Rigorous Computation of Surface Patch Intersection Curves

R. Baker Kearfott* Zhaoyun Xing

July 13, 1993

Abstract

A rigorous and efficient algorithm is presented for computing a sequence of points on all the branches of surface patch intersection curves within a given box. In the algorithm, an interval step control continuation method makes certain that the predictor algorithm will not jump from one branch to the another. These reliability properties are independent of any choice of tuning parameters. Both a 3-dimensional box complement method and a containment checking method are able to guarantee that all branches are located. Initial experimental results show that, even with this reliability, the amount of computation is orders of magnitude less than a uniform tesselation of the three-dimensional viewing box.

Keywords: computational geometry, marching method, continuation method, surface patch intersections, interval computations.

1 Introduction and Notation

The goal of this paper is to present general algorithms for computing all surface / surface intersection curves that are *mathematically rigorous* in the

*This work was partially supported by National Science Foundation grant no. CCR-9203730.

sense that they never overlook any distinct intersection curves in the viewing region. We achieve this with interval computations.

We assume familiarity with interval computations, though we explain one or two crucial properties here. General introductions to the subject appear in [11], [14], and elsewhere. A fairly complete introduction in the context of computational geometry appears in [12]. Other articles involving interval computations in this context include [5], [15] and [16].

The crucial underlying property of interval computations is that, with directed roundings, the arithmetic is defined on intervals that rigorously contain the true results. This allows both bounding of computational errors and rigorous computation of bounds on ranges of functions over regions of significant size. Such range bounds can be used for proving feasibility or infeasibility of inequality constraints, proving non-existence of a system of nonlinear equations, etc. Geometrically, we work with *interval vectors* $\mathbf{x}$ that are rectangular parallelepipeds, or *boxes*. We can prove existence and uniqueness of a solution of a nonlinear system of equations in a box $\mathbf{x}$ with an *interval Newton method*: Denote the image of $\mathbf{x}$ under an interval Newton method for the system of equations $F = 0$ by $N(F, \mathbf{x})$. Then $N(F, \mathbf{x}) \subset \overset{\circ}{\mathbf{x}}$ implies $F = 0$ has a unique solution within $\mathbf{x}$, where $\overset{\circ}{\mathbf{x}}$ is the topological interior of the box $\mathbf{x}$; conversely, if $N(F, \mathbf{x}) \cap \mathbf{x} = \emptyset$, then there are no roots of F in $\mathbf{x}$. For theory on interval Newton methods, consult [14]. The particular interval Newton method used here is a variant of the *interval Gauss–Seidel method*, with a special preconditioner. This method was introduced in [8] and was analyzed in the context of marching methods for curves in [9]. An alternate approach to using interval methods to delimit curves and other manifolds is the adaptive tesselation / interval Newton procedure given in [13].

Throughout this paper, we use boldface lowercase letters (such as $\mathbf{x}$) for interval vectors, uppercase letters (such as X) for point vectors, and lowercase letters and lowercase letters with subscripts (such as x_i) for scalar variables. We will use notation such as $\mathbf{H}(\mathbf{x})$ to denote an interval extension of $H(x)$ over $x \in \mathbf{x}$.

The problem we address can be posed as:

> Compute sequences of points on surface patch intersection curves defined by
> $$\begin{cases} f(x, y, z) &= 0 \\ g(x, y, z) &= 0 \end{cases}$$

with $(x, y, z) \in \mathbf{x}_0$, where f and g are two nonlinear functions and $\mathbf{x}_0$ is a given box.

Computing the intersection curve of two surface patches (hereinafter referred to as an SSI curve for surface-surface intersection) is a fundamental problem in computer aided geometric design and graphics. It arises in a broad spectum of tasks, ranging from constructive solid geometry to contouring of scattered data. In general, the intersection curve of two surface patches cannot be expressed exactly in terms of one parameter using an implicit equation; see [6]. Therefore, SSI curves can only be approximated. There are two dominant approaches to the problem of SSI curves: subdivision and marching. Subdivision-based algorithms characteristically tessellate the surfaces into piecewise linear approximations, and intersect the facets. Marching, continuation, or curve following algorithms begin by finding one point on the intersection curve, and then proceed to march around the curve. For a general introduction to continuation methods, see [1]. Selected references for alternate methods are [2], or [4], [10].

A major limitation of previous marching methods is that there is no assurance that all branches of the intersection curve have been detected. This paper presents an alternate marching algorithm that uses interval step control in conjunction with either a complementation method or with a containment checking method, along with box bisection, to locate all branches. In effect, this approach is a rigorous hybrid between marching and tesselation.

Overall, our algorithms are similar to the algorithm in [2] in the sense that we first find points on the intersection curves, then proceed to trace the corresponding curves. However, our algorithms, though adaptively subdividing the region, work with boxes (rectangular regions), and rigorously but exhaustively search the entire region, independent of the choice of tuning parameters.

In broad overview, our algorithms proceed as follows. Given two nonlinear surface patches and a box, we use box bisection and interval Gauss–Seidel methods to locate a point on one branch of the intersection curve. We then compute a sequence of points on this branch with an interval step control continuation method. The continution method generates a sequence of boxes through which only the current branch of the curve passes. It stops when a loop is detected, the branch is followed out of $\mathbf{x}_0$, or a bifurcation point is detected.

