On the Selection of a Transversal to Solve Nonlinear Systems with Interval Arithmetic

Frédéric Goualard and Christophe Jermann

LINA, FRE CNRS 2729 – University of Nantes – France
2, rue de la Houssinière – BP 92208 – F-44322 Nantes cedex 3
{Frederic.Goualard|Christophe.Jermann}@univ-nantes.fr

Abstract. This paper investigates the impact of the selection of a transversal on the speed of convergence of interval methods based on the nonlinear Gauss-Seidel scheme to solve nonlinear systems of equations. It is shown that, in a marked contrast with the linear case, such a selection does not speed up the computation in the general case; directions for researches on more flexible methods to select projections are then discussed.

1 Introduction

The extensions to interval arithmetic [10] of Newton and nonlinear Gauss-Seidel methods [13] do not suffer from lack of convergence or loss of solutions that cripple their floating-point counterparts, which makes them well suited to solve systems of highly nonlinear equations.

For the linear case, it is well known that reordering equations and variables to select a transversal is paramount to the speed of convergence of first-order iterative methods such as Gauss-Seidel [3, 5]. Transversals may also be computed [14, 7, 4] in the nonlinear case when using nonlinear Gauss-Seidel methods [12] and when solving the linear systems arising in Newton methods (e.g., preconditioned Newton-Gauss-Seidel, aka Hansen-Sengupta's method [6]).

Interval-based nonlinear Gauss-Seidel (INLGS) methods are of special importance because they constitute the basis for interval constraint algorithms [4] that often outperform extensions to intervals of numerical methods.

We show in this paper that, in the general case, it is not possible to choose statically at the beginning of the computation a good transversal when using an INLGS method. We also present evidences that reconsidering the choice of the transversals after each Gauss-Seidel outer iteration is potentially harmful since it may delay the splitting of domains when the INLGS method is floundering.

Section 2 gives some background on interval arithmetic and its use in the algorithm based on nonlinear Gauss-Seidel that is used in this paper; Section 3 describes previous works on the selection of a good transversal, and presents experimental evidences that such a choice may actually be baseless; Section 4 explores alternative ways to select a good set of projections by either choosing more than n projections for a system of n equations on n variables, or by reconsidering the choice of a transversal dynamically during the solving process;

Lastly, Section 5 delves into all the experimental facts presented so far to propose new directions of research for speeding up the solving of systems of nonlinear equations with INLGS-based algorithms.

2 An Interval Nonlinear Gauss-Seidel Method

Interval arithmetic [10] replaces floating-point numbers by closed connected sets of the form $\boldsymbol{I} = [\underline{\boldsymbol{I}}, \overline{\boldsymbol{I}}] = \{a \in \mathbb{R} \mid \underline{\boldsymbol{I}} \leqslant a \leqslant \overline{\boldsymbol{I}}\}$ from *the set* $\mathbb{I}$ *of intervals*, where $\underline{\boldsymbol{I}}$ and $\overline{\boldsymbol{I}}$ are floating-point numbers. In addition, each n-ary real function ϕ with domain $\mathcal{D}_\phi$ is extended to an interval function Φ with domain $\mathcal{D}_\Phi$ in such a way that the *containment principle* is verified:

$$\forall A \in \mathcal{D}_\phi \, \forall \boldsymbol{I} \in \mathcal{D}_\Phi \colon A \in \boldsymbol{I} \implies \phi(A) \in \Phi(\boldsymbol{I})$$

Example 1. The *natural interval extensions* of addition and multiplication are defined by:

$$\boldsymbol{I_1} + \boldsymbol{I_2} = [\underline{\boldsymbol{I_1}} + \underline{\boldsymbol{I_2}}, \overline{\boldsymbol{I_1}} + \overline{\boldsymbol{I_2}}]$$
$$\boldsymbol{I_1} \times \boldsymbol{I_2} = [\min(\underline{\boldsymbol{I_1}}\underline{\boldsymbol{I_2}}, \underline{\boldsymbol{I_1}}\overline{\boldsymbol{I_2}}, \overline{\boldsymbol{I_1}}\underline{\boldsymbol{I_2}}, \overline{\boldsymbol{I_1}\boldsymbol{I_2}}), \max(\underline{\boldsymbol{I_1}\boldsymbol{I_2}}, \underline{\boldsymbol{I_1}}\overline{\boldsymbol{I_2}}, \overline{\boldsymbol{I_1}}\underline{\boldsymbol{I_2}}, \overline{\boldsymbol{I_1}\boldsymbol{I_2}})]$$

