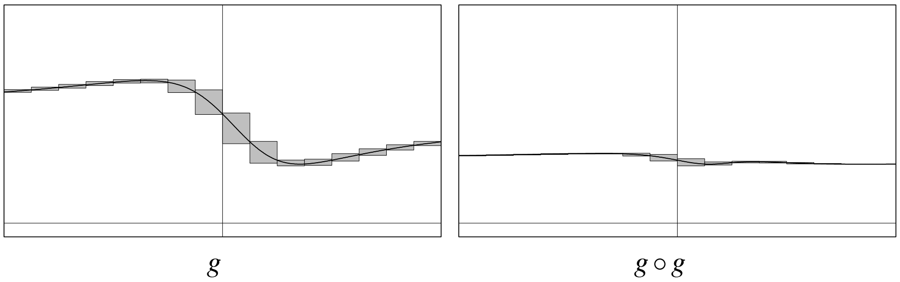


Figure 6. Avoding catastrophic overestimation.

These simple examples show that AA can used a mere replacement for standard interval arithmetic. Even if no attempt is made to exploit the additional information on joint ranges provided by the zonotopes enclosures computed with AA (that is, affine forms for the results of the computation are simply converted to intervals), the automatic exploitation of first-order correlations in primitive operations translates into better interval estimates for the results. For more examples in actual applications, see [3, 4, 8, 7]. For examples that do exploit the additional information on joint ranges, see [2, 5].

5. Affine arithmetic versus generalized interval arithmetic

Affine arithmetic is similar to *generalized interval arithmetic* (GIA), a computation model proposed by Hansen in 1975 [6] and recently studied in detail by Tupper [10]. In its original formulation, GIA addresses specifically the problem of computing one or more functions from a fixed set of n quantities $x_1, \ldots, x_n$, which are given as ordinary intervals $\bar{x}_1, \ldots, \bar{x}_n$. Every quantity z that is derived from these variables, including the final results, is represented by a GIA form

$$\tilde{z} = \bar{\zeta}_0 + \bar{\zeta}_1 x_1 + \bar{\zeta}_2 x_2 + \cdots + \bar{\zeta}_n x_n,$$

where each $\bar{\zeta}_i$ is an interval. The x_i in this formula are unknown quantities, so the formula is kept unevaluated — just as affine forms of AA. From this representation we can obtain an interval enclosure for z:

$$z \in \bar{\zeta}_0 + \bar{\zeta}_1 \bar{x}_1 + \bar{\zeta}_2 \bar{x}_2 + \cdots + \bar{\zeta}_n \bar{x}_n,$$

where the interval expression is evaluated as in standard IA.

This model is obviously similar to AA, not only in its representation, but also in its main virtue: in principle, it can model affine dependencies between quantities, with error that shrinks quadratically with the size of the input intervals.

However, GIA and AA are not mathematically equivalent, and neither is a special case of the other. In general, conversion from one representation to

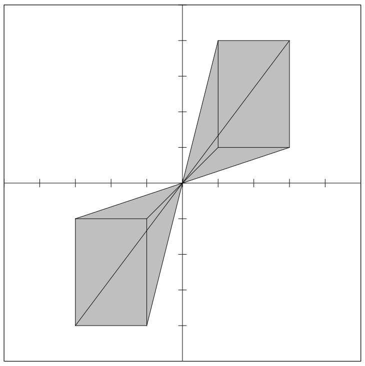


Figure 7. Joint range in GIA.

the other entails loss of information. The difference between the two models is evident when one considers the joint range of two quantities u and v when they are described in either model: As we have seen, in AA the joint range is always a zonotope, whereas in GIA it is typically a bowtie-shaped as shown Figure 7 for the GIA forms

$$\begin{aligned} \tilde{u} &= [0,0] + [1,3]\bar{x} \\ \tilde{v} &= [0,0] + [1,4]\bar{x}, \end{aligned}$$

where x is the only input variable, which ranges over $[-1,+1]$.