Our two algorithms differe in the way that additional curves are found. In our containment checking algorithm, we bisect the original box $\mathbf{x}_0$ until either the resulting subbox is contained in some box from the sequence of boxes generated from previous continuation, or this box is within roundoff error of a point on some branch. In the first case, the box can be rejected, and another box from the subdivision is fetched for further processing. In the second case, since the boxes from continuation cover the branches already followed, the point-sized box must be a point on another branch. The algorithm stops when all subboxes are rejected.

In our complement algorithm, the complements in the box $\mathbf{x}_0$ to the sequence of boxes are computed. This set of complements is the union of a sequence of subboxes of $\mathbf{x}_0$. The algorithm checks each subbox in this complement to see if it contains points on a solution branch. If a new branch is detected, it must be different from the previously computed branch, since the subbox is in the complement of the box cover of the previously computed branch.

Our experimental results show that both the complement method and containment checking method are efficient in certain contexts.

Once box coverings of the curves are obtained via our algorithm, fitting procedures can be used, as desired, to obtain economical representations.

2 The Algorithms

2.1 Locating A Starting Point

The following procedure is used in both overall algorithms to find an initial starting point on some branch. It uses box bisection and interval Gauss–Seidel iteration. Unlike the classical Newton's method, these techniques can guarantee that the located point is within the considered box. Figure 1 illustrates the procedure.

Algorithm 1 (Finding the starting point)

Step 1 *Input the initial box* $\mathbf{x}_0$*, the functions* f *and* g*, a resolution* ϵ*, and maximum stack length* $l_{\max}$*.*

Step 2 *Initialize the stack* $\mathcal{S}$ *by placing* $\mathbf{x}_0$ *onto it.*

Step 3 *Pop a box* $\mathbf{y}$ *from* $\mathcal{S}$ *(see Figure 1), and*

Step 4 *Do a sweep of interval Gauss–Seidel iteration on the box* $\mathbf{y}$.

If $\mathbf{y}$ *is determined to have no solutions, then*

a) Reject $\mathbf{y}$.

b) If $\mathcal{S}$ *is empty, then stop. Otherwise,* **cycle back to Step 3**.

else

a) Bisect $\mathbf{y}$ *into two boxes* $\mathbf{y}_1$ *and* $\mathbf{y}_2$.

b) If the diameter of either $\mathbf{y}_1$ *or* $\mathbf{y}_2$ *is smaller than* ϵ*, then*

i) return its midpoint,

ii) return the message "starting point is found by rounding."

else

i) put $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *onto* $\mathcal{S}$.

ii) If the length of $\mathcal{S}$ *is greater than* $l_{\max}$*, then return the message "there are too many boxes on the stack."*

endif

endif

2.2 Interval Step Control Continuation

A continuation method utilizing interval step control appears in [9]. It generates a sequence of boxes through which only one branch of the intersection curves passes. Continuation stops when a bifurcation point is detected or when the point on the branch is outside the initial box $\mathbf{x}_0$. In contrast to continuation with an approximate step control, this continuation method is deterministic in the sense that

1. it computationally but rigorously verifies that the corrector iteration will converge to a point on the same branch of the intersection curve as the previous point, i.e., the predictor / corrector iteration will never jump across branches, and

2. each predictor step is as large as possible, subject to verification that the curve is unique with the given interval extension.

The following alogrithm is a brief outline; for details see [9].

Algorithm 2 (Sketch of the continuation method)

step 1 *Input an initial point X on some branch of the intersection curve.*

step 2 *Compute the parameter coordinate and decide the orientation.*

If the condition number of the Jacobi matrix at X is too large, then

i) return the message "bifurcation point" detected.

ii) Construct a small box around X, then put this box into a box list $\mathcal{L}$.

endif

step 3 *Construct an appropriate box* $\mathbf{x}$ *such that X is on one of the faces of* $\mathbf{x}$ *which is perpendicalar to the parameter direction.*

step 4 *Do a sweep of interval Gauss–Seidel iteration on the box* $\mathbf{x}$.

If there is unique curve passing through $\mathbf{x}$, *then*

a) Place $\mathbf{x}$ *into $\mathcal{L}$.*

b) Using the classical Newton's method, compute a new point X_1 approximately on the curve and on the face of $\mathbf{x}$ *which is perpendicular to the parameter coordinate and opposite the face with X.*

c) If $X_1 \in \mathbf{x}_0$ then

α) Replace X by X_1

β) **cycle back to step 2**.

else

return.

endif.

endif

Figure 2 illustrates interval step control continuation. Starting at X_0, the procedure generates three points along with three boxes. The first two boxes correspond to the same parameter coordinate x_1, and they are tangent on a face perpendicular to x_1. The parameter coordinate for the third box is x_2, and the second and third boxes have a non-empty intersection.

