Combining Numerical Analysis and Constraint Processing by Means of Controlled Propagation and Redundant Constraints

M.H. van Emden

Department of Computer Science,
University of Victoria, Victoria, Canada
vanemden@cs.uvic.ca
http://www.cs.uvic.ca/~vanemden/

Abstract. In principle, interval constraints provide tight enclosures for the solutions of several types of numerical problem. These include constrained global optimization and the solution of nonlinear systems of equalities or inequalities. Interval constraints finds these enclosures by a combination of propagation and search. The challenge is to extend the "in principle" to problems of practical interest. In this paper we describe the concept of *controlled propagation*. It uses this in conjunction with redundant constraints to combine numerical analysis algorithms with constraint processing. The resulting combination retains the enclosure property of constraint processing in spite of rounding errors. We apply this technique in an algorithm for solving linear algebraic equations that initially simulates interval Gaussian elimination and then proceeds to refine the result with propagation and splitting. Application of our approach to nonlinear equations yields an algorithm with a similar relation to Newton's method.

1 Introduction

It is not unusual for mathematical models in engineering or management science to have both discrete and continuous variables and both linear and nonlinear constraints. Such is the nature of the problems that arise when we abstract from a real-world situation using physical and probabilistic theories as well as counting arguments. However, the treatment methods acceptable to practising engineers and operations analysts are classified as belonging to nonlinear global optimization (all variables continuous), linear programming (all variables continuous, all constraints linear), or combinatorial optimization (all variables nonnumeric, so that the distinction between linear and nonlinear constraints does not arise).

As a result of being forced into these disjoint categories, the desired mathematical model undergoes considerable change before it can be implemented on a computer to yield quantitative results. The most problematic of these changes is *linearization*. It causes an original model in a modest number of variables to be translated into a linear one in a much larger number of variables. One is never

sure whether the steps used in linearization are small enough, so that computers are never big enough.

This situation has existed for decades. In recent years, constraint programming arose to promise interesting changes. These changes started with CHIP [5], which was restricted to discrete variables. CHIP and its successors made it possible to apply simultaneously techniques from combinatorics, integer linear programming, and boolean programming — hitherto mutually exclusive disciplines.

This approach has been successful in practice. It has also been recognized as a breakthrough in the theory of modeling discrete optimization problems; see the landmark book by J. Hooker [7]. This paper is based on the premise that what Hooker describes for discrete optimization problems can also be done for their continuous counterparts. In this way scientific computation will be transformed in the same way as operations research is transformed in Hooker's book.

The basis for such an advance already exists in the form of interval constraint systems. These are constraint systems in the sense of CHIP. The difference is that variables are continuous. CHIP relies on an efficient way to represent and manipulate finite domains of values for variables. Interval constraints needs a corresponding method for sets of reals. This need is met by interval arithmetic. The generality of constraint processing allows the use of a constraint that requires a real-valued variable to be integer. That this is an effective way of solving discrete problems has been shown by Older and Benhamou [**?**].

Thus it becomes possible to solve problems with both continuous and discrete variables and with both linear and nonlinear constraints. In this way, constraint processing does not force unwarranted simplification while assuring that no solution is lost. Moreover, each solution can be located to a precision only limited by the precision of the underlying hardware.

However, the methods of constraint processing, propagation and splitting, yield very poor performance by themselves. CHIP and its descendants overcame this disadvantage by harnessing specialized algorithms in combinatorial optimization and integer linear programming to the constraint-processing framework. In this way all of the advantages were combined: modelling flexibility, the enclosure property of constraint processing, and performance approaching that of the conventional methods. This point of view has been elaborated in [7]. *It is the aim of this paper to survey what has been done, and introduce what can done, to ensure that numerical analysis can be harnessed to the constraint-processing framework in the same way.* Accordingly, we will first describe (section 2) the work of Hansen, Sengupta, and Walster (summarized in [6]) in using intervals for nonlinear global optimization. The way they use classical, noninterval numerical methods to speed up the inefficient but sound Moore-Skelboe algorithm will serve as a model for how classical numerical methods can be used in interval constraints.

