Computing 53, 337–353 (1994)

Computing

Box-Splitting Strategies for the Interval Gauss-Seidel Step in a Global Optimization Method

D. Ratz, Karlsruhe

Abstract — Zusammenfassung

Box-Splitting Strategies for the Interval Gauss-Seidel Step in a Global Optimization Method. We consider an algorithm for computing verified enclosures for all global minimizers x^* and for the global minimum value $f^* = f(x^*)$ of a twice continuously differentiable function $f : I\!R^n \to I\!R$ within a box $[x] \in I I\!R^n$. Our algorithm incorporates the interval Gauss-Seidel step applied to the problem of finding the zeros of the gradient of f. Here, we have to deal with the gaps produced by the extended interval division. It is possible to use different box-splitting strategies for handling these gaps, producing different numbers of subboxes. We present results concerning the impact of these strategies on the interval Gauss-Seidel step and therefore on our global optimization method.

First, we give an overview of some of the techniques used in our algorithm, and we describe the modifications improving the efficiency of the interval Gauss-Seidel step by applying a special box-splitting strategy. Then, we have a look on special preconditioners for the Gauss-Seidel step, and we investigate the corresponding results for different splitting strategies. Test results for standard global optimization problems are discussed for different variants of our method in its portable PASCAL–XSC implementation. These results demonstrate that there are many cases in which the splitting strategy is more important for the efficiency of the algorithm than the use of preconditioners.

AMS Subject Classification: 65G10, 65K10, 65H20, 90C26.

Key words: Global optimization, interval arithmetic, Gauss-Seidel method, box-splitting strategies.

Box-Splitting-Strategien für den Intervall-Gauss-Seidel-Schritt in einem globalen Optimierungsverfahren. Wir betrachten einen Algorithmus zur Berechnung von verifizierten Einschließungen für alle globale Minimalstellen x^* und für den Wert des globalen Minimums $f^* = f(x^*)$ einer zweimal stetig differenzierbare Funktion $f : I\!R^n \to I\!R$ im Intervall $[x] \in I I\!R^n$. Unser Verfahren beinhaltet den Intervall-Gauss-Seidel-Schritt angewandt auf das entsprechende Nullstellenproblem für den Gradienten von f. Dabei ergibt sich die Aufgabe, die von der erweiterten Intervalldivision produzierten Lücken zu behandeln. Es ist möglich, verschiedene Box-Splitting-Strategien einzusetzen, die jeweils eine unterschiedliche Anzahl von Teilboxen erzeugen. Wir präsentieren Ergebnisse im Hinblick auf den Einfluß dieser Strategien auf den Intervall-Gauss-Seidel-Schritt und damit auf das globale Optimierungsverfahren.

Zunächst geben wir einen Überblick über einige der in unserem Algorithmus angewandten Techniken, und wir beschreiben die Modifikationen, die durch Anwendung einer speziellen Box-Splitting-Strategie, die Effizienz des Intervall-Gauss-Seidel-Schrittes verbessern. Dann betrachten wir spezielle Präkonditionierer für den Gauss-Seidel-Schritt, und wir untersuchen die entsprechenden Ergebnisse für unterschiedliche Splitting-Strategien. Testergebnisse für Standardaufgaben der globalen Optimierung werden diskutiert für unterschiedliche Varianten unserer Methode in ihrer portablen PASCAL–XSC Implementierung. Die Resultate zeigen, daß es viele Fälle gibt, in denen die Splitting-Strategie wichtiger für die Effizienz des Algorithmus ist als die Verwendung von Präkonditionierern.

1. Introduction

We consider the problem of finding the global minimizers of multi-dimensional nonlinear functions. Our algorithm is based on the method of Hansen [5]. The algorithm computes enclosures for all global minimizers and for the global minimum value of a twice continuously differentiable function in a given interval vector.

Classical numerical global optimization methods for the multi-dimensional case start from some approximate trial points and iterate. Thus, classical optimization methods sample the objective function at only a finite number of points. There is no way to guarantee that the function does not have some unexpectedly small values between these trial points.

Hansen's algorithm uses interval arithmetic to evaluate the objective function and its first- and second-order partial derivatives over a continuum of points, including those points that are not finitely representable on a computer. Interval analysis supplies the prerequisite for solving the global optimization problem with *automatic result verification*, i.e. with the guarantee that the global minimum points and the global minimum values have been found.

Let $f : I\!R^n \to I\!R$ be a twice continuously differentiable function, and let $[x] \in II\!R^n$. We address the problem of finding all points x^* in the interval vector $[x]$ such that

$$f(x^*) = \min_{x \in [x]} f(x). \tag{1}$$

We are interested in both the global minimizers x^* and the minimum value $f^* = f(x^*)$.

