

Affine Arithmetic*

Jun Inuoe

Gregory M. Malecha

February 01, 2008

The motivation for the talk is the understanding how to control floating point error during calculations. The paper looks at this as it applies to DSPs for floating point computation; however, Jun's interest lies in numerical simulation.

Since we're interested in controlling the amount of error, we want to be able to compute the amount of error that a computation can introduce. There are several ways that this can be done:

Simulation Compute the error based on running a wide variety of input through the algorithm and computing the error. The problem with this is that it is time consuming and does not provide a guarantee.

Interval Arithmetic Represent each number as a range of values. This provides a formal bound but, because the source of the error is not tracked, this often considerably overestimates the error.

Affine Arithmetic Represent each number as a linear combination of errors and combine errors at operations by applying the operation to the equations.

This talk focuses on the affine arithmetic which is good because it has uniform reasoning about different operations and is relatively cheap. (Corky isn't convinced that it is cheap without sacrificing bound tightness)

1 Interval Arithmetic

A short example of interval arithmetic is enough to gain a basic understanding of the concept. Consider the following code:

code	values
<code>get(x,y)</code>	$\hat{x} = [x - \epsilon_x, x + \epsilon_x], \hat{y} = [y - \epsilon_y, y + \epsilon_y]$
<code>z := x + y</code>	$\hat{z} = [(x - y) - \epsilon_z, (x - y) + \epsilon_z], \epsilon_z = \epsilon_x + \epsilon_y$
<code>n := z - y</code>	$\hat{n} = [x - (\epsilon_z + \epsilon_y), x + (\epsilon_z + \epsilon_y)]$

*Floating-Point Error Analysis Based On Affine Arithmetic. Claire Fang Fang, Tsuhan Chen, Rob A. Rutenbar (2003)

Since the intervals are represented as pairs of numbers, the source of the error is lost and the final value of n has much larger error than one would expect. Affine arithmetic seeks to solve this problem.

2 Affine Arithmetic

To address the above problem, we need to keep the sources of the error around. We can represent a number in the following *affine form*:

$$\hat{x} = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + \dots + x_n\epsilon_n$$

In this equation the x_i represent the maximum error contribution of the term. These values are known. The ϵ_i represent the amount of effect that the corresponding x_i has. These values are unknown, but it is known that there exists $(\epsilon_1, \dots, \epsilon_n) \in [-1, 1]^n$ such that the above equation holds where \hat{x} is the value being stored by the program.

Repeating the above example when operation error is not factored in is shown below:

code	values
<code>get(x,y)</code>	$\hat{x} = x_0 + x_1\epsilon_x, \hat{y} = y_0 + y_1\epsilon_y$
<code>z := x + y</code>	$\hat{z} = x_0 + y_0 + x_1\epsilon_x + y_1\epsilon_y$
<code>n := z - y</code>	$\hat{n} = x_0 + x_1\epsilon_x$

The above example is idealized for two reasons. First, if non-linear operations are performed, then performing the operations on the affine forms will yield a non-affine form. An approximation of this form is taken but will lead to a tend toward larger radii. Second, the error contributed by operations is ignored in this model. For example, if x is very small and y is very large, then $x + y = y$ is possible in which case the error is the entire value of x .

3 Analysis

Analyzing the results can be done conservatively or aggressively. A conservative approach is to assume that all ϵ_i are equal to 1 which leads to maximal error radii. Experimental results from the paper suggest that these can be roughly 3X the maximum error from simulation.

The aggressive approach is treat the epsilon as random variables using the following equation:

$$\frac{c_1\epsilon_1 + c_2\epsilon_2 + \dots + c_n\epsilon_n - n\mu}{\sigma\sqrt{n}} \sim \mathcal{N}(0, 1)$$

This equation yields better results, but often requires assumptions that can not be rigorously justified.