In section 3 we give a brief survey of constraint processing: just enough to formulate main result on the convergence of fair sequences in propagation. We

extend this by what we call the *prefix theorem*, the basis for integrating numerical analysis into interval constraint processing.

With this basis we describe in successive sections applications of the integrated approach to solving linear equations and solving nonlinear equations.

2 Enhancements of the Moore-Skelboe algorithm

Originally, the role of interval arithmetic was to detect situations where rounding errors are likely to cast doubt upon the result of a numerical computation. Later it was found that interval arithmetic has a more important role to play: it can solve problems that numerical analysis cannot. An example is that of finding the global minimum of a function with an unknown number of local minima. This can be done by the Moore-Skelboe algorithm. However, it is very inefficient. It performs an exhaustive search that finds the global optimum without using any of the techniques of numerical analysis for finding local optima. Without compromising the advantages of the Moore-Skelboe algorithm, Hansen and Walster extended it by adapting the techniques of numerical analysis. We describe their method briefly.

The Moore-Skelboe algorithm subdivides n-space into boxes and computes an interval for the objective function for each box. The bounds of these intervals are used to eliminate, according to the branch-and-bound principle, regions that do not contain the global optimum.

In conventional numerical analysis one uses a necessary condition for a local optimum: that the gradient vanishes. Thus one solves the nonlinear vector equation that expresses this condition. Hansen, Sengupta, and Walster [6] extend the Moore-Skelboe algorithm by using interval arithmetic to determine, for each box arising in the branch-and-bound search, whether it is possible to contain any point at which the gradient vanishes. They use the interval Newton operator [10, 6]. In this way many more boxes are typically eliminated in the search than in the original Moore-Skelboe algorithm. In the worst case, there is no positive effect. This in contrast to the classical numerical methods where there are severe convergence problems in solving nonlinear vector equations.

Not all points with zero gradient are local minima. In numerical analysis this uncertainty is decided by using another necessary condition: that the matrix of second derivatives of the objective function be positive definite at the point with zero gradient. Hansen and Walster use this condition indirectly to obtain a further enhancement of the Moore-Skelboe algorithm: an easily computable necessary condition for the positive definiteness is that the trace of the matrix of second derivatives be positive. This possibility can sometimes be excluded by evaluating the trace in interval arithmetic. Thus a necessary condition for a necessary condition is used to further prune the search space.

In constrained global optimization, the Kuhn-Tucker and the John conditions are necessary conditions for minima. Hansen *et al.* also studied this and used these necessary conditions in the same way as the ones mentioned above. For further developments in this direction, see Kearfott [8]. When translating

from interval arithmetic to interval constraints, the same opportunities exist for pruning the search space.

The reason for reviewing this work is to distill from it the principle of *pruning by redundant constraints*:

> *One starts with a correct, complete and universally applicable solving method based on search. One finds that it is too inefficient for most practically interesting problems. One adapts a technique of numerical analysis that is based on a* necessary condition *for solutions. By adding this condition as a* redundant constraint*, one reduces the search space sufficiently to increase the size of problem that can be handled without losing the correctness properties of the original method.*

In the next section we review constraint processing to show how this principle can be applied.

3 Constraint processing

In constraint processing one can distinguish *point methods* from *set methods*. In the former, a single value is associated with each variable. Each constraint comes with a rule to adjust a variable's value using the values of the other variables. This method originated with Southwell's relaxation method [11] for systems of linear algebraic equations. It was adopted by Sutherland in his Sketchpad system [13]. The approach was further developed by Steele [12] and Borning [4] and his school.

The point method has in common with numerical analysis that no certainty can be assigned to its results. The set method, on other hand, associates a *set of values* with each variable. It is a property of the contraction operators applied in the propagation algorithm that values are only removed from a set if they do not occur in any solution. As a result, it is a property of the solving algorithms employed in the set method that it is a logical consequence of the constraints that no solutions exist outside the Cartesian product of the sets associated with the variables.

In this section we review constraint systems, interval constraint systems, and their solution methods. Most of this is due to Benhamou and Older [3].