We use the branch-and-bound method of E. Hansen [5, 6] with the modifications described in [15], [16], and [4]. Our method

- starts from an initial box $[x] \in II\!R^n$,
- subdivides $[x]$ and stores the subboxes in a list, and
- discards subintervals which are guaranteed not to contain a global minimizer,

until the desired accuracy of the intervals in the list is achieved.

The power and speed of the method comes not so much from the ability to find the answer as from the ability to discard from consideration regions where the answer is not. The tests we use to discard pending subboxes are

- midpoint test,
- monotonicity test,
- concavity test, and
- interval Newton Gauss-Seidel step.

The midpoint test determines or improves an upper bound $\tilde{f}$ for f^* and discards all intervals from the list L for which $\tilde{f}$ is lower than the lower bound of the the corresponding interval function evaluation. The value $\tilde{f}$ is also used to check whether a newly subdivided interval is to be entered in the list L.

The monotonicity test determines whether the function f is strictly monotone in an entire subinterval $[y]$. If f is strictly monotone in $[y]$, then $[y]$ cannot contain a global minimizer in its interior. The concavity test (non-convexity test) examines the concavity of f. If f is not convex in a subinterval $[y]$, then $[y]$ cannot contain a global minimizer in its interior.

For details on these tests and on the method itself, see [4], [6], or [16]. The basic features for our method are extended interval arithmetic and differentiation arithmetic, where the latter is applied to compute the values of the derivatives of the objective function. An introduction to these features is given in [4].

The algorithm is implemented in PASCAL–XSC [12], a portable PASCAL compiler extension including interval arithmetic. Therefore, the tests presented in this paper produce identical results on the different platforms for the PASCAL–XSC system (e.g. PC, Workstation).

Given a list L, a list element E, a function $f : I\!R^n \to I\!R$, and an interval vector $[y] \in II\!R^n$, we use the following notation in our algorithms:

Notation	Meaning
$L := \{\,\}$	Initialization of L by an empty list
$L := L + E$	Enter element E in L according to condition (2)
$L := L - E$	Discard element E from L
$E :=$ Head (L)	Set E to the first element of L
∇f	Gradient of f
$\nabla^2 f$	Hessian matrix of f
$m([y])$	Midpoint of $[y]$
$\underline{f_y}$	Lower bound of the interval function evaluation $[f_y] := f([y])$
$f_\diamond()$	Machine interval evaluation of f

2. Global Optimization Algorithm

In the following, we give a simplified algorithmic description and an overview on our global optimization method.

Algorithm 1: GlobalOptimize $(f, [x], \varepsilon, L_{\text{res}}, [f^*])$

1. $[f_c] := f_\diamond(m([x]))$; $\tilde{f} := \overline{f_c}$; {Compute upper bound for f^*}
2. $[y] := [x]$; $L := \{\,\}$; $L_{\text{res}} := \{\,\}$;
3. **repeat** {Start iteration}
   (a) $k :=$ OptimalComponent $([y])$; Bisection $([y], k, [u_1], [u_2])$;
   (b) **for** $i := 1$ **to** 2 **do**
      i. $[f_u] := f([u]_i)$;
      ii. **if** $\tilde{f} < \underline{f_u}$ **then next**$_i$;
      iii. $[g] := \nabla f([u]_i)$;
      iv. **if** MonotonicityTest $([g])$ **then next**$_i$;

v. $[H] := \nabla^2 f([u]_i)$;
vi. **if** ConcavityTest $([H])$ **then next**$_i$;
vii. IntervalGaussSeidelStep $(f, [u]_i, [H], [V], p)$;
viii. **for** $j := 1$ **to** p **do**
A. $[f_V] := f([V]_j)$;
B. **if** $\tilde{f} < \underline{f_V}$ **then next**$_j$;
C. $[g] := \nabla f([V]_j)$;
D. **if** MonotonicityTest $([g])$ **then next**$_j$;
E. $L := L + ([V]_j, \underline{f_V})$; $\{\text{Store } [V]_j\}$

(c) $Bisect := false$;

(d) **while** $(L \neq \{\,\})$ **and** (**not** $Bisect$) **do**
i. $([y], \underline{f_y}) :=$ Head (L); $L := L - ([y], \underline{f_y})$;
ii. $\tilde{f} := \min\{\tilde{f}, f(m([y]))\}$; MidpointTest $(L, \tilde{f})$;
iii. **if** Accurate $(f([y]), [y], \varepsilon)$ **then**
$L_{\text{res}} := L_{\text{res}} + ([y], \underline{f_y})$;
else $Bisect := true$;

until (**not** $Bisect$);

4. $([y], \underline{f_y}) :=$ Head (L_{res}); $[f^*] := [\underline{f_y}, \tilde{f}]$;
5. **return** L_{res}, $[f^*]$;

