3.5 Summary

In this section, we have introduced new method to perform validated numerical calculations for embedded applications.

Numerical validation tools have existed before, but none of those are specifically designed for embedded applications, because they lack support for fixed point representation. Our library tries to fill this gap. It is based on the use of a new library that applies various known validation methods to fixed point numbers.

This library is just the first piece of work towards a complete toolbox dedicated to numerical validation of embedded applications.

4 GlobSol (R. Baker Kearfott)

4.1 Introduction

GlobSol began as a research code to study algorithms for verified Global Optimization. GlobSol grew out of `INTBIS` [23], a relatively simple FORTRAN-77 code and *ACM Transactions on Mathematical Software* algorithm for finding all solutions, with validation, to nonlinear algebraic systems. For ease of experimentation, simple automatic differentiation, consistent with the relatively small problems originally envisioned, was added, and a special technique for bound constraints (originally tried in [16]) was implemented. We also provided extensive capability for a technique we described in [15], a technique (discovered independently and probably earlier by others) that has developed into the field of "constraint propagation." One of the first projects done within this environment was development of techniques for avoiding the "cluster" problem ([7], [22], [46]) that occurs in exhaustive search algorithms when the system is ill-conditioned or singular near the global optimum. We also implemented a technique for verifying feasible points [19] and thus included a capability for handling general equality-constrained problems. (We added separate handling of inequality-constrained problems later.) We studied and implemented extensions to the idea of interval slopes and slope arithmetic (perhaps first appearing [25]) to non-smooth functions, as we explained in [17] and [18, Ch. 6].

During this development (roughly from 1993 to 1998), we referred to GlobSol as `INTOPT-90`. A collected review of these and other techniques, some new theoretical analyses, and a description of the structure of `INTOPT-90` appears in [18].

GlobSol took on its present form (and its present name) as part of a cooperative research and development contract funded by Sun Microsystems and directed by G. W. Walster (and with extensive participation of George Corliss). The most significant advances during this phase of GlobSol's development are perhaps

- extensive testing and bug-removal (extremely important for software that purports to validate),

– polishing of the user interface,
– experimentation with GlobSol on a variety of practical problems, and
– polishing of the packaging, distribution, and installation process.

Although at first glance these advances may seem mundane, they are both a significant part of the total effort and absolutely indispensable for widely-used, lasting software.

We have recently provided some details of the above in the succinct review [20]. Here, we very briefly review requirements for installation and use of GlobSol, then focus on present weaknesses in GlobSol and how we are eliminating these weaknesses.

4.2 Statement of the Problem GlobSol Treats

For reference below, we now formally state the type of problem GlobSol solves. The general optimization problem is

$$\boxed{\begin{array}{l}\text{minimize } \phi(x)\\ \text{subject to } c_i(x) = 0,\ i = 1, \ldots, m_1,\\ \qquad\qquad g_i(x) \le 0,\ i = 1, \ldots, m_2,\\ \text{where } \phi : \mathbb{R}^n \to \mathbb{R} \text{ and } c_i, g_i : \mathbb{R}^n \to \mathbb{R}.\end{array}} \tag{2}$$

The sense in which GlobSol will solve problem (2) is

$$\boxed{\begin{array}{l}\text{Given a box } \boldsymbol{x} = ([\underline{x}_1, \overline{x}_1], \ldots [\underline{x}_n, \overline{x}_n]), \text{ find small boxes}\\ \boldsymbol{x}^* = ([\underline{x}_1^*, \overline{x}_1^*], \ldots [\underline{x}_n^*, \overline{x}_n]^*) \text{ such that any solutions of}\\ \text{minimize } \phi(x)\\ \text{subject to } c_i(x) = 0,\ i = 1, \ldots, m_1,\\ \qquad\qquad g_i(x) \le 0,\ i = 1, \ldots, m_2,\\ \text{where } \phi : \mathbb{R}^n \to \mathbb{R} \text{ and } c_i, g_i : \mathbb{R}^n \to \mathbb{R}\\ \text{are guaranteed to be within one of the } \boldsymbol{x}^* \text{ that has been found.}\end{array}} \tag{3}$$

4.3 Installation and Use of GlobSol

The main requirements for GlobSol are

1. a standard-conforming Fortran 90 or Fortran 95 compiler, and
2. a "make" utility.

A Fortran compiler is required because the user defines the optimization problem as a Fortran program. Even though GlobSol is compiled and linked only once (and the user's program is compiled and linked separately), the same version of the same compiler must nonetheless be used for both building GlobSol and compiling the user's input.

GlobSol can be obtained as a "zip" file from
`http://interval.louisiana.edu/GlobSol/download_globsol.html`
From there, one downloads a compressed file and an "unpack" script appropriate

to the particular operating system and compiler. The scripts are for compilers on various Unix/Linux and Microsoft systems. However, the makefile that builds GlobSol has extensive in-line documentation, and can be changed as appropriate for new compilers and systems.