3.1 Constraint systems

To demonstrate the versatility of the approach, we first present an example of a non-numeric constraint system.

Example 1. Let us consider a constraint system representing a graph colouring problem. Now the constraints all have two arguments and have as meaning the complement of the equality relation. Suppose that for one of the constraints, the domains of the variables are $\{red\}$ and $\{red, yellow, blue\}$. Then the contraction operator for this constraint leaves the first unchanged and removes *red* from

the second. If the domains are initially $\{red, yellow\}$ and $\{red, yellow, blue\}$, then contraction operator leaves both domains unchanged.

We present the next example in a more mathematical, but informal style. Then we define constraint systems formally and use the example to illustrate the some of the features of the formal definition.

Example 2. Consider a system with set of constraints

$$\{p(x_1, x_2, x_1), s(x_3, x_1, x_4), g(x_3, x_5), p(x_5, x_5, x_1)\}.$$

The variables $x_1, \ldots, x_5$ are of types $\mathcal{R}$, $\mathcal{Z}$, $\mathcal{N}$, $\mathcal{R}$, and $\mathcal{Z}$ respectively. The meaning of p is given by $p(x, y, z)$ is true iff $x \times y = z$. Thus p names the ternary product relation. Similarly, s denotes the ternary sum relation and g the binary greater-than relation.

The constraint system describes a *constraint satisfaction problem* (CSP) which is to find a value for each variable of the corresponding type in such a way that all constraints are true.

Definition 1. *A constraint system has the following attributes.*
(1) A set $\{T_1, \ldots, T_k\}$ of sets called types.
(2) A set $\{x_1, \ldots, x_n\}$ of variables where $x_i \in T_j$ (" x_i is of type T_j") for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$.
A valuation *v is a map in $\{x_1, \ldots, x_n\} \to T_1 \cup \cdots \cup T_n$ such that for all $i \in \{1, \ldots, n\}$ we have $v(x_i) \in T_i$.*
(3) A state*, which is $D_1 \times \cdots \times D_n$, where, for $i \in \{1, \ldots, n\}$, $D_i \subseteq T_i$. We say that, in this state of the constraint system, D_i is the* domain *of x_i.*
(4) A set $\{A_1, \ldots, A_m\}$ of constraints where A_i is an atomic formula: a predicate symbol followed by a sequence of variables.
$\{x_1, \ldots, x_n\}$ is the set of all variables occurring in $\{A_1, \ldots, A_m\}$.
If A is an atomic formula and v a valuation, then A/v is the substitution of $v(x_i)$ for x_i in A, for all $i \in \{1, \ldots, n\}$.
The index sequence *s of a constraint A is the sequence of occurrences of variables in A. Thus $|s|$ is the arity of A and s maps $\{1, \ldots, |s|\}$ to $\{1, \ldots, n\}$ in such a way that $s(j) = k$ iff x_k is the j-th variable occurrence in A.*
The meaning ρ of the predicate symbol in A is a relation, which is a set of tuples of length $|s|$. The j-th element of such a tuple is an element of T_{s_j}. The relation ρ is defined by $\tau \in \rho$ iff A/v is true, where v is the valuation such that, for all $i \in \{1, \ldots, n\}$, $v(x_{s_i}) = \tau_i$.
The projection π_s associated with an index sequence s is such that $\pi_i(\pi_s(D_1 \times \cdots \times D_n)) = D_{s_i}$ for $i \in \{1, \ldots, n\}$.
The contraction operator associated with a constraint A with index sequence s maps a state D to a state D' in such a way that $D'_i = \pi_i(\rho \cap \pi_s(D))$ if x_i is a variable of A and $D'_i = D_i$ otherwise.
(5) An initial state*, which is a state.*

Example 3. To illustrate the definition, let us use it to formalize example 2. The index sequence of the first constraint listed is $s = [1, 2, 1]$. The relation of this constraint is p, which is a subset of $\mathcal{R} \times \mathcal{Z} \times \mathcal{R}$. The contraction operator of this constraint maps states D to D' such that $D'_1 = \pi_1(p \cap \pi_s(D))$, $D'_2 = \pi_2(p \cap \pi_s(D))$, $D_3 = D'_3$, $D_4 = D'_4$, and $D_5 = D'_5$.