In step 2 i) of Algorithm 2, we mean numerical or imperfect bifurcation point. For example, the curve defined by

$$x^2 - (t - 0.5)^2 = p^2$$

has two branches in the box $[-1, 1] \times [-1, 1]$; see Figure 3. As p tends to 0, the point $(0, 0.5)$ is thought of as a numerical bifurcation point. But it is not a bifurcation point in the strict sense, no matter how small p is.

Since uniqueness of the curve branch is guaranteed by the Guass–Seidel check, we have the following theorem.

Theorem 1 *([9]) There is only one branch of the curve passing through any box in the list $\mathcal{L}$ generated by interval step control continuation, and $\mathcal{L}$ is a cover of the branch followed.*

2.3 The Containment Checking Algorithm

We present the *containment checking algorithm* here. This algorithm computes sequences of points on all branches of the intersection curves arising from CAD problems. The containment checking algorithm rejects boxes contained in any box in the list $\mathcal{L}$ generated from interval step control continuation, since such boxes either do not contain any point of the curve, or else the points in those boxes are on branches already followed. See Figure 4.

Algorithm 3 (Containment Checking Algorithm)

step 1 *Input the initial box* $\mathbf{x}_0$ *and functions* f *and* g.

step 2 *Initialize the stack* $\mathcal{S}$ *by placing* $\mathbf{x}_0$ *onto it.*

step 3 *Pop a box* $\mathbf{y}$ *from* $\mathcal{S}$.

If the box list $\mathcal{L}$ generated from continuation is non-empty, then

If $\mathbf{y}$ *is contained in some box in* $\mathcal{L}$, *then*

i) reject $\mathbf{y}$.

ii) *If $\mathcal{S}$ is empty, then* **return**.

else

i) *bisect* $\mathbf{y}$ *into two subboxes* $\mathbf{y}_1$ *and* $\mathbf{y}_2$.

ii) *Put* $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *onto* $\mathcal{S}$.

iii) **Cycle** *back to the beginning of step 3.*

endif

endif

step 4 *Use Algorithm 1 (the starting point location algorithm) to find an initial point* $X \in \mathbf{y}$. *Join* $\mathcal{S}$ *with the stack generated during location of* X.

step 5 *Use Algorithm 2 (the interval step control algorithm) to follow the curve in two opposite directions starting from* X.

step 6 **Cycle** *back to step 3.*

2.4 Complement Algorithm

Two algorithms are presented in this section. The first is finds the complement of a box $\mathbf{y}$ in a box $\mathbf{x}$, as in Figure 5. The second is used to compute sequences of points on each branch of the intersection curve arising from CAD problems, as in Figure 6. This complement method is used to locate all branches of the intersection curves.

There are three cases when finding the complement of $\mathbf{y}$ in $\mathbf{x}$.

1. $\mathbf{y}$ is properly contained in $\mathbf{x}$. There are four boxes in the complement. Those are $\mathbf{c}_i, i = 1, 2, 3, 4$.

2. $\mathbf{y} \cap \mathbf{x} \neq \phi$, but $\mathbf{y} \not\subset \mathbf{x}$. There are two boxes in the complement. Those are $\mathbf{c}_i$, $i = 1, 2$.

3. $\mathbf{y} \cap \mathbf{x} = \emptyset$. There is only one box $\mathbf{c}_1 = \mathbf{x}$.

Algorithm 4 (Box complementation algorithm)

step 1 *Input the boxes* $\mathbf{x}$ *and* $\mathbf{y}$.

step 2 *Set:* $\mathbf{y}_1 \leftarrow \mathbf{x}$, $\mathbf{y}_2 \leftarrow \mathbf{x}$.

step 3 *For* $i = 1, 3$ *do*

a) Compute the one dimensional intersection $\mathbf{z}$ *of* $\mathbf{x}_i$ *and* $\mathbf{y}_i$.

b) If $\mathbf{z}$ *is properly contained in* $\mathbf{x}$, *then*

i) let $\mathbf{y}_{1i}$ *and* $\mathbf{y}_{2i}$ *be the one dimensional complements of* $\mathbf{z}$ *in* $\mathbf{x}_i$.

ii) Put $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *onto the complement list* $\mathcal{L}_c$.

iii) Replace $\mathbf{x}_i$ *by* $\mathbf{z}$.

else

If the complement of $\mathbf{z}$ *in* $\mathbf{x}$ *is empty, then*

Cycle *to next* i.

else

α) *Let* $\mathbf{y}_{1i}$ *be the one dimensional complement of* $\mathbf{z}$ *in* $\mathbf{x}_i$.

β) *Put* $\mathbf{y}_1$ *onto* $\mathcal{L}_c$.

γ) *Replace* $\mathbf{x}_i$ *by* $\mathbf{z}$.

endif

endif

end do

Algorithm 4 is related to the bisection algorithm (Algorithm 3.1) of [7]. In fact, a similar technique can be used to find the complement of an n-dimensional box in another n-dimensional box.