Algorithm 1 manages the bisection of subboxes and their insertion in the pending list L. The subdivided boxes $[y]$ are stored as pairs $([y], \underline{f_y})$ in the list sorted in nondecreasing order of lower bounds $\underline{f_y}$. Therefore, a newly computed pair is stored in the list L according to the ordering rule (cf. [16]):

$$\left.\begin{array}{lll} \bullet \text{ either} & \underline{f_w} \le \underline{f_y} < \underline{f_z} \text{ holds,} \\ \bullet \text{ or} & \underline{f_y} < \underline{f_z} \text{ holds, and } ([y], \underline{f_y}) \text{ is the first element of the list,} \\ \bullet \text{ or} & \underline{f_w} \le \underline{f_y} \text{ holds, and } ([y], \underline{f_y}) \text{ is the last element of the list,} \\ \bullet \text{ or} & ([y], \underline{f_y}) \text{ is the only element of the list,} \end{array}\right\} \quad (2)$$

where $([w], \underline{f_w})$ is the predecessor and $([z], \underline{f_z})$ is the successor of $([y], \underline{f_y})$ in L. That is, the second components of the list elements may not decrease, and a new pair is entered behind all other pairs with the same second component.

In GlobalOptimize, we first compute an upper bound for the global minimum value, and we do some initializations. Step 3 is the main iteration. Here, we first do a bisection of the actual box $[y]$. Then in Step 3(b), we apply a function value check, the monotonicity test, the concavity test, and the interval Newton step to the bisected boxes $[u_1]$ and $[u_2]$. The interval Newton step results in p boxes. We have to handle them all in Step 3(b)viii, where we again apply a function value check and a monotonicity test. If the actual box

$[V]_j$ has not been discarded, then it is still a candidate for a minimizer, and we store it in L.

In Step 3(d), we remove the first element from the list L, i.e. the element of L with the smallest lower bound of the interval function evaluation, and we perform the midpoint test. Then, we check the tolerance criterion for the new actual interval. If the desired accuracy is achieved, we store this interval in the result list L_{res}. Otherwise, we go to the bisection step.

When the iteration stops because the pending list L is empty, we compute a final enclosure $[f^*]$ for the global minimum value in Step 4, and we return L_{res} and $[f^*]$.

The closer the upper bound $\tilde{f}$ is to the global minimum value f^*, the more intervals we can delete in the midpoint test (Step 3(d)ii of Algorithm 1). Thus, the method can be improved by incorporating an approximate local search procedure, to try to decrease the value $\tilde{f}$. See [6], [9], or [14] for the description of such local search procedures.

3. Interval Gauss-Seidel Step

In our global optimization method, we apply one step of the extended interval Newton Gauss-Seidel method (cf. [1] or [8]) to the nonlinear system

$$\nabla f(y) = 0, \quad y \in [y]. \tag{3}$$

The subbox $[y]$ is a candidate for containing a minimizer x^*, which we have assumed must satisfy $\nabla f(x^*) = 0$. When we apply the algorithm, three things may happen. First, we may validate that $[y]$ contains no stationary point, in which case we may discard $[y]$. Second, the Newton step may contract $[y]$ significantly. Subsequently, f can be evaluated on the narrower box $[y]$ with less overestimation, so the midpoint, monotonicity, and concavity tests are likely to be more effective. Third, we may get splittings of the box $[y]$ due to gaps produced by the extended interval divisions applied in the Newton step. We only apply one Newton step because this test is relatively expensive, and the other tests (with bisection) may subsequently discard even subboxes containing local minimizers. In addition, a stationary point is not necessarily even a local minimizer.

So, one step of the extended interval Newton Gauss-Seidel method shall improve an enclosure of the set of solutions y of

$$g = [H] \cdot (c - y)$$

where $c = m([y])$, $g = \nabla f(c)$, and $[H] = \nabla^2 f([y])$. This method works better if we first apply a *preconditioning*, by using a special matrix $R \in I\!R^{n \times n}$ for computing

$$b := R \cdot g \quad \text{and} \quad [A] := R \cdot [H],$$

and consider then

$$b = [A] \cdot (c - y).$$

Then, we compute $N'_{\mathrm{GS}}([y])$ according to

$$\left.\begin{aligned} [z] &:= [y] \\ [z]_i &:= \Big(c_i - \Big(b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} [A]_{ij} \cdot ([z]_j - c_j) \Big) \Big/ [A]_{ii} \Big) \cap [z]_i \\ & \qquad i = 1, \dots, n \\ N'_{\mathrm{GS}}([y]) &:= [z] \end{aligned}\right\} . \tag{4}$$

In the following theorem, we summarize the most important properties of the interval Newton Gauss-Seidel method.

Theorem 1 *Let $f : D \subseteq I\!R^n \to I\!R$ be a twice continuously differentiable function, and let $[x] \in I\,I\!R^n$ be an interval vector with $[x] \subseteq D$. Then $N'_{\mathrm{GS}}([x])$ has the following properties:*

1. *Every zero $x^* \in [x]$ of ∇f satisfies $x^* \in N'_{\mathrm{GS}}([x])$.*
2. *If $N'_{\mathrm{GS}}([x]) = \emptyset$, then there exists no zero of ∇f in $[x]$.*
3. *If $N'_{\mathrm{GS}}([x]) \overset{\circ}{\subset} [x]$, then there exists a unique zero of ∇f in $[x]$ and hence in $N'_{\mathrm{GS}}([x])$.*

For proofs, see [6] or [13].