Then, given the real function $f(x, y) = x \times x + y$, we may define its natural interval extension by $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \times \boldsymbol{x} + \boldsymbol{y}$, and we have that $\boldsymbol{f}([2, 3], [-1, 5]) = [3, 14]$.

Implementations of interval arithmetic use outward rounding to enlarge the domains computed so as not to violate the containment principle, should some bounds be unrepresentable with floating-point numbers [8].

Many numerical methods have been extended to interval arithmetic [11, 13]. Given a system of nonlinear equations of the form:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\ddots \\ f_n(x_1, \ldots, x_n) &= 0 \end{aligned} \tag{1}$$

and initial domains $\boldsymbol{I_1}, \ldots, \boldsymbol{I_n}$ for the variables, these methods are usually embedded into a *branch-and-prune* algorithm BaP that manages a set of boxes of domains to tighten. Starting from the initial box $\boldsymbol{D} = \boldsymbol{I_1} \times \cdots \times \boldsymbol{I_n}$, BaP applies a numerical method "prune" to tighten the domains in $\boldsymbol{D}$ around the solutions of System (1), and bisects the resulting box along one of its dimensions whose width is larger than some specified threshold ε. The BaP algorithm eventually returns a set of boxes whose largest dimension has a width smaller than ε and whose union contains all the solutions to Eq. (1)—note that the boxes returned may contain zero, one, or more than one solution.

The *interval nonlinear Gauss-Seidel method* is a possible implementation for prune. It considers the n *unary projections*:

$$\begin{aligned} \boldsymbol{f_1^{(1)}}(\boldsymbol{x_1}, \boldsymbol{I_2}, \ldots, \boldsymbol{I_n}) &= 0 \\ &\ddots \\ \boldsymbol{f_n^{(n)}}(\boldsymbol{I_1}, \ldots, \boldsymbol{I_{n-1}}, \boldsymbol{x_n}) &= 0 \end{aligned} \tag{2}$$

and uses any unidimensional root-finding method to tighten the domain of each variable $\boldsymbol{x_i}$ in turn. Unidimensional Newton leads to the *Gauss-Seidel-Newton method* [12], whose extension to intervals is the *Herbort-Ratz method* [7].