Succinct instructions for installing GlobSol appear in
`http://interval.louisiana.edu/GlobSol/install.html`.

GlobSol has extensive configuration options, accessible by editing a configuration file. GlobSol is run by supplying a command-line script. A simple example is accessible by following the installation instructions. For more details, see [20], or examine the various preprints related to GlobSol at
`http://interval.louisiana.edu/preprints.html`.

4.4 Improvements to GlobSol in Progress

GlobSol works relatively well for unconstrained problems, but performs weakly when there are many equality constraints. There are several reasons for this. We give these reasons, along with present work to overcome these problems, in the following paragraphs.

Obtaining Upper Bounds on the Global Optimum First, GlobSol is weak at finding an upper bound on the global optimum, when constrained optimization is used. For unconstrained optimization, GlobSol (and other interval branch and bound algorithms) can obtain an upper bound on the global optimum by evaluating the objective function at any point $\check{x}$ (and using outwardly rounded interval arithmetic in the evaluation, for mathematical rigor); the closer $\check{x}$ is to an actual global optimizing point x^*, the sharper the upper bound on the global optimum. For constrained problems, there is a complication as outlined in [18, §5.2.4]: the interval evaluation needs to be taken over a small box in which a feasible point has been proven to lie. However, the same principle holds for constrained problems.

For unconstrained problems, GlobSol uses a simple steepest descent procedure followed by the MINPACK-1 routine `HYBRJ1` [33] to find a critical point of the Fritz–John equations, to increase the chances that $\check{x}$ is near a global optimizer. The MINPACK routines are freely available through NETLIB (`http://www.netlib.org/`), and can thus be distributed with GlobSol. In contrast, until recently, good routines that find approximations $\check{x}$ to local optimizers of constrained problems have been proprietary, and cannot be distributed with GlobSol. Since GlobSol is meant to be self-contained, we have instead provided our own routine that employs a generalized-inverse-based Newton method to project onto the feasible set [21]. As a consequence, in the constrained case, GlobSol finds rigorous upper bounds for the global optimizer, but may not find a reasonably sharp upper bound until late in the search process. For some applications, this is not a problem, but it can have a disastrous effect on efficiency in others.

Recently, Wächter's quality Fortran code Ipopt for constrained optimization (see `http://www-124.ibm.com/developerworks/opensource/coin/` and [50]) has become available under the Common Public License. (See `http://www.opensource.org/licenses/cpl.php`.) This code should provide approximate feasible points $\check{x}$ that are highly likely to be near global optimizers, thus enabling GlobSol to compute sharp upper bounds on global optimizers in the constrained case. We have recently interfaced Ipopt with GlobSol, and we are formulating experiments to analyze performance improvements.

Obtaining Lower Bounds on the Range over Large Regions A good upper bound on the global optimum is generally combined in global search algorithms with good lower bounds on the range of the objective function over subregions $\boldsymbol{x}$ of the search space. If the lower bound of the objective over $\boldsymbol{x}$ is larger than the upper bound on the global optimum, then the subregion $\boldsymbol{x}$ can be rejected as not containing any global optima. In principle, a simple interval evaluation (occasionally replaced by a mean value extension) of the objective over $\boldsymbol{x}$ provides the required lower bound. Such a simple interval evaluation is what is currently implemented in GlobSol.

However, since such an evaluation does not take account of the constraints, it can have an enormous overestimation. As an example, consider the nonlinear minimax problem:

$$\min_x \max_{1\le i\le m} |f_i(x)|, \quad f_i : \mathbb{R}^n \to \mathbb{R}, \quad x \in \mathbb{R}^n, \quad m \ge n. \tag{4}$$

To date, we have had limited success in solving realistic problems of this type directly using GlobSol's non-smooth slope extensions. Alternately, we can convert the problem to a to a smooth problem with Lemaréchal's technique [29] as follows:

$$\begin{array}{l} \min_{x\in\mathbb{R}^n} v \\ \text{such that } \left\{ \begin{array}{r} f_i(x) \le v \\ -f_i(x) \le v \end{array} \right\}, \quad 1 \le i \le m. \end{array} \tag{5}$$

In (5), we have introduced a single additional slack variable v, which becomes the value of the objective function. If v is treated as an arbitrary additional independent variable, then GlobSol presently employs constraint propagation to narrow the range of v when a subset of the region for the variables x is given. However, this process does not take account of the coupling between the constraints, and has not enabled GlobSol to solve minimax problems efficiently. Furthermore, interval Newton methods applied to the Lagrange multiplier (or Fritz–John) system associated with (5) over large regions have not adequately accelerated the search process within GlobSol for realistic minimax problems.