In a practical realization of the interval Newton Gauss-Seidel method (4), it is not necessary to compute the $[y]_i$ in fixed order $i = 1, \dots, n$. A well-known strategy is the Hansen/Greenberg realization [7], which first performs the single component steps of the Gauss-Seidel step for all i with $0 \notin [A]_{ii}$ and then for the remaining indices with $0 \in [A]_{ii}$ by using extended interval arithmetic. In this case, a gap can be produced in the corresponding components $[y]_i$ of $[y]$. Therefore, $N'_{\mathrm{GS}}([x])$ may be given by one or more interval vectors. This leads directly to the question of splitting strategies.

4. Splitting Strategies

If $0 \in [A]_{ii}$ for several components i, then the extended interval divisions in the interval Newton Gauss-Seidel method possibly produces several gaps in the actual box $[y]$. Therefore, we have to split the result $N'_{\mathrm{GS}}([y])$ in two or more boxes. In this case, different splitting strategies may be applied. We give four examples:

1. *Use only the largest gap to split the box $[y]$.*
   This strategy is known from Hansen/Greenberg [7], and the Newton step results in at most 2 boxes.
2. *Use all gaps to split the box $[y]$ in a special way.*
   This strategy we suggested in [16], and the Newton step results in at most $n + 1$ boxes. We give the details in the following paragraph.
3. *Use at most three gaps to split the box $[y]$.*
   This strategy was suggested by Hansen [6], and the Newton step results in at most $2^3 = 8$ boxes.

4. *Use all gaps to split the box* $[y]$.
   As far as we know, nobody uses this strategy, because the Newton step results in at most 2^n boxes causing a proliferation of subboxes.

Let us now have a closer look on Strategy 2 suggested in [16] and also used in [4]. As already mentioned, this strategy uses *all* gaps in a special way. That means: if a gap is produced in the i-th component step, we store one part of the actual box $[y]$ by using one part of the component $[y]_i$ as i-th component of $[y]$. The other part of $[y]_i$ is used to update $[y]$ and to go on with the next component steps of the interval Gauss-Seidel method. That is, we perform one component step of the Gauss-Seidel step according to the scheme:

1. Compute $[y]_i = [v]_i \cup [w]_i$.
2. If $[v]_i = [w]_i = \emptyset$, then stop {no solution in $[y]$}.
3. If $[w]_i \neq \emptyset$, then set $[y]_i := [w]_i$ and store $[y]$.
4. Set $[y]_i := [v]_i$.
5. Continue with next i.

As an example, we now handle a box of dimension $n = 3$ assuming that the interval Gauss-Seidel step produces a gap in each component. Incorporating the above strategy, the three component steps of the method split the actual box $[y]$ in the following manner:

$$\begin{array}{ccccccc}
[y] = \begin{pmatrix} [y]_1 \\ [y]_2 \\ [y]_3 \end{pmatrix} & \xrightarrow{1} & \begin{pmatrix} [v]_1 \\ [y]_2 \\ [y]_3 \end{pmatrix} & \xrightarrow{2} & \begin{pmatrix} [v]_1 \\ [v]_2 \\ [y]_3 \end{pmatrix} & \xrightarrow{3} & \begin{pmatrix} [v]_1 \\ [v]_2 \\ [v]_3 \end{pmatrix} \\
\downarrow^{1} & & \downarrow^{2} & & \downarrow^{3} & & \\
\begin{pmatrix} [w]_1 \\ [y]_2 \\ [y]_3 \end{pmatrix} & & \begin{pmatrix} [v]_1 \\ [w]_2 \\ [y]_3 \end{pmatrix} & & \begin{pmatrix} [v]_1 \\ [v]_2 \\ [w]_3 \end{pmatrix} & &
\end{array}$$

Each step i is marked by a numbered arrow. The vertical arrows correspond to the storing of a subbox, whereas the horizontal arrows correspond to the updating of the actual box in the i-th component. Thus, the interval Gauss-Seidel step results in 4 boxes, i.e. the 3 boxes in the second row and the outmost right box in the first row of the graphic.

In the following, we give an algorithmic description of the interval Gauss-Seidel step incorporating this strategy.