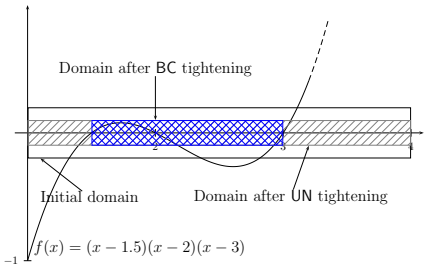

Fig. 1. Comparison of UN and BC

Let UN be the elementary step performed by one unidimensional Newton application to the projection $\boldsymbol{f_i^{(j)}}$, where $\boldsymbol{i}$ and $\boldsymbol{j}$ may be different [12]. As soon as $\boldsymbol{D}$ is moderately large, it is very likely that each projection constraint will have many "solutions" that are not solutions of the original real system, and whose discarding slows down the computation. The Newton method will also fail to narrow down the domain of some $\boldsymbol{x_i}$ if there is more than one solution to the corresponding projection constraint for the current box $\boldsymbol{D}$, thereby demanding more splittings in the BaP algorithm. Achieving the right balance between the amount of work required by the prune method and the number of splittings performed overall is the key to the maximum efficiency of BaP. In this very situation, experimental evidences show that trying harder to narrow down the domain of $\boldsymbol{x_i}$ pays off [2]. A way to do it is to ensure that the canonical intervals $[\underline{\boldsymbol{I_j}}, \underline{\boldsymbol{I_j}}^+]$ and $[\overline{\boldsymbol{I_j}}^-, \overline{\boldsymbol{I_j}}]$, whose bounds are two consecutive floating-point-numbers, are solutions of $\boldsymbol{f_i^{(j)}}(\boldsymbol{I_1}, \ldots, \boldsymbol{I_{j-1}}, \boldsymbol{x_j}, \boldsymbol{I_{j+1}}, \ldots, \boldsymbol{I_n}) = 0$. Let BC be an algorithm that ensures this property (called *box consistency [2] of* $\boldsymbol{x_j}$ *w.r.t. the constraint* $\boldsymbol{f_i} = 0$ *and* $\boldsymbol{D}$) for a projection $\boldsymbol{f_i^{(j)}}$. A simple method to implement it uses a dichotomic process to isolate the leftmost and rightmost solutions included in $\boldsymbol{D}$ of each projection constraint.

Example 2. Consider the constraint $f(x) = (x - 1.5)(x - 2)(x - 3) = 0$ and the domain $\boldsymbol{I} = [1, 4]$ for x (See Fig. 1). The UN method leaves $\boldsymbol{I}$ unchanged because the derivative of f over the initial domain contains 0 while BC narrows down $\boldsymbol{I}$ to $\boldsymbol{I'} = [1.5, 3]$, which is the smallest interval included in $\boldsymbol{I}$ that contains all the solutions to the interval constraint $\boldsymbol{f}(\boldsymbol{x}) = 0$.

3 Static Selection of a Transversal

When System (1) is linear, it is well known that one should reorder f_is and x_js such that the coefficient matrix becomes strictly diagonal dominant [12]. Many authors have noticed that nonlinear Gauss-Seidel (be it on intervals or not) is equally sensitive to such a reordering. When System (1) is nonlinear, one may exchange rows and columns in its *incidence matrix*[1] so as to obtain a transversal

[1] The incidence matrix M associated to System (1) is the zero-one matrix where M_{ij} is 1 if and only if x_j occurs in f_i.

of n pairs (f_i, x_i) corresponding to the unary projections in System (2) that will hopefully maximize the convergence rate of INLGS.

Several authors have attempted to compute good transversals for nonlinear problems:

- Sotiropoulos *et al.* [14] select a transversal for a polynomial system at the beginning of the computation by looking at the syntactic structure of the equations (variables with the largest degree in the system, ...), and by using numerical considerations only to break ties. In their paper, the static transversal is used in an interval Newton-Gauss-Seidel algorithm;
- Herbort and Ratz [7] compute the Jacobian $\boldsymbol{J}$ of System (1) w.r.t. the initial box $\boldsymbol{D}$ and select projections according to whether the corresponding entry in the Jacobian straddles zero or not. Their method is not completely static since they recompute the Jacobian after each iteration of INLGS (the choice of projections is not completely reconsidered, though). In addition, it theoretically allows for the choice of more than n projections;
- Goualard [4] determines an $n \times n$ matrix of weights W from the Jacobian $\boldsymbol{J}$ of Eq. (1) w.r.t. the initial box $\boldsymbol{D}$ corresponding to the distance of each interval $\boldsymbol{J}_{ij}$ to zero. He then computes a maximum weight perfect matching in the bipartite weighted graph associated to W, which gives a set of n projections on which to apply INLGS.