For each constraint there has to be an efficiently computable contraction operator.

3.2 Interval constraint systems

In this paper we consider *interval constraint systems*, which are constraint systems where the types are all equal to $\mathcal{R}$ and where the domains are intervals. The constraints include $x \leq y$, $x = y$, $x + y = z$, $x \times y = z$, $x^n = y$, $\sin(x) = y$ and other trigonometric functions, $\exp(x) = y$, and $\log x = y$.

Definition 2 (Floating-point numbers, intervals). *A floating-point number is any element of $F \cup \{-\infty, +\infty\}$, there F is a finite set of reals that includes 0 and that is closed under negation. If x is a finite floating-point number, then x^- (x^+) is the greatest (least) floating-point number less (greater) than x. In addition, $-\infty^- = -\infty$, $-\infty^+ = -M$, $+\infty^- = M$, and $+\infty^+ = +\infty$ where M is the greatest finite floating-point number.*

A floating-point interval *is a closed connected set of reals, where the bounds, in so far as they exist, are floating-point numbers. When we write "interval" without qualification, we mean floating-point interval.*

An interval that does not properly contain an interval is called canonical.

A box *is a cartesian product of floating-point intervals.*

Thus canonical intervals are non-empty sets of reals. They may have positive width and they may have zero width. Examples are $[a^-, a]$, $[a, a^+]$, and $[a, a]$, where a is a finite float-point number. For any real, there is a unique smallest canonical floating-point interval containing it.

3.3 Numerical problems

A useful collection of numerical problems can be expressed as interval constraint systems. Consider the system of nonlinear inequalities in Figure 1. For simplicity of notation we made all the relations inequalities, even though the system might include $f(x_1, \ldots, x_n) \leq 0$ as well as $f(x_1, \ldots, x_n) \geq 0$, which can of course be written more succinctly. If the $f_1, \ldots, f_m$ are rational expressions where terms also include exponential logarithmic and trigonometrical functions, then such a system translates to an interval constraint system. Such a translation is necessary because in general the contraction operator for $f_k(x_1, \ldots, x_n) \leq 0$ is not efficiently computed. The translation of a system such as in Figure 1 with n and m in the order of a dozen typically results in an interval constraint system with hundreds of constraints in a similar number of variables.

$$
\begin{array}{l}
f_1(x_1\,, x_2, \ldots, x_n) \leq 0 \\
f_2(x_1\,, x_2, \ldots, x_n) \leq 0 \\
\ldots\ldots\ldots\ldots\ldots\ldots \\
f_m(x_1, x_2, \ldots, x_n) \leq 0
\end{array}
$$

Fig. 1. A system of non-linear inequalities. The left-hand sides are non-linear functions; the right-hand sides are constants.

Definition 3. *An* expression comparison *consists of two numerical expressions connected by one of the relations* $\leq$, $=$, *or* $\geq$. *An* expression comparison system *consists of a set* X *of real variables and a set of expression comparisons containing no variables other than those in* X.

In the conventional way, we consider expressions as trees. The leaf nodes are constants or variables; the nonleaf nodes are the operation symbols.

Definition 4. *The* relation associated with *a function* $f : \mathcal{R}^k \to \mathcal{R}$ *is*

$$\{\langle x_0, \ldots, x_k\rangle \mid x_0 = f(x_1, \ldots, x_k)\},$$

for $k = 0, 1, \ldots$

Definition 5. *The* constraint system $\mathcal{C}$ associated *with an expression comparison system* $\mathcal{E}$ *depends on a one-one correspondence between the non-leaf nodes of the trees in* $\mathcal{E}$ *and a set of variables that is disjoint from the variables in* $\mathcal{E}$ *and that is defined as follows.*