Algorithm 2: IntervalGaussSeidelStep $(f, [y], [H], [V], p)$

1. Compute preconditioner R;
2. $c := m([y])$; $[A] := R \cdot [H]$; $[b] := R \cdot \nabla f_\diamond(c)$; $[y_c] := [y] - c$;
3. $p := 0$; {Initialization for loop}

4. **for** $i := 1$ **to** n **do** $\{$Interval Gauss-Seidel step for $0 \notin [A]_{ii}\}$
   (a) **if** $(0 \in [A]_{ii})$ **then next**$_i$;
   (b) $[y]_i := \Big(c_i - \Big([b]_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} [A]_{ij} \cdot [y_c]_j \Big) \Big/ [A]_{ii} \Big) \cap [y]_i$;
   (c) **if** $[y]_i = \emptyset$ **then return** $p := 0$;
   (d) $[y_c]_i := [y]_i - c_i$;
5. **for** $i := 1$ **to** n **do** $\{$Interval Gauss-Seidel step for $0 \in [A]_{ii}\}$
   (a) **if** $(0 \notin [A]_{ii})$ **then next**$_i$;
   (b) $[z] := \Big(c_i - \Big([b]_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} [A]_{ij} \cdot [y_c]_j \Big) \Big/ [A]_{ii} \Big) \cap [y]_i$; $\{[z] = [z_1] \cup [z_2]\}$
   (c) **if** $([z] = \emptyset)$ **then return** $p := 0$;
   (d) $[y]_i := [z_1]$; $[y_c]_i := [y]_i - c_i$;
   (e) **if** $([z_2] \neq \emptyset)$ **then** $\{$Store part of $[y]$ in $[V]_p\}$
   $p := p + 1$; $[V]_p := [y]$; $[V]_{pi} := [z_2]$;
6. $p := p + 1$; $[V]_p := [y]$;
7. **return** $[V], p$;

In the first steps, we compute a preconditioner R, the interval matrix $[A]$, and the interval vectors $[b]$ and $[y_c]$. We perform the single component steps of the Gauss-Seidel step for all i with $0 \notin [A]_{ii}$ (Step 4) and then for the remaining indices with $0 \in [A]_{ii}$ (Step 5). Using this strategy, it is possible that the intervals $[y]_i$ become smaller by the intersections with the old values $[y]_i$ in Step 4(b), before the first gap is produced in Step 5(b). If a splitting is necessary in Step 5, then we store one part of the actual box $[y]$ in the p-th row of the interval matrix $[V]$, and we continue the iteration for the other part of $[y]$. The variable p returns the number of subboxes produced in Algorithm 2. If an empty intersection occurs, the value $p = 0$ is returned. Thus, IntervalGaussSeidelStep returns the interval matrix $[V]$ containing row by row the p interval vectors (boxes) $[V]_i$, $i = 1, \ldots, p$, where $p \leq n + 1$.

5. Preconditioners

As already mentioned, the interval Gauss-Seidel method generally works better if we first apply a preconditioning. A preconditioner matrix R commonly recommended in the literature is the inverse of the midpoint of the interval matrix $[H]$. In Algorithm 2, we also use this kind of preconditioner, that is, we compute

$$R \approx (m([H]))^{-1}.$$

In our test results we refer to this case as "InvMid". For our tests, we also use Algorithm 2 *without* any preconditioning, i.e. with R set to the identity matrix I.

In [10], Kearfott introduced the concept of contracting and splitting preconditioners in the context of root-finding problems. In our special case of global optimization problems, we investigated a combination of two kinds of these preconditioners: *width optimal contracting preconditioners* and *pivoting splitting preconditioners* as suggested by Kearfott and Hu [11]. We give only a very brief description of these preconditioning techniques.

The width optimal preconditioners are part of a class of preconditioners which can be computed as solutions of linear programming problems [11]. A preconditioner row R_i minimizes the diameter of

$$b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} [A]_{ij} \cdot ([y]_j - c_j),$$

where $[A]_i = R_i \cdot [H]$ and $b_i = R_i \cdot g$.

Pivoting splitting preconditioners can be computed by simply solving $g = [H] \cdot (c - [y])$ in each row for each variable $[y]_i$ (cf. [11]). This is done by computing

for $m := 1$ **to** n **do**
  for $i := 1$ **to** n **do**

$$[y]_i := c_i - ([S]_{mi} + g_m) / [H]_{mi}$$

where

$$[S]_{mi} = \sum_{\substack{j=1 \\ j \neq i}}^{n} [H]_{mj} \cdot ([y]_j - c_j).$$

When computing the $[S]_{mi}$, we may use a special subtraction technique (cf. [11], [16]), which avoids interval dependencies and allows to compute all pivoting preconditioners for all variables in $O(n^2)$ operations.

An algorithmic scheme for the interval Gauss-Seidel step with special preconditioners looks as follows:

Algorithm 3: PreconGaussSeidelStep $(f, [y], [H], [V], p)$

1. $c := m([y])$; $g := \nabla f(c)$; $[z] := [y]$; $\delta := 10^{-6}$; $\tau := 0.8$.
2. SolveAllPivot $([y], [H], c, g, [V], p)$.
3. **if** $p = 0$ **then return** p.
4. **for** $i := 1$ **to** n **do**

   if $d([y]_i) \geq \delta |[y]_i|$ **and** $d([y]_i) \geq \tau d([z]_i)$ **then**

   (a) $R_i :=$ WOptimalCPrecon $([y], [H], i)$.
   (b) OneStepOfGaussSeidel $([y], [H], g, c, R_i, i, [V], p)$.
   (c) **if** $p = 0$ **then return** p.

