Solving Interval Constraints in Computer-Aided Design

Yan Wang
NSF Center for e-Design, University of Pittsburgh 1048 Benedum Hall Pittsburgh, PA 15261
e-mail: ywang@engr.pitt.edu

Abstract. Currently available CAD systems require geometric parameters to have fixed values. Valid range information on parameters cannot be properly represented and embedded in existing CAD data. Specifying fixed parameter values implicitly adds rigid constraints on the geometry, which have the potential to introduce conflicts during later design stages.
In this paper, a geometric modeling scheme based on nominal interval representation and analysis is presented to represent design uncertainty and inexactness. Parameters are represented by nominal intervals, which contain the information of nominal values, lower bounds, and upper bounds. Interval constraints represent inexactness at the early design stages, uncertainty in the detailed design, as well as the boundary information for design optimization.

To solve under-constrained and over-constrained interval problems, iteration-based equation solving methods are used. A generalized nonlinear constraint solving method based on linear enclosure is developed for fast convergence. Inequalities are transformed into equations and can be solved uniformly. Interval subdivision and constraint re-specification methods are developed for design refinement. Active and inactive constraints are differentiated in sensitivity analysis.

1. Introduction

During the process of design, various parameters are specified, which include geometric parameters (e.g. dimension, coordinate, and tolerance) and non-geometric ones (e.g. material characteristics, tooling speed, and expected life). Current CAD systems only allow geometric parameters to have fixed values, such as the position of a point in 3D space, the direction of a line, and the distance between two axes. Instead of simply assigning one real value to a parameter, there are some advantages to give an interval value to each parameter in a CAD model, which means that the parameter can take any valid value between the lower and upper bounds of the interval.

Fixed parameter values generate some problems. First, fixed-value constraints bring up conflicts easily at later design stages. Specifying determined parameter values implicitly adds rigid constraints on geometry. The rigid constraints reduce the freedom of geometric entities to the minimal level. These dominant constraints will be carried to other design stages and become the sources of conflicts. To resolve the conflicts, some parameter values have to be changed. This trial-and-error cycle will continue until no conflicts are found. If an interval instead of a fixed value is assigned to a parameter so that any real value within the interval is valid, the degrees of freedom of geometric entities are increased. As more constraints are imposed onto the designed object during the process of design, the freedom of geometric entities will be restricted gradually. The allowable intervals of parameter values are reduced by stages. There will be fewer chances for conflicts to occur, and several cycles of modification can be saved.

Second, the requirement of fixed parameter values makes the development of conceptual design tools difficult. At the conceptual design stage, actual values of parameters may not be known. Usually it is not important to specify fixed values of certain parameters at this stage yet. Current CAD systems require that parameter values be fully specified and fixed, thus they are not effective tools for conceptual design. It is quite challenging to develop a practically usable

Computer-Aided Conceptual Design tool based on the current scheme of fixed parameter values. Nevertheless, if a parameter is specified as a range, the problem of parameter partial integrity can be solved, i.e., it is not necessary to fix all values of parameters. This increases the flexibility of the geometric shape, and the inexactness of design is modeled.

Third, the specifications of valid parameter range are not captured by fixed-value data. Current design optimization process often occurs after parameters are specified at the detailed design stage, while the original intention of feasible ranges of parameters from upstream design activities is not transferable with the fixed-value scheme. Parameter bounds have to be added separately for optimization purpose. However, with the interval representation, the inherent range information is directly applicable for parameter optimization. Parameter intervals appropriately represent design intent of feasibility, thus integrating the sketching and optimization of design.

In real design situations, there are some uncertainty factors in CAD modeling. The dimensions and shape of the designed objects are computed and stored digitally in CAD systems. Representing an infinite number of real numbers by a finite number of bits requires approximation. Not all decimal numbers can be represented in binary format exactly. Rounding errors come from the approximation. Cancellation errors occur because of catastrophic and benign cancellation. The precision of numbers in a computer depends on the word size and floating-point representation. Variation exists among different systems with different architectures. Uncertainty also comes from the measurement and tolerance of human perception during the parameter specification. The real value of measurement is the ideal case that is hard to realize from the statistical point of view. In the interval geometry representation, a computer-generated value can be looked as a sample from the range of values, while the CAD data of a designed object is a sample from the population of models. Parameter intervals capture the uncertainty characteristics of design.