Algorithm 5 (Complement Algorithm)

step 1 *Input a box* $\mathbf{x}_0$ *and functions* f *and* g.

step 2 *Initialize* $\mathcal{S}$ *by placing the box* $\mathbf{x}_0$ *onto it.*

step 3 *Do while* $\mathcal{S} \neq \emptyset$

a) Pop a box $\mathbf{y}$ *from* $\mathcal{S}$.

b) Use Algorithm 1 (starting point location algorithm) to locate an initial point $X \in \mathbf{y}$ on the curve.

c) If X was successfully found in the previous step, then create a box cover of the path passing through X.

i) Use Algorithm 2 (the interval step control algorithm) to follow the curve in two opposite directions starting from X.

ii) Replace $\mathcal{L}$ by the new box list generated by interval step control continuation.

d) If a new box list $\mathcal{L}$ was generated from continuation in the previous step, then

i) for each box $\mathbf{z}$ in the new list $\mathcal{L}$, find the complement of $\mathbf{z}$ in each box $\mathbf{x} \in \mathcal{S}$. This procedure generates a new stack of boxes.

ii) Replace $\mathcal{S}$ by this new stack.

endif

end do

3 Modified interval step control continuation

Performance of the containment checking algorithm and the complement algorithm depends on how well the box list generated by interval step control continuation covers the followed branch. But Figure 2 illustrates how the union of the sequence of boxes can fail to contain the curve in its interior. Since the parameter coordinate changes from x_1 to x_2 at the point X_2, the cover at the point X_2 is bad in the sense that the point X_2, while approximately on the curve, is also on the boundary of the union of boxes. This may result in small boxes near the path that cannot be rejected, as in Figure 7. To avoid this, we present a modified version of the continuation method for the containment checking algorithm. In a step corresponding a change of parameter, one more box is constructed with the same parameter to get a better cover of the followed branch. See Figure 8.

The complement algorithm could also benefit from this modified continuation method, since interval Newton methods, in general, cannot prove non-existence or uniqueness when roots lie near boundaries.

Algorithm 6 (Modified interval step control continuation)

step 1 *Input an initial point X on some branch of the intersection curve.*

step 2 *Compute the parameter coordinate and decide the orientation.*

a) If the condition number of the Jacobi matrix at X is too large, then

i) return the message "bifurcation point" detected.

ii) Construct a small box around X, then place this box into a list $\mathcal{L}$.

endif

b) If the current parameter is different from the previous parameter, then

i) use the previous parameter.

ii) Construct an appropriate box $\mathbf{x}$ such that X is on one of the faces of $\mathbf{x}$ which is perpendicalar to the parameter direction.

iii) Do a sweep of interval Gauss–Seidel iteration on the box $\mathbf{x}$. If this computation determines that a unique curve passes through $\mathbf{x}$, then place $\mathbf{x}$ into $\mathcal{L}$.

endif

step 3 *Construct an appropriate box $\mathbf{x}$ such that X is on one of the faces of $\mathbf{x}$ which is perpendicular to the parameter direction.*

step 4 *Do a sweep of interval Gauss–Seidel iteration on the box $\mathbf{x}$. If this computation determines that there is a unique curve passing through $\mathbf{x}$, then*

a) Place $\mathbf{x}$ into $\mathcal{L}$.

b) Using the classical Newton's method, compute a new point X_1 approxiamtely on the curve and on the face of $\mathbf{x}$ which is perpendicular to the parameter coordinate and opposite X_1.

c) If $X_1 \in \mathbf{x}_0$, where $\mathbf{x}_0$ is the entire box, then **cycle** *back to step 2.*
else return.
endif

endif

4 Numerical Experiments

We implemented both the containment checking algorithm and complement algorithm in the language FORTRAN-SC (ACITH-XSC) on an IBM3090. We used Algorithm 1 for the box complement algorithm and Algorithm 6 for the containment checking algorithm, although additional experiments may show that the box complement algorithm also benefits from Algorithm 6.

We tried the algorithm on two simple examples.

4.1 Bilinear Surfaces

In [17], an intersection curve of two bilinear surfaces is given. In parametric form, the surfaces are

$$P(s,t) = P_{00}(1-s)(1-t) + P_{10}s(1-t) + P_{01}(1-s)t + P_{11}st$$

where $P_{00} = (0,0,0), P_{10} = (0,1,4), P_{01} = (3,3,0)$, and $P_{11} = (4,0,4)$ and

$$Q(u,v) = Q_{00}(1-u)(1-v) + Q_{10}u(1-v) + Q_{01}(1-u)v + Q_{11}uv$$

where $Q_{00} = (0,0,0), Q_{10} = (0,4,4), Q_{01} = (4,2,0)$, and $Q_{11} = (4,0,4)$. In implicit form, they are

$$\begin{cases} z^2 - 4xz + 12x - 12y & = & 0 \\ 3xz - 4x + 8y - 8z & = & 0. \end{cases}$$

By inspection, point $(0,0,0)$ must be on the intersection curve. We fill follow curves within the box $[-2,2] \times [-2,2] \times [-2,2]$. In tables 1 and 2, we use the following abbreviations.

CCA : the containment checking algorithm;

BCA : the complement algorithm.