5. **return** $[V], p$;

In Algorithm 3, we use the hybrid scheme and the values δ and τ suggested by [11]. At first, all pivoting preconditioners are used in SolveAllPivot , where splittings are produced with respect to all gaps. Depending on a diameter criterion, we then compute width optimal contracting preconditioners and do additional component steps of the Gauss-Seidel method. The variable p returns the number of subboxes produced. If an empty intersection occurs, the value $p = 0$ is returned. Due to the splitting with respect to all gaps and to the storing of parts of $[y]$ in the rows of the interval matrix $[V]$, PreconGaussSeidelStep returns the interval matrix $[V]$ containing row by row the p interval vectors (boxes) $[V]_i$, $i = 1, \ldots, p$, where $p \leq 2^n$.

6. Results

We use an implementation of Algorithm 1 including some minor modifications (cf. [16]) to compare the different versions of the interval Gauss-Seidel step. That is, in Step 3(b)vii either IntervalGaussSeidelStep (Algorithm 2) or PreconGaussSeidelStep (Algorithm 3) is called. In the following, we present results for problems which have been used previously in testing global optimization methods in [16].

In our tables, we distinguish the results with different splitting strategies and preconditioners by the maximum number of subboxes (splittings) which could be generated in the interval Gauss-Seidel step and by a shorthand description of the preconditioning used, respectively. Furthermore, we give informations on evaluation effort for the function, gradient, and Hessian values and on the maximum number of elements in our pending list. The total time for optimizing the corresponding test function is given in STUs, where the standard time unit STU is the computation time to evaluate the S5 test function (as usual real function) 1000 times. In detail, we use the following rows of information in the tables below:

Splittings	2	Use only one gap, Strategy 1
	$n+1$	Special use of all gaps, Strategy 2
	2^n	Use all gaps with splitting preconditioners
Preconditioning	None	No preconditioner, $R = I$
	InvMid	Inverse midpoint preconditioner, $R \approx (m([H]))^{-1}$
	WOptC/Piv	Optimal preconditioners, Algorithm 3
FE	Number of function evaluations	
GE	Number of gradient evaluations	
HE	Number of Hessian evaluations	
E_effort_1	Evaluation effort $\mathrm{FE} + n \cdot \mathrm{GE} + \frac{n\cdot(n+1)}{2} \cdot \mathrm{HE}$	
E_effort_2	Evaluation effort $\mathrm{FE} + \min\{4, n\} \cdot \mathrm{GE} + n \cdot \mathrm{HE}$	
L length	Maximum number of list elements	
STU	Standard time units	

We introduce the E_effort values which combine the three values FE, GE, and HE to a single value approximating the total evaluation effort in terms of objective function evaluations. In this way, we are able to compare roughly the evaluation efforts of our different variants of the Gauss-Seidel step by the E_effort values. We give two E_effort values to take into account the two methods of automatic differentiation.

According to [3], we have $W(f, \nabla f, \nabla^2 f) \,/\, W(f) \leq 7n^2 + 3n + 1$ for the forward mode (1) and $W(f, \nabla f, \nabla^2 f) \,/\, W(f) \leq 11n + 5$ for the reverse or backward mode (2) of automatic differentiation. These lead to the upper bounds $\mathrm{FE} + 3n\mathrm{GE} + 7n^2\mathrm{HE}$ in mode (1) and $\mathrm{FE} + 4\mathrm{GE} + 11n\mathrm{HE}$ in mode (2), respectively. The formulas for E_effort_1 and E_effort_2, described above, approximate the evaluation efforts for the average case.

For details and references concerning the used test functions, see [16].

As a first example, we compare Strategies 1 and 2 in minimizing Griewank's function (**G5**) where $x \in I\!R^5$ and

$$f_{\mathrm{G5}}(x) = \sum_{i=1}^{5} \frac{x_i^2}{400} - \prod_{i=1}^{5} \cos(\frac{x_i}{\sqrt{i}}) + 1,$$

within the initial box specified by $-500 \leq x_i \leq 600$, $i = 1, \ldots, 5$.

G5		
Splittings	2	$n+1$
FE	62	247
GE	404	140
HE	96	44
E_effort_1	3522	1607
E_effort_2	2158	1027
L length	68	87
STU	13.5	7.1

We see, that the use of our special splitting strategy improves the performance of the global optimization method, by drastically decreasing the number of Hessian and gradient evaluations.