Each of the variables in $\mathcal{E}$ *also occurs in* $\mathcal{C}$, *where it is called a "primary variable". To each non-leaf node of a tree in* $\mathcal{E}$ *there corresponds a variable in* $\mathcal{C}$ *that does not occur in* $\mathcal{E}$ *and is called "auxiliary variable".*

The constraints in $\mathcal{C}$ *are determined as follows. For every non-leaf node* n *(which is an operation* f*), with children* $n_1, \ldots, n_k$, *of a tree in* $\mathcal{E}$, *there is an atomic formula in* $\mathcal{C}$ *with variables* $x_0, x_1, \ldots, x_k$. *The predicate in the atomic formula denotes the relation associated with* f. *This formula is called a* functional atom *in* $\mathcal{C}$. *The variables* $x_1, \ldots, x_k$ *are the* input variables *of the atom;* x_0 *is its* output variable.

In addition there is in $\mathcal{C}$, *for every expression comparison with trees* E_1 *and* E_2 *an atomic formula* $v_1 \leq v_2$, $v_1 = v_2$, *or* $v_1 \geq v_2$ *according to the relation in the expression comparison. This atomic formula is a* relational atom *in* $\mathcal{C}$.

3.4 Solving constraint systems

A constraint system is a problem to be solved: within the initial state, find values for the variables that make the constraints true. This is typically not possible if some of the types are the real numbers, as these are usually not representable as floating-point numbers. Interval constraint systems are therefore solved in

the following weak sense. By applying contraction operators in conjunction with the splitting of intervals, a set of states with canonical intervals as projections is found with the property that their union contains any solutions that the initial state may contain. For the example in Figure 1, this process may result in intervals for $f_1, \ldots, f_m$ with upper bounds less than $c_1, \ldots, c_m$. In that case solutions are known to exist. Otherwise, the precision of the system's arithmetic does not allow the existence of a solution to be decided.

Propagation In the initial state, applying a contraction operator may result in reducing the interval for one of the variables. If this is the case, then it is possible that application of the contraction operators of possibly existing other constraints in which this variable occurs will result in interval reductions, which, in turn, can be exploited by applying contraction operators. This suggests a process of *propagation* where contraction operators continue to be applied until none has any effect. It may also happen that a contraction operator generates an empty interval, which then proves that the interval constraint system is *inconsistent*, that is, has no solution.

If the interval constraint system is not proved inconsistent during propagation, a stable state is reached. If in this stable state an interval for, say, variable x is larger than desired, then the entire interval constraint system can be split in two. Each has as initial state the stable state, except that in one interval constraint system, x has as domain the left half of the stable state's interval, and in the other, x has the right half. After this, propagation can make further progress in reducing the domains in each of the newly created interval constraint systems, which can then be split after stabilization. Or it may show inconsistency. In this way propagation and splitting alternate until only interval constraint systems remain with states consisting of canonical intervals.

There are several propagation algorithms [1]. They have in common that they apply the contraction operators for the constraints in a *fair sequence* [14, 1]. The fairness property by itself is sufficient to ensure that the sequence converges to the unique least common fixpoint of all contraction operators [1]. We review the necessary definitions and results below.

It is remarkable that such a loose criterion as fairness is sufficient to imply convergence to the correct limit. This property is exploited by using a propagation algorithm that is extremely simple. Examples of this can be found in [1]. We are interested in controlling the order in which contraction operators are applied to a greater extent than is necessary for ensuring fairness. We use this additional control to speed up convergence; we call this *controlled propagation*.

As we show presently, controlled propagation can be used to speed up convergence by initially simulating a suitable numerical algorithm. First we need to show that propagation can be controlled for a more basic purpose: the evaluation of an expression according to interval arithmetic [2, 16].

Definition 6. *A* trace *for a constraint system $\mathcal{C}$ with attributes as in Definition 1 has the following components:*

(1) An index sequence *t, which is an infinite sequence with elements in $\{1, \ldots, m\}$. (Note that m is the number of constraints in $\mathcal{C}$.)*

(2) A sequence of contraction operators *of which the i-th element is τ_{t_i} defined by the atom A_{t_i} in $\mathcal{C}$, for $i = 0, 1, \ldots$*

(3) A sequence U of boxes of which the i-th element is the initial state of $\mathcal{C}$ if $i = 0$ and is $\tau_{t_{i-1}}(U_{i-1})$ if $i > 0$.