In contrast, Floudas [10], Sahinidis [47] and their respective groups have used convex or linear relaxations of problem (2) to significant advantage in non-verified global optimization software. For example, to obtain a lower bound on an objective function over a region $\boldsymbol{x}$, the objective ϕ in problem (2) is replaced by a convex (or linear) objective that is known to be less than or equal to

the actual objective over $\boldsymbol{x}$. Each left member g_i of the inequality constraints is similarly replaced by a convex (or linear) underestimator. Likewise, each equality constraint $c_i(x) = 0$ is replaced by the two inequality constraints $c_i(x) \leq 0$ and $-c_i(x) \leq 0$, and then underestimated. The optimum of the resulting convex (or linear) program then is less than or equal to the global optimum of the original problem (2).

Experimenting with Sahinidis' BARON [44] software, we have been able to successfully find global optima of minimax problems of the form (4). Apparently, the reasons these techniques are successful where the others are not are because

1. they take account of the coupling between the constraints, and
2. the resulting relaxations (i.e. the derived simpler problems) have solutions, and these solutions are easy to obtain.

The computations with convex linear underestimators can be made rigorous with the following procedure:

1. Compute a relaxed (simplified) convex or linear problem over $\boldsymbol{x}$.
2. Compute the solution to the convex or linear problem with a floating-point solver that gives an approximate solution $\check{x}$.
3. Use $\check{x}$ with the validation technique in [13] to provide a rigorous lower bound on the solution to the relaxed (and hence on the solution to the original) problem over $\boldsymbol{x}$.

We are presently experimenting with this procedure, and will eventually incorporate it into GlobSol for minimax procedures.

Although each group develops different techniques, both Floudas [10] and Sahinidis [47] develop methods by which these underestimators can be computed automatically (with automatic-differentiation-like technology; see [10] and [47]); such techniques (and others) could eventually be incorporated into GlobSol.

Efficiency of GlobSol's Automatic Differentiation, List Processing, etc. As outlined in [18, §1.4 and §2.2], GlobSol interprets an internal representation of the objective and constraints, termed a "code list", to compute point and interval values of the objective, constraint residuals, Jacobi and Hessian matrices, etc. This internal representation was designed with simplicity in mind, under the assumption that problems GlobSol would solve are relatively small and would not be limited by inefficiencies in function evaluation. However, for a number of problems, evaluation of the code list could speed computation.

Experiments by Corliss et al. under the Sun project have indicated that, for some problems, converting the code list to Fortran code then compiling it gave a noticeable performance improvement, but did not make a difference in the practicality of solving particular problems. On the other hand, operations for evaluating every constraint and the objective are included in a single code list, and all of these operations are performed whenever a particular objective or constraint value is needed at a new point (or interval) of evaluation. Separating the operations could benefit particular problems.

Another area of possible efficiency gains in GlobSol is in its list processing. In the global search, regions $\boldsymbol{x}$ are repeatedly bisected into $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$; $\boldsymbol{x}^{(1)}$ is processed further, while $\boldsymbol{x}^{(2)}$ is stored in a linked list structure. Memory is allocated whenever a box is stored on the list, and is freed whenever a box is removed. For some problems, a more sophisticated allocation / deallocation scheme would greatly improve performance.

Although, with time, we intend to implement these GlobSol improvements, we do not place them at as high a priority as algorithmic improvements, such as use of convex underestimators. In our view, fundamental algorithmic improvements will advance both the practicality of GlobSol and the fundamental state of the art in verified global optimization more.

4.5 Simplification of GlobSol

At present, there are many optional algorithm paths in GlobSol, some of which are not used. This is a result of the original research nature of GlobSol. Eventually, some of these paths (along with supporting code) can be eliminated.

Other improvements in this general category include updating GlobSol's installation scripts.

4.6 Summary

In this section, we have described GlobSol, validated global optimization software for Fortran. GlobSol represents a little over a decade of work on algorithms and implementations. GlobSol is unusual among such packages in being openly available and self-contained. Although GlobSol has weaknesses for certain kinds of constrained problems, we are excited about alternate algorithms, as yet untried in a validated context, that promise to remove many of these weaknesses.

5 ACETAF (Markus Neher)

5.1 Introduction and Overview

S. Oishi and S. M. Rump have developed a new verification method called *rounding mode controlled verification*, and have applied this method to simultaneous linear equations [41]. It has been shown in [41] that the total cost of calculating an approximate solution of a system of n-dimensional simultaneous linear equations and of calculating a rigorous error bound is $4/3\ n^3$ flops. Let us consider a computing system which conforms to the IEEE 754 floating point standard. Let A and B be $n \times n$ matrices whose elements are IEEE 754 double precision numbers. Then, we have shown [41] that an inclusion of a product of A and B can be calculated by

```
setround(down);
L = A * B;
setround(up);
U = A * B;
```