4.2 Spherical and Cylindrical Surfaces

In [4], the intersection curve of a spherical surface and a cylinderical surface is given. In implicit form, they are

$$\begin{cases} x^2 + y^2 + z^2 - 4 &= 0 \\ (x-1)^2 + z^2 - 1 &= 0. \end{cases}$$

By inspection $(2, 0, 0)$ is on the intersection curve. When curves within the box $[-2, 2] \times [-2, 2] \times [-2, 2]$ are followed, we obtain Table 1.

4.3 Comparisions of the Two Algorithms

Basically, the ideas for both the containment checking algorithm and complement algorithm are the same. They both try to use the fact that there is unique curve passing through the boxes generated by interval step control continuation, and try to reject boxes contained in the union of the boxes generated by interval step control continuation. In the containment checking algorithm, starting with the initial box, a stack of boxes is generated. A box is rejected if it is contained in some box obtained by continuation. A limitation of this approach is that any box only partially contained in some box obtained by continuation can not be rejected. This can be avoided by the complement algorithm, since, in that case, only the boxes within the complement of the cover of the boxes obtained by continuation are considered. But the complement algorithm also has a limitation: when the number of boxes generated by interval step control continuation is large, finding the complement is time consuming. A preliminary conclusion evident from the experiments reported here is:

> When the number of the boxes generated by interval step control continuation is less than 100, the complement algorithm works better: see tables 1 and 2. Otherwise, the containment checking algorithm is more efficient.
>
> To pick an efficient algorithm, our suggestion is:
>
> 1. If the initial box size is small compared to the curvatures of the surfaces, choose the box complement algorithm. Otherwise, the containment checking algorithm is a better choice.

2. First partition the initial box into several subboxes, and use the complement algorithm on each of them.

In our experiments with simple algebraic curves, the average step size in the interval step control continuation is 0.01 to 0.05. In such cases, to make the number of boxes produced by continuation less than 100, the box size (i.e. the length of each edge of the boxes) should be less than about 1 to 5.

5 Conclusions and Future Work

We have presented two algorithms for surface / surface intersections based on interval step control continuation. Both algorithms can locate all branches of the intersection curves of two surfaces within a given box. The efficiencies of these algorithms depend on the initial box size and the implementation. Our numerical experiments hint that each is efficient when the appropriate one is chosen for a particular problem.

Our initial experiments, reported above, involved two simple systems of equations given in implicit form. However, the techniques are equally applicable to surfaces given in parametric form, and indeed, to surfaces defined in a piecewise fashion, in which the pieces are given in parametric form. For example, to find the intersection

$$P(s,t) = Q(u,v) = 0,$$

we simply apply the techniques to the system of three equations in the four unknowns s, t, u , and v given by

$$\begin{aligned} P_1(s,t) &= Q_1(u,v) \\ P_2(s,t) &= Q_2(u,v) \\ P_3(s,t) &= Q_3(u,v). \end{aligned}$$

The continuation method in [9] is general, and our adaptive subdivision algorithms and complementation algorithm explained above generalize in a straightforward way. We will perform a definitive set of such tests upon completion of the appropriate software environment.

References

[1] Allgower, E., and Georg, K., *Numerical Continuation Methods: An Introduction*, Springer-Verlag, New York, 1990.

[2] Barnhill, R. E. and Farin, G., *Surface / Surface Intersection*, Comput.-Aided Geom. Des. **4**, pp. 3–16, 1987.

[3] Faux, I. D. and Pratt, M. J., *Computational Geometry for Design and Manufacture*, Horwood, Chichester, England, 1983.

[4] Garrity, T., and Warren, J., *On computing the intersection of a pair of algebraic surfaces.*, Comput.-Aided Geom. Des. **6** (2), pp. 137–153, 1989.

[5] Grandine, T. A., *Computing zeroes of spline functions.*, Comput.-Aided Geom. Des. **6** (2), pp. 129–136, 1989.

[6] Katz, S., Sederberg, T. W., *Genus of the intersection curve of two rational surface patches.*, Comput.-Aided Geom. Des. **5** (3), pp. 253–258, 1988.

[7] Kearfott, R. B., *Interval Newton / Generalized Bisection When There are Singularities near Roots*, Annals of Operations Research **25**, pp. 181–196, 1990.

[8] Kearfott, R. B., *Preconditioners for the Interval Gauss–Seidel Method*, SIAM J. Numer. Anal. **27** (3), pp. 804–822, 1990.

[9] Kearfott, R. B. and Xing, X. Y., *An Interval Step Control for Continuation Methods*, SIAM J. Numer. Anal., 1993 (in press).

[10] Lee, R. B. and Fredricks, D. A., *Intersections of Parametric Surfaces and a Plane*, IEEE Comput. Graphics and Appl. **4** (8), pp. 48–51, 1984.

[11] Moore, R. E., *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[12] Mudur, S. P. and Koparkar, P. A., *Interval Methods for Processing Geometric Objects*, IEEE Comput. Graphics and Appl. **4** (2), pp. 7–17, 1984.

[13] Neumaier, A., *The Enclosure of Solutions of Parameter-Dependent Systems of Equations*, Academic Press, New York, etc., in Reliability in Computing: The Role of Interval Methods in Scientific Computing, pp. 269–286, Academic Press, New York, etc., 1988.