As we are primarily interested in the sequence of boxes, we think of the sequence of contraction operators as "activations" of the corresponding constraints. We think of the elements of t as "selecting" a constraint to be activated.

Definition 7. *A sequence is* fair *if every one of its elements occurs infinitely many times.*

The following proposition is based on the fact that the contraction operators are monotone nonincreasing and idempotent and that there are finitely many domains.

Proposition 1. *See [14, 1].*
For any interval constraint system with box B as initial state we have:
(1) The sequence of boxes has a limit for every trace.
(2) The greatest lower bound of these limits is also a limit of a trace.
(3) All traces in which the index sequence is fair have the same limit. This limit equals the greatest common fixpoint of $\tau_1, \ldots, \tau_m$ that is less than B.
(4) This fixpoint is uniquely determined by the constraint system. It is computed by a suitable instance of Apt's Generic Chaotic Iteration algorithm [1].

Definition 8. *A constraint system is* failed *(*non-failed*) if its fixpoint is empty (non-empty).*

A failed constraint system has no solutions. A non-failed constraint system may, but need not, have solutions.

Proposition 2. *The fixpoint of a constraint system contains all its solutions.*

This follows from the fact that the fixpoint is obtained by only applying contraction operators, each of which only removes values that cannot occur in any solution.

Definition 9. *A segment of a trace is* functional *if*

(1) The sequence of atoms only contains functional atoms.

(2) For every atom, any input variable that is an auxiliary variable has occurred as output variable earlier in the segment.

(3) No atom occurs more than once.

The following proposition shows that a trace of a constraint system can simulate the evaluation of an expression.

Proposition 3. *Let $\mathcal{C}$ be an interval constraint system with attributes as in Definition 1 that is associated with an expression-comparison system $\mathcal{E}$ containing an expression E. Let box B be the initial state of $\mathcal{C}$ such that all its projections corresponding to auxiliary variables are $[-\infty, +\infty]$. Let s be the sequence of occurrences of variables in E. Let x_i be the variable in $\mathcal{C}$ that corresponds to the root of E. Let $\mathcal{T}$ be a functional initial segment of a trace.*

The i-th projection of the last of the sequence-of-boxes of $\mathcal{T}$ is the same interval as the one obtained when E is evaluated in interval arithmetic with $\pi_{s_j}(B)$ as the interval substituted for x_{s_j} in E, for all $j \in \{1, \ldots, |s|\}$.

In this paper we are interested in exploiting the fairness property by taking a sequence of contraction operators that mimics a conventional numerical algorithm and prefixing this sequence in front of a fair, but otherwise uncontrolled, propagation. We describe examples of this later. These are based on a theorem described next.

Proposition 4. *Let T be a fair trace of an interval constraint system with constraints $A_1, \ldots, A_m$. Let T' be the result of prefixing to T a finite sequence of elements of $\{1, \ldots, m\}$. Then T' is a fair trace.*

As T is fair, each of $\{1, \ldots, m\}$ occurs infinitely many times in T. Adding any of these does not change this property.

4 Linear Algebraic Equations

Let us consider the problem of solving for x a system $Ax = b$, with A an $n \times n$ matrix, x and b column n-vectors.

In this section we describe an algorithm in interval constraints that has the correctness property of constraint processing, yet is speeded up by the use of a modified version of Gaussian elimination. The description is structured into the following sequence of algorithms: LAE_0, Gaussian Elimination, LAE_1, and LAE_2.

Algorithm LAE_0 The first step of the algorithm is to translate $Ax = b$ to an interval constraint system. Not only the variables in the sense of numerical analysis (the components of x) become interval constraint variables, but also all coefficients of A and all components of b. The difference between these categories is that the components of x are associated with the interval $[-\infty, +\infty]$, whereas the others are associated with the canonical intervals containing the numerical value given in the statement of the problem.

After this step, the algorithm performs propagation steps alternating with splitting.