Very interesting results come from testing our method with the three functions of Shekel (**S5, S7, S10**), where $x \in I\!R^4$ and

$$f_{\mathrm{S}m}(x) = -\sum_{i=1}^{m} \frac{1}{(x - A_i)(x - A_i)^T + c_i},$$

for $m = 5$, $m = 7$, and $m = 10$, with

$$A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}.$$

The initial box was specified by $0 \le x_i \le 10$, $i = 1, \ldots, 4$.

S5					
Splittings	2	$n+1$	2	$n+1$	2^n
Preconditioning	None	None	InvMid	InvMid	WOptC/Piv
FE	25	95	28	99	316
GE	132	68	145	70	73
HE	29	19	31	19	19
E_effort_1	843	557	918	569	798
E_effort_2	669	443	732	455	684
L length	18	38	18	38	53
STU	1.8	1.1	2.0	1.2	1.4

S7					
Splittings	2	$n+1$	2	$n+1$	2^n
Preconditioning	None	None	InvMid	InvMid	WOptC/Piv
FE	61	84	80	87	305
GE	298	68	366	70	72
HE	60	20	70	20	19
E_effort_1	1853	556	2244	567	783
E_effort_2	1493	436	1824	447	669
L length	38	28	44	28	44
STU	5.0	1.4	6.4	1.7	2.0

S10					
Splittings	2	$n+1$	2	$n+1$	2^n
Preconditioning	None	None	InvMid	InvMid	WOptC/Piv
FE	64	99	84	104	338
GE	346	72	413	76	80
HE	74	21	83	21	20
E_effort_1	2188	597	2566	618	858
E_effort_2	1744	471	2068	492	738
L length	66	31	67	31	42
STU	8.2	2.2	10.2	2.4	3.1

These test results show clearly, that our special splitting strategy is more important for the efficiency of the algorithm than the use of preconditioners. Especially, we see that we get the best performance if we use our special splitting strategy *without* any preconditioner in the Gauss-Seidel step. Kearfott's optimal linear programming preconditioners work much better than the inverse midpoint preconditioner with the standard splitting strategy (Strategy 1). Another interesting result is given by the increasing evaluation effort of the Sm-functions for increasing values of m. If the standard splitting strategy is used, then the evaluation effort increases drastically. If our special technique is used, we have only a slightly increasing evaluation effort.

Our next example is minimizing Hartman's function of dimension 6 (**H6**), where $x \in I\!R^6$ and

$$f_{\mathrm{H6}}(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right).$$

with

$$A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}, \quad \text{and}$$

$$P = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}.$$

The initial box is given by $0 \le x_i \le 1$, $i = 1, \ldots, 6$.

H6					
Splittings	2	$n+1$	2	$n+1$	2^n
Preconditioning	None	None	InvMid	InvMid	WOptC/Piv
FE	4607	1491	22003	2229	10229
GE	18682	1087	81264	1613	3184
HE	3269	223	13334	289	379
E_effort_1	185348	12696	776267	17976	37292
E_effort_2	98948	7177	427063	10415	25539
L length	1652	274	5818	386	662
STU	772	57.3	3641	86.6	199

Again, we see that our special splitting strategy improves the efficiency of the algorithm, and we get the best performance if we use it without preconditioner. The inverse midpoint preconditioning increases the evaluation effort by a factor of approximately 4.2 if standard splitting (Strategy 1) is used. If we use our special splitting (Strategy 2), the factor is only 1.4.

In the following, we compare some results of our new method including Strategy 2 (marked by "New") with the results presented in [6] (marked by "Han"). For the test functions f_{W4}, f_{W10}, and f_{W29} we choose $\varepsilon = 10^{-12}$ in our method.

Function **W4** is defined by

$$\begin{aligned} f_{\mathrm{W4}}(x) &= \sum_{i=1}^{5} i\cos((i-1)x_1+i)\sum_{j=1}^{5} j\cos((j+1)x_2+j) \\ &\quad +(x_1+1.42513)^2+(x_2+0.80032)^2, \end{aligned}$$

and the initial box is $-10 \le x_i \le 10$, $i = 1, 2$. We have 760 local minima in the initial box.

Function **W10** is defined by

$$\begin{aligned} f_{\mathrm{W10}}(x) &= \sum_{i=1}^{9} y_i^2(1+10\sin^2(\pi(1+y_{i+1}))) + \sin^2(\pi(1+y_1)) + y_{10}^2, \\ &\quad \text{where } y_i = (x_i-1)/4,\ i = 1,\dots,10, \end{aligned}$$

and the initial box is $-10 \le x_i \le 10$, $i = 1, \dots, 10$. We have 10^{10} local minima in the initial box.

Function **W29** is defined by

$$f_{\mathrm{W29}}(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2,$$

and the initial boxes are defined by $-1.2 \le x_i \le 1.2$, $i = 1, 2$ and $-10^6 \le x_i \le 10^6$, $i = 1, 2$.