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

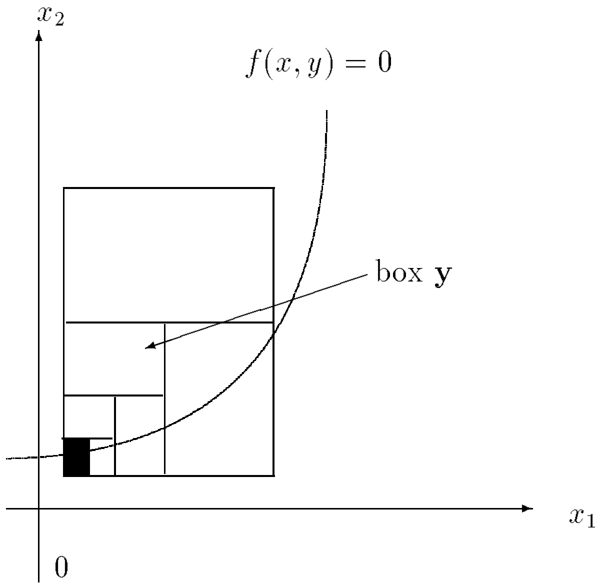
[15] Ratschek, H. and Rokne, J., *Test for Intersection between Plane and Box*, Comput. Aided Des. **25** (4), pp. 249ff, 1993.

[16] Sederberg, T. W., and Parry, S. R., *Comparison of Three Curve Intersection Algorithms*, Comput. Aided Des. **18** (1), pp. 58–63, 1986.

[17] Sederberg, T. W., Meyers, R. J., *Loop detection in surface patch intersections.*, Comput.-Aided Geom. Des. **5** (2), pp. 161–171, 1988.

[18] Walker, R. J., *Algebraic Curves*, Princeton Univ. Press, Princeton, N.J., 1950.

æ



æ

Figure 1: In this 2-dimensional picture, the filled box is accepted as containing a starting point. The other boxes, such as box **y**, are either on the stack $\mathcal{S}$ or have been rejected.

algorithm type	*CCA*	*BCA*
# of branches	1	1
# of bifurcation point	0	0
# of boxes rejected by containment checking	19,482	
# of boxes rejected by Gauss–Seidel method	26,023	1,470
total # of boxes rejected	45,505	1,470
total # of boxes generated by continuation	64	64
CPU Time in seconds	184	20.5
maximum stack length	34	

Table 1: experimental results for the intersection curves of two bilinear surfaces.

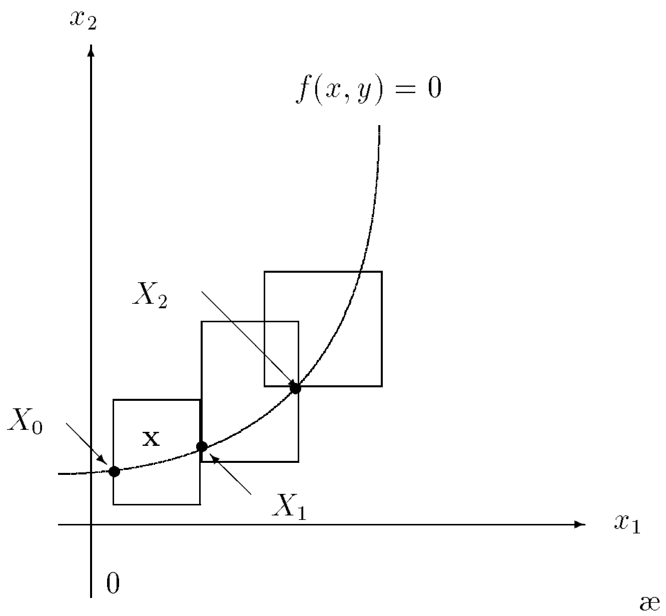


Figure 2: Interval step control continuation.

