Validated Constraint Compilation

Timothy J. Hickey and David K. Wittenberg

Computer Science Department, Brandeis University
{tim|dkw}@cs.brandeis.edu

Inaccurate scientific computation is useless at best and dangerous at worst. We address several major sources of inaccuracy. Roundoff error is well known and there is a great deal of work on minimizing it [Act96,Tay97]. By using interval constraints, we don't eliminate roundoff error, but we make it explicit, so each answer comes with a clear indication of its accuracy. Another source of error arises from misapplying an algorithm (e.g. starting the Newton method with a poor initial choice, or using a method in a case where it does not perform well). We propose a method for reducing the chance of numerical errors in scientific programming by casting the problem as the design of an appropriate constraint solving algorithm and then separating the algorithm design process into two steps.

- First, the automatic construction of a set of validated constraint contractors which can be applied to the initial intervals in any order without affecting the correctness of the algorithm, and
- Second, the specification of the algorithm which determines which contractors are applied in which order. The specification of which operators are applied in what order determines the speed of convergence of the algorithm, but has no effect on correctness, which depends only on the correctness of the constraint contractors.

The chief advantage of our method is that it results in a procedure which returns an interval which is guaranteed[1] to contain the correct solution. We provide a comparison of the classical method with our proposed method for creating a procedure to invert $f(x) = x\sin(x)$ on a subset of its domain.

Standard Methods: We consider the following simple numerical problem ([Act96], pp. 77-80, pp. 223-224): Solve the validated constraint contraction problem for the following constraint:

$$h = x\sin(x), \quad 0 \le x \le a_1,$$

where h is the input variable and x is the output variable. The method of first choice in numerical analysis is the Newton method, which works will except near the endpoints of the interval. Acton suggests two different approaches near these dangerous points. Near zero, he uses what he calls the pseudoconstant gambit, which converges rapidly for h near zero, but is slow for other h. Near the right endpoint $b_1 \approx 1.8197$, he uses what he calls the Quadratic method, which converges rapidly only near b_1. The end result of this standard numerical approach is

[1] Throughout this paper, we assume that hardware and compilers correctly meet their specifications.

to write a procedure which uses some number of iterations of the pseudoconstant gambit for h near zero, some number of iterations of the quadratic method for h near the upper bound b_1, and some number of iterations of the Newton method for the remaining cases. This method of constructing and analyzing numerical routines is highly heuristic and does not provide any guarantee that the stated precision bounds will be achieved. Either an interval arithmetic approach or a detailed line-by-line numerical analysis will be required if one wants a provably correct estimate of the error.

The key idea behind interval arithmetic constraints [BO97] is to view numeric computing problems as constraint systems that relate a set of real (or complex) variables or functions. The variables whose values are to be computed are initially unbounded in this model (i.e., they have the value $[-\infty, \infty]$). The goal of the computation is to shrink the intervals of the variables in such a way that no solution to the original system is removed. The shrinking is done by iteratively applying various contraction operators which are automatically generated from the constraint set.

In the full version of this paper [HW99] we present the **The Constraint Contractor Method**. The basic idea is to define a general family of validated contractors, based on arithmetic constraints[BO97] and to experiment with various combinations of these contractors. When a reasonably good sequence of contractions is found, one can then generate a procedure to implement these contractors and to signal an exception if the width of the computed intervals exceeds the required error bounds. This approach has three key benefits:

- First, the contractors can be generated automatically from the constraints using various parameters specified by the user (thereby eliminating a potential source of programming errors).
- Second, if the particular sequence of contractors does not sufficiently narrow the result intervals, then this can be detected at runtime and an exception can be thrown.
- Third, the contractors can be applied in any order without affecting the correctness of the algorithm. The only possible danger is that for some inputs, the particular sequence of contractors does not narrow the result interval sufficiently, which as noted above can be detected and handled at runtime.

References

[Act96] Forman S. Acton. *Real computing made real: Preventing Errors in Scientific and Engineering calculations.* Princeton University Press, Princeton, New Jersey, 1996.

[BO97] Frédéric Benhamou and William J. Older. Applying interval arithmetic to real, integer, and Boolean constraints. *Journal of Logic Programming*, 32, 1997.

[HW99] Timothy J. Hickey and David K. Wittenberg. Validated constraint compilation. Technical Report CS-99-201, Computer Science Department, Brandeis University, April 1999. URL: www.cs.brandeis.edu/~tim/Papers/cs99201.ps.gz.

[Tay97] John R. Taylor. *An introduction to Error Analysis: The Study of Uncertainties in Physical Measurements.* University Science Books, second edition, 1997.