In this paper, a nominal interval constraint representation (NICR) scheme based on nominal interval values is described to represent design uncertainty and inexactness. It represents soft constraints, thus reducing the chance of conflicts during constraint imposition. It provides a generic numerical parameter scheme to represent inexactness at the early design stages, uncertainty in the detailed design, as well as the boundary information for design optimization.

2. Background

Methods of interval analysis have started being used in computer graphics, including rasterizing parametric surfaces (Mudur and Koparkar [1]), ray tracing of parametric surfaces (Toth [2]) and implicit surfaces (Kalra and Barr [3]), collision detection of polyhedral objects (Moore and Wilhelms [4]) and surfaces (Von Herzen et al. [5], Duff [6], Snyder et al. [7, 8]). In design and engineering applications, Bliek [9] studied the interval analysis to ensure the numerical reliability in design computation. Rao and Berke [10] used interval arithmetic in imprecise structural analysis. Rao and Cao [11] applied interval analysis in design optimization of mechanical systems. Muhanna and Mullen [12, 13] developed interval-based finite-element formulation methods for uncertainty in solid and structural mechanics. Interval arithmetic and analysis provide efficient and scalable methods to solve constraint systems. Related to interval representation, set-based modeling [14], probabilistic modeling, and fuzzy logic are also applied in engineering design.

In CAD applications, Sederberg and Farouki [15] used interval arithmetic in approximating Bezier curves. Maekawa and Patrikalakis [16, 17] used interval Bezier curves to solve shape interrogation problems. Hu *et al*. [18, 19] developed rounded-interval arithmetic (RIA) to ensure numerical robustness in Boolean operations and boundary evaluation. Further, Abrams *et al*. [20], Shen and Patrikalakis [21] applied RIA in interval non-uniform rational B-splines. Tuohy *et al*. [22] applied interval methods for interpolating measured data with B-spline curves and surfaces. Wallner *et al*. [23] used intervals to bound errors in geometric construction. Chen and Lou [24] proposed methods to bound interval Bezier curve with lower degree interval Bezier curve. Lin *et*

al. [25] investigated the boundary evaluation of interval Bezier curve. The above research concentrates on the improvement of model's robustness. Interval parameters embody rounding and cancellation errors during floating-point computation.
From a different perspective, the NICR presented here allows all numerical values of parameters including coordinates, dimensions, and other values to be nominal interval numbers. Thus, it allows incomplete and inexact of design specification especially at the conceptual design stage. *Soft* constraints compared to traditional fixed-value *rigid* constraints can be represented. Parameters have intrinsic bounds for design optimization. Design intent of parameter validity can be integrated into constraints. The probabilistic property of variables is captured in interval constraints, which provides a unified representation for different model applications, such as Monte Carlo simulation for finite element analysis, tolerance analysis and synthesis, and producibility analysis

3. Nominal interval constraints

In NICR, we define interval number X as X = [xL, xN, xU] which contains lower bound value xL, nominal value xN, and upper bound value xU. The nominal value is usually corresponding to the specified fixed value in current CAD systems.

The introduction of the nominal value into an interval is necessary for CAD modeling. The nominal value represents the actual user specification if the parameter is fixed. It allows current CAD modeling system to adopt interval parameters so that ICR can be integrated with current fixed-value schemes and visualization methods. Furthermore, the nominal value is allowed to change within the range when user specifies a different value. This allows more user interaction and captures more specification information. For example, a 2D point P([*1,2,3*],[*4,5,6*]) can be displayed at (*2,5*). When P is fixed, its coordinates are ([*2,2,2*],[*5,5,5*]). To simplify the notation, we can use a real number for a degenerated interval.