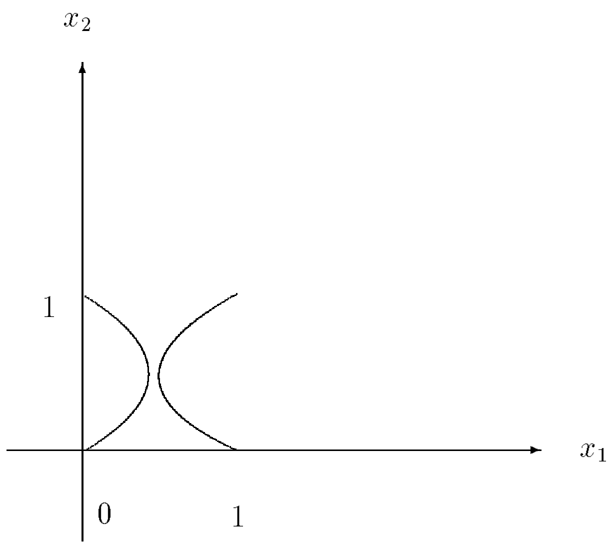
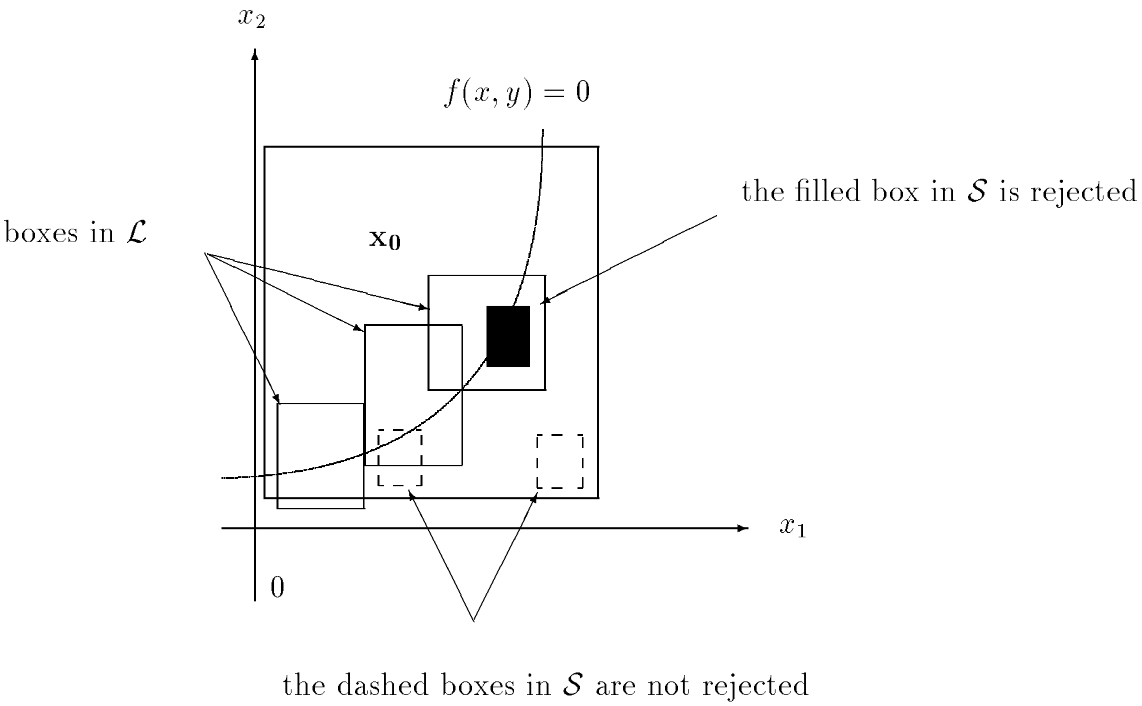


Figure 3: The hyperbolic curve.



æ

Figure 4: Illustration of a sequence of boxes $\mathcal{L}$. A box on stack $\mathcal{S}$ may be rejected by checking if it is contained in a box in $\mathcal{L}$.

algorithm type	*CCA*	*BCA*
# of branches	2	2
# of bifurcation point	1	1
# of boxes rejected by containment checking	49,192	
# of boxes rejected by Gauss–Seidel method	70,074	675,816
total # of boxes rejected	119,266	675,816
total # of boxes generated by continuation	1,216	1,216
CPU Time in seconds	429	1132.5
maximum stack length	35	

Table 2: experiment for intersection curves of a sphere and a cylinder surfaces.