In principle, the splitting can be continued until boxes are as small as the precision of the hardware allows. Let us say that splitting has continued to the *canonical level*.

When propagation and splitting have continued as far as possible, all remaining boxes are canonical and have the property that they cannot be shown not to contain a solution within the precision of the machine. It seems plausible that no more information about the solution can be obtained by any algorithm, using the same basic hardware operations, than is obtained by LAE_0.

Gaussian Elimination (GE) Let us first briefly review the Gaussian Elimination algorithm for this problem. The steps of the algorithm derive from $Ax = b$ an equivalent system $Tx = b'$ where T has upper-triangular form. This latter system is then trivially solved by back substitution: it directly gives the value of x_n as $b'_n/T_{n,n}$; this value is then substituted into the $(n-1)$-th row; both x_n and x_{n-1} are then substituted in the next row above, and so on until all components of x are known.

In this simple form, the algorithm works in the most favourable situations. Less favourable situations can be rescued by *pivoting*: interchanges of rows and columns to avoid division by small numbers. When the matrix A is close to being singular, no pivoting strategy helps.

Gaussian Elimination can be tested by LAE_0: we check whether the x as found by GE is contained in any of the boxes remaining after completion of LAE_0. It usually is not, though often there will be one close by. This is the effect of rounding errors in GE.

Algorithm LAE_1 This is an algorithm in interval arithmetic. It follows the same steps as GE, except that every arithmetical operation is performed in interval arithmetic. As a result, all coefficients become intervals. The coefficients of $Ax = b$ are canonical intervals. Those of $Tx = b'$ are in general wider, and may become arbitrarily wide for sufficiently poorly conditioned A. The result is a box for x that contains the solution, if any.

Algorithm LAE_2 This is an algorithm on an interval constraint system. In the first stage, the same interval constraint system is created from $Ax = b$ as is done in LAE_0. Next, the equations of $Tx = b'$ in LAE_1 are added to the interval constraint system as redundant constraints. The resulting system is then subjected to controlled propagation, where the initial part of the sequence of constraint contractions simulates the back substitution of GE. In cases where GE works well, the box for x is small at the end of this stage. Propagation is then completed to the limit.

If A is ill-conditioned, or if LAE_1 does not use an adequate pivoting strategy, then the intervals at end of back substitution are wide. This will always be so for systems where GE fails with every pivoting strategy. In such cases LAE_1 gives an improvement by at least bounding a possibly existing solution. LAE_2 improves in such a situation by reducing those intervals. Though it may be too time-consuming to continue all the way to canonical intervals, any time available can be used by LAE_2 to improve the result that LAE_1 would give.

5 Nonlinear equations

For the sake of simplicity, we consider a single nonlinear equation in a single variable.

Let us try to repeat the success we had with Gaussian elimination and follow the same sequence of algorithms: raw interval constraints, the conventional numerical algorithm, the translation of same to interval arithmetic, and finally the simulation of the previous algorithm by controlled propagation prefixed to the raw interval constraints algorithm.

Algorithm NLE_0 Translate nonlinear equation to interval constraint system in the same way as was done for a system of linear equations: each constant is translated to a constraint system variable with a canonical interval and the variable that is to be solved for to a constraint system variable with interval $[-\infty, +\infty]$. The resulting interval constraint system is then subject to search consisting of alternating propagation and splitting. This is correct, gets all information available, but is very inefficient.

Newton's Algorithm In Gaussian elimination, each step derives a logical consequence of the current system of equations. This is also possible with a nonlinear equation, but this does not lead to an equation that can be handled in conventional numerical analysis. What is done instead is the derivation of the linearization of f at the starting point. This linearization, being linear, is then easily solved. As the linearization is not a logical consequence, and is in general false, the result of solving the linearization is in general not a solution. However, if the linearization takes place sufficiently close to the solution, then the result of solving the linearization is closer to the solution and is made the new starting point. The process is repeated.