Table 1. Selecting a transversal vs. using all projections

	Problems Name $(n, \#sols)$	**HH**	**HG**	**HS**	**HB**	**GSA**
1	Barton (5,1)	NA	881	—	881	**378**
2	Bronstein (3,4)	8593	6712	**4430**	6712	10204
3	Broyden-banded (100,1)	**4500**	**4500**	NA	**4500**	30780
4	Broyden-tridiag. (10,2)	**12714**	1192697	13334	13385	31917
5	Combustion (10,1)	39324	8711299	NA	69546	**2581**
6	Extended Crag-Levy (8,36)	61707	7532	—	7532	**6612**
7	Extended Powell (10,32)	272977	268485	—	267591	**25130**
8	Grapsa-Vrahatis (3,2)	NA	246755	299085	17880	**15978**
9	MAT (3,1)	9639	**8590**	—	8657	18744
10	Moré-Cosnard (100,1)	**1200**	**1200**	NA	**1200**	100000
11	Robot (8,16)	NA	36179	NA	36179	**8510**
12	Troesch (10,1)	**260**	**260**	—	**260**	728
13	Yamamura (5,5)	**690**	4984061	**690**	**690**	3450

Number of calls to BC *to find all solutions up to a precision of* 10^{-8}

We tested the above heuristics on thirteen classical problems [1, 9, 14]. Eight problems are polynomial, and five are not. The heuristics of Herbort and Ratz (**HH**), Goualard (**HG**), and Sotiropoulos *et al.* (**HS**) served to compute a

transversal of n projections used in an INLGS algorithm where the univariate root-finding method is BC. The initial box is the one published in the papers cited. Table 1 presents the overall number of calls to BC needed to compute a set of solution boxes with width less than 10^{-8}. The dash entries correspond to the cases where an heuristics is not applicable (non polynomial problems for **HS**). An "NA" entry signals that the problem could not be solved in less than 5 minutes on an AMD Athlon™ XP 2600+. Each problem is identified by its name followed by its dimension and its number of solutions between parentheses. Heuristics **HB** corresponds to the choice of a maximum weight perfect matching computed from a matrix W where W_{ij} is the *relative reduction*[2] performed by applying BC on $\boldsymbol{f_i^{(j)}}$ for the initial box. This heuristics serves as a benchmark of what could be the best choice of a transversal, assuming the efficiency of a projection does not vary during the solving process. Column **GSA** gives the results when using all possible projections (that is, at most n^2). The boldfaced entries correspond to the smallest number of calls to BC per problem.

Analysis of Table 1. Whatever the heuristics chosen, selecting statically (or semi-statically for **HH**) a transversal of n projections is better than using the at most n^2 projections (**GSA**) for only 7 problems out of 13. We have monitored the amount of reduction obtained by applying BC on each of the possible projections during the whole solving process for every problem[3]. Upon close study, it is possible to separate the problems into two categories: those for which exactly n projections reduce the domains much better than the others, and those for which no such dichotomy can be made. The results for **GSA** follow closely that division: if there indeed exists a good transversal, the heuristics usually fare better than **GSA** because they are likely to choose it; if there is no good transversal, **GSA** is better because it avoids selecting a bad set of n projections, and makes the most out of the capability of all the projections to tighten the domains. This is in line with the conclusions by Goualard [4]. For those problems having a good transversal (2, 3, 4, 9, 10, 12, 13), the results of the heuristics vary widely, and some problems cannot be solved in the allotted time. At this point, the only reason that comes to mind is that the heuristics did choose a bad transversal and were stuck with it since the choice is not reconsidered dynamically. We will see in the next section that another reason may explain this situation. The **HB** heuristics also appears a good choice for problems with a good transversal since it never flounders on the seven presented here. However, the fact that it is not always the best method shows that the first reduction of a projection does not measure its overall efficiency, i.e., the efficiency of a projection varies during the solving process. In addition, it is slow on problems without a transversal. On the other hand, we cannot rely on **GSA** to solve problems with a good transversal since it is too computationally expensive to handle n^2 projections

[2] The relative reduction is defined by $(\mathsf{w}(\boldsymbol{I_j^b})-\mathsf{w}(\boldsymbol{I_j^a}))/\mathsf{w}(\boldsymbol{I_j^b})$ where $\mathsf{w}(\boldsymbol{I_j^b})$ (resp. $\mathsf{w}(\boldsymbol{I_j^a})$) is the width of the domain of $\boldsymbol{x_j}$ before (resp. after) applying BC on $\boldsymbol{f_i^{(j)}}$.