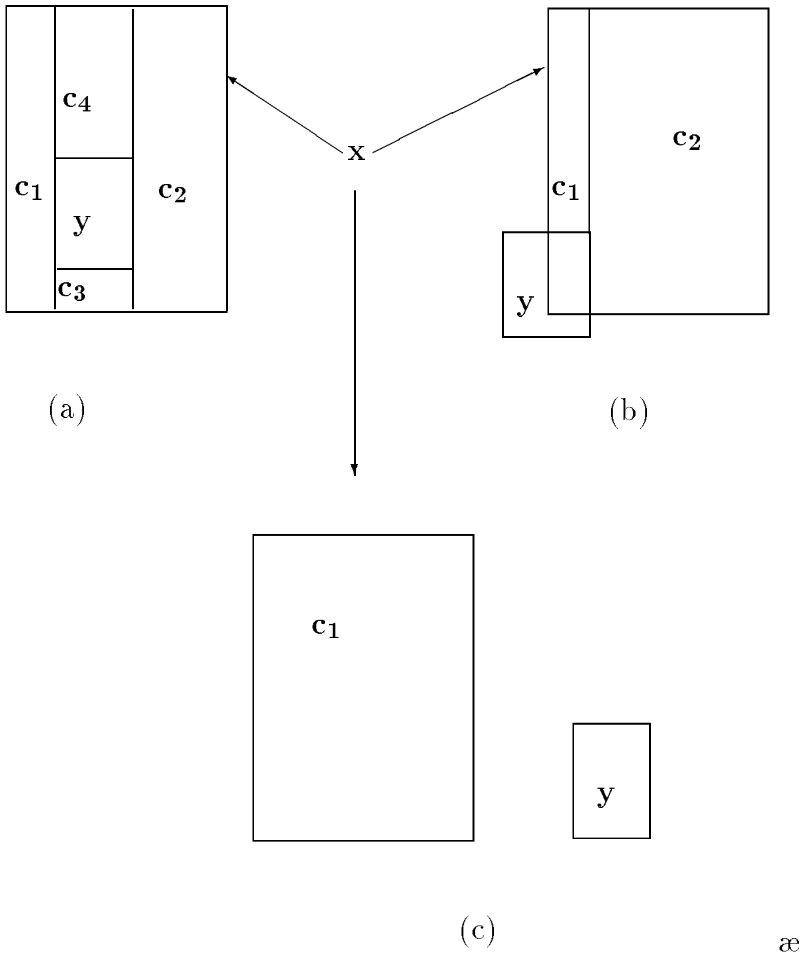


Figure 5: This figure illustrates how to represent the complement of the box $\mathbf{y}$ in the box $\mathbf{x}$ by minimum number of the boxes $\{\mathbf{c}_i\}$.

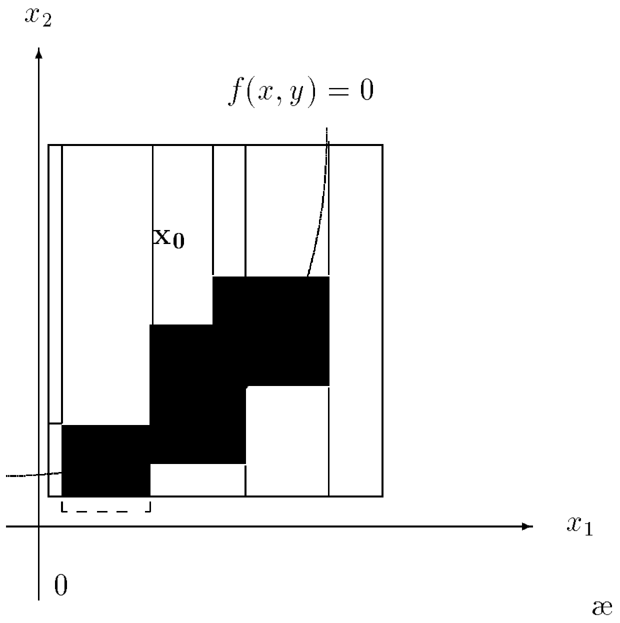


Figure 6: The filled boxes are boxes in $\mathcal{L}$, while all other boxes are in $\mathcal{L}_c$.

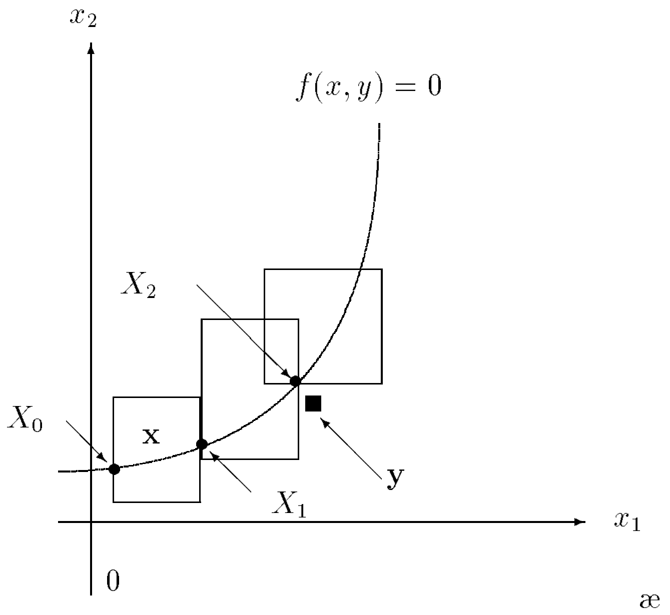
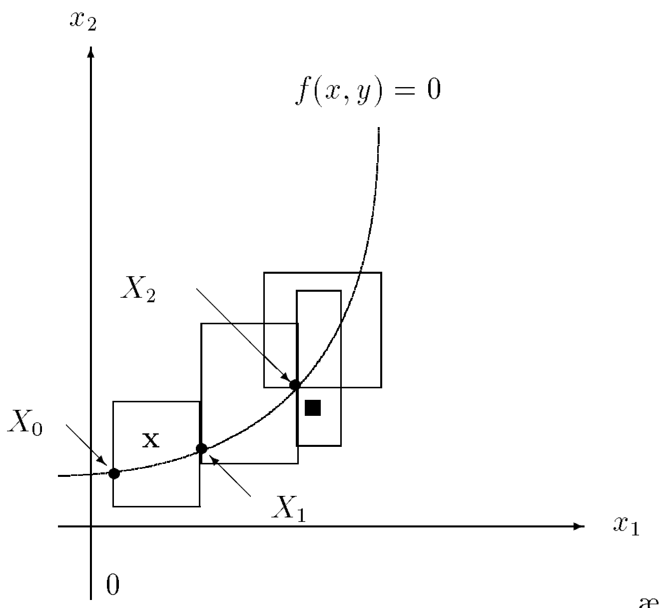


Figure 7: A difficulty when Algorithm 1 is used. The box $\mathbf{y}$ may not be rejected because roundout error may make it indistinguishable from X_2.



æ

Figure 8: Modified interval step control applied to the containment checking algorithm.