Algorithm NLE_1 In Gaussian elimination, the soundness of the elimination step made a straight translation to interval arithmetic possible. The lack of soundness makes such a translation pointless in this case. Instead, Moore [9] derived a sound version of the Newton step in terms of interval arithmetic. This leads to an iterative process that has a superficial resemblance to conventional Newton. It achieves something that is not possible in conventional numerical analysis: to find enclosures for all solutions.

Algorithm NLE_2 The analogy with algorithm LAE_2 suggests that NLE_1 be simulated with controlled propagation as a prefix to the standard chaotic propagation. However, principle of pruning by redundant constraints requires adding redundant constraints to the interval constraint system in NLE_0. As the interval Newton step does not seem to have an interpretation as a logical consequence of the equation, it cannot play the role of a redundant constraint.

Such a redundant constraint has been tried in [15]. It is the first order version of Taylor's theorem:

$$f(x) = f(x_0) + (x - x_0)f'(\xi) \tag{1}$$

where

$$(x_0 \leq \xi \leq x) \vee (x \leq \xi \leq x_0). \tag{2}$$

This was added as a redundant constraint with x_0 equal to the midpoint of the interval for x. Propagation was then repeated, yielding a new interval for x, with a new midpoint and a new redundant constraint. This process was found to have quadratic convergence on a few examples with simple zeros. An analysis of the conditions under which this behaviour is found does not seem to have been made.

6 Conclusions

Classical algorithms of conventional numerical analysis can be used to speed up solving by interval constraint systems.

The fact that any fair sequence converges to the correct limit for the sets associated with the variables leaves a vast amount of freedom in choosing the order in which contraction operators can be applied.

In [2, 16] it has been shown that the evaluation of complex expressions can be simulated in this way by propagation. In this paper we describe how constraint propagation can also simulate the Gaussian elimination algorithm.

Another way in which numerical analysis can speed up solving by interval constraints is to use its algorithms as suggestions for useful redundant constraints to add. In the case of Gaussian elimination, exactly the same derived equation could be added redundantly to the interval constraint system. In the case of Newton's method one cannot add the linearized version of the equation, but instead one can add the instance of Taylor's that serves as basis of Newton's algorithm.

7 Acknowledgments

The ideas in this paper evolved from a conversation with Bill Older. Thanks to the referees for their suggestions. We acknowledge generous support from the Natural Science and Engineering Research Council NSERC.

References

1. K.R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999.
2. Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revising hull and box consistency. In *Proceedings of the 16th International Conference on Logic Programming*, pages 230–244. MIT Press, 1999.
3. Frédéric Benhamou and William J. Older. Applying interval arithmetic to real, integer, and Boolean constraints. *Journal of Logic Programming*, 32:1–24, 1997.
4. Alan Borning. ThingLab — a constraint-oriented simulation laboratory. Technical Report SSL-79-3, Xerox Palo Alto Research Center, 1979.

5. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint programming language CHIP. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, 1988.
6. Eldon Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, 1992.
7. J. Hooker. *Logic-Based Methods for Optimization - Combining Optimization and Constraint Satisfaction*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons, 2000.
8. R. Baker Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, 1996. Nonconvex Optimization and Its Applications.
9. Ramon E. Moore. *Interval Analysis*. Prentice-Hall, 1966.
10. Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.
11. R.V. Southwell. *Relaxation Methods in Engineering*. Oxford University Press, 1940.
12. Gerald Jay Sussman and Guy Lewis Steele Jr. Constraints — a language for expressing almost-hierarchical descriptions. *AI Journal*, 14:1–39, 1980.
13. I. Sutherland. *Sketchpad: a Man-Machine Graphical Communication System*. PhD thesis, Dept. of Electrical Engineering, MIT, 1963.
14. M.H. van Emden. Value constraints in the CLP Scheme. *Constraints*, 2:163–183, 1997.
15. M.H. van Emden. Algorithmic power from declarative use of redundant constraints. *Constraints*, pages 363–381, 1999.
16. M.H. van Emden. Computing functional and relational box consistency by structured propagation in atomic constraint systems. In *Proc. 6th Annual Workshop of the ERCIM Working Group on Constraints; downloadable from CoRR*, 2001.