[3] All the log files containing detailed statistics for the problems presented are available at `http://interval.constraint.free.fr/problem-statistics.tar.gz`

instead of just n of them, especially for problems with a dense incidence matrix (e.g., Moré-Cosnard).

The problems without a good transversal may have no projection that is consistently better than the others; alternatively, it is possible that there exists a set of n or more good projections whose composition varies with the domains of the variables during the solving process. The next section investigates this matter to find out whether it is possible to optimize the computation for all problems.

4 Beyond the Choice of a Static Transversal

In order to assess conclusively whether a dynamic recomputation of a transversal may lead to good performances for all problems, we have decided to consider the **HB** heuristics only. Obviously, the results obtained are then only a benchmark of the level of performances that is theoretically attainable, since the **HB** heuristics works as an oracle able to tell us in advance what are the best reductions possible at some point in the computation.

Table 2 presents the results of our tests:

- $\mathbf{HB}_1^n$ is the **HB** heuristics presented in the previous section;
- $\mathbf{HB}_\infty^n$ corresponds to the dynamic recomputation of a transversal in the same way as **HB** after each outer iteration of INLGS;
- $\mathbf{HB}_1^{n+k}$ statically selects the best projection per variable (n) and then the best projection for each equation not already covered in the first selection ($0 \leqslant k \leqslant n-1$). It removes the *transversality constraint* that may yield suboptimal projections, avoiding also the use of costly matching algorithms;
- $\mathbf{HB}_\infty^{n+k}$ recomputes dynamically a set of $n+k$ projections according to $\mathbf{HB}_1^{n+k}$ after each outer iteration of INLGS.

Analysis of Table 2. **GSA** is no longer the best method on any of the problems, leading to the conclusion that it is always worthwhile to consider subsets of good projections. The dynamic heuristics ($\mathbf{HB}_\infty^n$ and $\mathbf{HB}_\infty^{n+k}$) lead to the least number of calls to BC for the majority of the problems, which is not so surprising since they guarantee to know in advance for the current iteration the projections for which BC will tighten variables domains the most. The fact that $\mathbf{HB}_\infty^n$ is slightly better than $\mathbf{HB}_\infty^{n+k}$ is an evidence that most problems that do not have a static transversal may still have a dynamic one. Note, however, that the transversality constraint might lead to choose some of the less effective projections. This may explain why $\mathbf{HB}_\infty^{n+k}$ is better than $\mathbf{HB}_\infty^n$ on Problems 6, 9, and 11. While dynamic heuristics are often better, they sometimes lead to poorer performances (e.g., for Problems 4 and 13). A close look at the computation logs for these problems reveals that the intervals of time between two bisections in $\mathbf{HB}_\infty^n$ and $\mathbf{HB}_\infty^{n+k}$ are larger than in $\mathbf{HB}_1^n$ and $\mathbf{HB}_1^{n+k}$, which is quite natural since the better choice of the projections leads to more reductions, and then to more iterations before the quiescence of INLGS and the necessity to split. This

Table 2. Dynamic and static selection of projections

Problems Name $(n, \#sols)$	$\mathbf{HB}_1^n$	$\mathbf{HB}_\infty^n$	$\mathbf{HB}_1^{n+k}$	$\mathbf{HB}_\infty^{n+k}$	**GSA**
1 Barton (5,1)	881	**140**	1470	299	378
2 Bronstein (3,4)	6712	**6011**	7961	6308	10204
3 Broyden-banded (100,1)	**4500**	**4500**	**4500**	6041	30780
4 Broyden-tridiag. (10,2)	**13385**	41019	**13385**	48941	31917
5 Combustion (10,1)	69546	**950**	47848	1309	2581
6 Extended Crag-Levy (8,36)	7532	7544	63947	**5252**	6612
7 Extended Powell (10,32)	267591	**12137**	415969	18412	25130
8 Grapsa-Vrahatis (3,2)	17880	**6575**	421945	9341	15978
9 MAT (3,1,5)	8657	8324	12554	**7589**	18744
10 Moré-Cosnard (100,1)	**1200**	**1200**	**1200**	1201	100000
11 Robot (8,16)	36179	5711	2219189	**3422**	8510
12 Troesch (10,1)	**260**	**260**	**260**	265	728
13 Yamamura (5,5)	**690**	710	**690**	931	3450