Moreover, since the GIA coefficients are intervals, rather than numbers, uncertainty cancellations will not be as complete as it can be in AA. For example, if $\tilde{u}$ is the form given above, then the GIA evaluation of $\tilde{u} - \tilde{u}$ will produce $[0,0] + [-2,+2]x$, whereas the analogous computation in AA returns exactly zero. Also, by lumping all internal errors into the independent interval $\bar{\zeta}_0$ (or in the coefficients $\bar{\zeta}_1, \ldots, \bar{\zeta}_n$), the GIA model prevents those errors from canceling out. Thus, we can expect that catastrophic overestimation will occur in GIA more often than in AA.

For a fuller discussion of some practical details where GIA and AA differ, see [9].

6. Conclusion

On the whole, we consider AA a good compromise between complexity and efficiency: although it is clearly more complex than standard IA (but similar to GIA), AA automatically exploits first-order correlations in complex expressions, not only for input variables but also for intermediate results — this alone frequently results in better interval estimates and even in faster solutions, despite AA's larger cost per primitive. Moreover, AA provides a good geometric representation for joint ranges as zonotopes that can be exploited to improve interval methods.

Acknowledgements

L. H. de Figueiredo thanks the organizers of SCAN 2002 (René Alt, Jean Vignes, and Jean-Luc Lamotte) for the invitation to present this work in Paris. The authors are partially supported by CNPq research grants. L. H. de Figueiredo is a member of Visgraf, the computer graphics laboratory at IMPA, which is sponsored by CNPq, FAPERJ, FINEP, and IBM Brasil.

References

1. Berz, M. and G. Hoffstätter: 1998, 'Computation and application of Taylor polynomials with interval remainder bounds'. *Reliable Computing* **4**(1), 83–97.
2. de Cusatis Jr., A., L. H. de Figueiredo, and M. Gattass: 1999, 'Interval methods for ray casting implicit surfaces with affine arithmetic'. In: *Proceedings of SIBGRAPI'99*. pp. 65–71.
3. de Figueiredo, L. H.: 1996, 'Surface intersection using affine arithmetic'. In: *Proceedings of Graphics Interface'96*. pp. 168–175.
4. de Figueiredo, L. H. and J. Stolfi: 1996, 'Adaptive enumeration of implicit surfaces with affine arithmetic'. *Computer Graphics Forum* **15**(5), 287–296.
5. de Figueiredo, L. H., J. Stolfi, and L. Velho: 2002, 'Approximating parametric curves with strip trees using affine arithmetic'. In: *Proceedings of SIBGRAPI 2002*. pp. 163–170.
6. Hansen, E.: 1975, 'A generalized interval arithmetic'. In: K. Nickel (ed.): *Interval Mathematics*. pp. 7–18. Lecture Notes in Computer Science 29, Springer-Verlag.
7. Heidrich, W. and H.-P. Seidel: 1998, 'Ray-tracing procedural displacement shaders'. In: *Proceedings of Graphics Interface'98*. pp. 8–16.
8. Heidrich, W., P. Slusallek, and H.-P. Seidel: 1998, 'Sampling procedural shaders using affine arithmetic'. *ACM Transactions on Graphics* **17**(3), 158–176.
9. Stolfi, J. and L. H. de Figueiredo: 1997, *Self-Validated Numerical Methods and Applications*. Rio de Janeiro: Monograph for 21st Brazilian Mathematics Colloquium, IMPA. Available at `ftp://ftp.tecgraf.puc-rio.br/pub/lhf/doc/cbm97.ps.gz`.
10. Tupper, J. A.: 1996, 'Graphing Equations with Generalized Interval Arithmetic'. Master's thesis, Graduate Department of Computer Science, University of Toronto. Available at `http://www.dgp.utoronto.ca/people/mooncake/msc.html`.
11. Ziegler, G. M.: 1995, *Lectures on Polytopes*, Graduate Texts in Mathematics, 152. Springer-Verlag.