Function	**W4**		**W10**		**W29**			
Initial box $[x]_i$	$[-10, 10]$		$[-10, 10]$		$[-1.2, 1.2]$		$[-10^6, 10^6]$	
$\varepsilon = 10^{-12}$	Han	New	Han	New	Han	New	Han	New
FE	2166	59	559	89	640	111	12321	1213
GE	2021	319	497	177	583	187	12827	2399
HE	725	69	184	41	238	50	4949	593
E_effort_1	8383	904	15649	4114	2520	475	52822	7790
E_effort_2	7658	835	4387	1207	2282	585	47873	7197

Additional test results for the special splitting strategy are given in [16].

In Section 4, we mentioned two other strategies for splitting the box in the interval Gauss-Seidel step. If we use Strategy 3 and 4, we get no improvement, in general. In many cases, we have

$$0 \in [A]_{ii} \quad \text{and} \quad (b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} [A]_{ij} \cdot ([z]_j - c_j))/[A]_{ii} = (-\infty, \infty)$$

in the computation of $N'_{\mathrm{GS}}([y])$ according to (4). Thus, we get $[z]_i := (-\infty, \infty) \cap [z]_i$ unchanged.

But if we use a little trick, we can get some advantage of the knowledge of this situation. We use "gaps" of width zero, by splitting $[z]_i$ in $[z]_i = [v]_i \cup [w]_i$ with $[v]_i := [\underline{y}_i, m([y]_i)]$ and $[w]_i := [m([y]_i), \overline{y}_i]$, whenever we have $0 \in [A]_{ii}$ and $[z]_i$ remains unchanged. This is actually a bisection step.

Testing our different strategies including the trick for gaps of width zero, for Griewank's function of dimension 5 and 7, we get the following encouraging results:

G5				
Splittings	2	$n+1$	2^3	2^n
Preconditioning	InvMid	InvMid	InvMid	InvMid
FE	127	199	233	447
GE	261	127	105	61
HE	78	40	34	19
E_effort_1	2602	1434	1268	1037
E_effort_2	1561	907	823	786

G7				
Splittings	2	$n+1$	2^3	2^n
Preconditioning	InvMid	InvMid	InvMid	InvMid
FE	?	1360	528	2045
GE	?	727	249	63
HE	?	173	74	20
E_effort_1	>> 100000	11293	4343	3046
E_effort_1	>> 100000	5479	2042	2437

The results presented in this paper demonstrate that it is possible to improve the efficiency of global optimization method incorporating the interval Gauss-Seidel step including special splitting strategies which use as much information (from the already computed data) as possible. Future research work should investigate possible strategies for treating the "gaps of width zero" more effective.

References

[1] Alefeld, G., Herzberger, J.: *Introduction to Interval Computations*. Academic Press, New York, 1983.

[2] In: Atanassova, L., Herzberger, J. (Eds.): *Computer Arithmetic and Enclosure Methods*. North-Holland, Elsevier, Amsterdam, 1992.

[3] Fischer, H.-C.: *Schnelle automatische Differentiation, Einschließungsmethoden und Anwendungen*. Dissertation, Universität Karlsruhe, 1990.

[4] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *Numerical Toolbox for Verified Computing I – Basic Numerical Problems*. Springer-Verlag, Heidelberg, New York, 1993.

[5] Hansen, E.: *Global Optimization Using Interval Analysis – The Multi-Dimensional Case*. Numerische Mathematik **34**, pp 247–270, 1980.

[6] Hansen, E.: *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 1992.

[7] Hansen, E. und Greenberg, R.: *An Interval Newton Method*. Applied Mathematics and Computations **12**, pp 89–98, 1983.

[8] Hansen, E., Sengupta, S.: *Bounding Solutions of Systems of Equations Using Interval Analysis*. BIT **21**, pp 203–211, 1981.

[9] Jansson, C.: *A Global Optimization Method Using Interval Arithmetic*. In: [2], pp 259–267, 1992.

[10] Kearfott, R. B.: *Preconditioners for the Interval Gauss-Seidel Method*. SIAM Journal of Numerical Analysis **27**, pp 804–822, 1990.

[11] Kearfott, R. B., Hu, C. und Novoa, M.: *A Review of Preconditioners for the Interval Gauss-Seidel Method*. Interval Computations **1**, pp 59–85, Institute for New Technologies, St. Petersburg, 1991.

[12] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL-XSC – Language Reference with Examples*. Springer-Verlag, New York, 1992.

[13] Neumaier, A.: *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[14] Ratschek, H., Rokne, J.: *New Computer Methods for Global Optimization*. Ellis Horwood Limited, Chichester, 1988.

[15] Ratz, D.: *An Inclusion Algorithm for Global Optimization in a Portable PASCAL-XSC Implementation*. In: [2], pp 329–338, 1992.

[16] Ratz, D.: *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. Dissertation, Universität Karlsruhe, 1992.

Dr. Dietmar Ratz
Institut für Angewandte Mathematik
Universität Karlsruhe
D-76128 Karlsruhe
Federal Republic of Germany