Number of calls to BC *to find all solutions up to a precision of* 10^{-8}

is good when the reductions are significant; on the other hand, such a behavior is undesirable when the reductions performed are small: it would then be much better to stop the iterations and bisect the domains altogether.

5 Conclusions

We have seen in Section 3 that heuristics that select a transversal only once at the beginning are doomed to fail on many problems for which such a static transversal does not exist. However, we have found in Section 4 that it is usually possible to isolate dynamically a set of good projections whose composition evolves during the solving process; as said before, the method we have used to find the elements of this set is only a benchmark of the optimizations attainable since it requires to find in advance what are the projections with which BC will tighten the domains of the variables the most. What is more, we may expect that the cost of recomputing a good transversal dynamically by any heuristics, however cheap, may more than offset the benefit of not having to consider n^2 projections for all problems but the ones with the densest incidence matrix.

There is still hope however: we have seen in Table 2 that a static heuristics like $\mathbf{HB}_1^n$ works as well as the best dynamic heuristics for the problems that have a good static transversal. A direction for future researches is then to identify beforehand whether a problem has such a good static transversal, and revert to **GSA** if it does not. More generally, it would be interesting to determine whether a projection will be always good, always bad, or of varying interest in order to use it in an appropriate way. Along these lines, our preliminary experiments suggest that Artificial Intelligence-based methods such as *reinforcement learning* approaches [15] used to solve the *Nonstationary Multi-Armed Bandit Problem*

are well-suited to tackle the dynamics behind the behavior of the projections during the solving process.

References

1. P. I. Barton. The equation oriented strategy for process flowsheeting. Dept. of Chemical Eng. MIT, Cambridge, MA, 2000.
2. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Procs. Intl. Symp. on Logic Prog.*, pages 124–138, Ithaca, NY, November 1994. The MIT Press.
3. I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Software*, 7(3):315–330, September 1981.
4. F. Goualard. On considering an interval constraint solving algorithm as a free-steering nonlinear gauss-seidel procedure. In *Procs. 20th Annual ACM Symp. on Applied Computing (Reliable Comp. and Applications track)*, volume 2, pages 1434–1438. Ass. for Comp. Machinery, Inc., March 2005.
5. A. J. Hughes Hallett and L. Piscitelli. Simple reordering techniques for expanding the convergence radius of first-order iterative techniques. *J. Econom. Dynam. Control*, 22:1319–1333, 1998.
6. E. R. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:203–211, 1981.
7. S. Herbort and D. Ratz. Improving the efficiency of a nonlinear-system-solver using a componentwise newton method. Research report 2/1997, Institut für Angewandte Mathematik, Universität Karslruhe (TH), 1997.
8. T. J. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic: from principles to implementation. *J. ACM*, 48(5):1038–1068, September 2001.
9. INRIA project COPRIN: Contraintes, OPtimisation, Résolution par INtervalles. The COPRIN examples page. Web page at `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`.
10. R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N. J., 1966.
11. A. Neumaier. *Interval methods for systems of equations*, volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1990.
12. J. M. Ortega and W. C. Rheinboldt. *Iterative solutions of nonlinear equations in several variables*. Academic Press Inc., 1970.
13. H. Ratschek and J. Rokne. Interval methods. In *Handbook of Global Optimization*, pages 751–828. Kluwer Academic, 1995.
14. D. G. Sotiropoulos, J. A. Nikas, and T. N. Grapsa. Improving the efficiency of a polynomial system solver via a reordering technique. In *Procs. 4th GRACM Congress on Computational Mechanics*, volume III, pages 970–976, 2002.
15. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.