3.1 Basic Nominal Interval Definitions

An n dimensional real number space is denoted as Rn. An n dimensional interval number space is denoted as IRn. $X = [x_L, x_N, x_U] = \{x | x_L \le x \le x_U, x_L \le x_N \le x_U\}$, where xL $\in$ R, xN $\in$ R, xU $\in$ R, and X $\in$ IR.
Given that A = [aL, aN, aU], B = [bL, bN, bU], and $\wedge$ is logical and, we have the following relations:

- equivalence: $A = B \Leftrightarrow (a_L = b_L) \wedge (a_U = b_U)$.
- nominal equivalence: $A := B \Leftrightarrow (a_L = b_L) \wedge (a_N = b_N) \wedge (a_U = b_U)$.
- strictly greater than or equal to: $A \sim\!\ge B \Leftrightarrow a_L \ge b_U$.
- strictly greater than: $A \sim\!> B \Leftrightarrow a_L > b_U$.
- strictly less than or equal to: $A \sim\!\le B \Leftrightarrow a_U \le b_L$.
- strictly less than: $A \sim\!< B \Leftrightarrow a_U < b_L$.
- inclusion: $A \subseteq B \Leftrightarrow (a_U \le b_U) \wedge (a_L \ge b_L)$, $A \subset B \Leftrightarrow (a_U < b_U) \wedge (a_L > b_L)$.

The relations of intervals are illustrated in Figure 1. *0* = [*0,0,0*] is *zero interval*. Interval *A* is *positive* (*negative*), iff *A* $\sim\!>$ *0* (*A* $\sim\!<$ *0*). If the nominal value of *A* = [a_L, a_N, a_U] is not of concern, *A* can simply be denoted as [a_L, a_U].

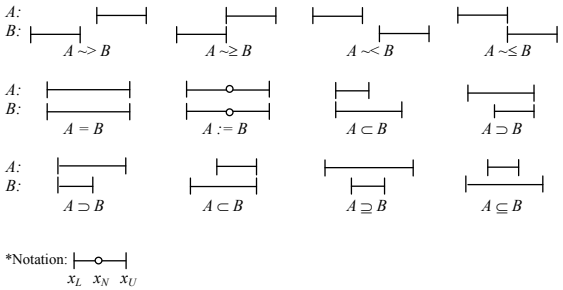

Figure 1: Relations between intervals

Interval $A = [a_L, a_N, a_U]$ is *empty*, denoted as $A = \varnothing$, iff $a_L > a_U$. A is *invalid* when $a_N > a_U$, or $a_L > a_N$, or A is empty. The basic arithmetic and set operations are defined as:

- $A \cap B = \{x \mid x \in A \text{ and } x \in B, x \in \mathbf{R}\}$. If $A \cap B \neq \varnothing$, it can be derived by $A \cap B = [\max\{a_L, b_L\}, (\max\{a_L, b_L\} + \min\{a_U, b_U\})/2, \min\{a_U, b_U\}]$.
- $A \cup B = \{x \mid x \in A \text{ or } x \in B, x \in \mathbf{R}\}$. If $A \cap B \neq \varnothing$, it can be derived by $A \cup B = [\min\{a_L, b_L\}, (\min\{a_L, b_L\} + \max\{a_U, b_U\})/2, \max\{a_U, b_U\}]$.
- $A \setminus B = \{x \mid x \in A \text{ and } x \notin B, x \in \mathbf{R}\}$.
- $A + B = [a_L + b_L, a_N + b_N, a_U + b_U]$.
- $A - B = [a_L - b_U, a_N - b_N, a_U - b_L]$.
- $A \cdot B = [\min\{a_L b_L, a_L b_U, a_U b_L, a_U b_U\}, a_N b_N, \max\{a_L b_L, a_L b_U, a_U b_L, a_U b_U\}]$.
- $\frac{1}{B} = \left\{ \frac{1}{y} \middle| y \in B, 0 \notin B \right\}$.
- $$\frac{A}{B} = \begin{cases} A \cdot \frac{1}{B} & (0 \notin B) \\ [-\infty, 0, +\infty] & (B = 0) \\ [\frac{a_U}{b_L}, \frac{a_U}{b_L}, +\infty] & (a_U \le 0, b_L < 0, b_U = 0) \\ [-\infty, \frac{a_U}{b_U}, \frac{a_U}{b_U}] \cup [\frac{a_U}{b_L}, \frac{a_U}{b_L}, +\infty] & (a_U \le 0, b_L < 0, b_U > 0) \\ [-\infty, \frac{a_U}{b_U}, \frac{a_U}{b_U}] & (a_U \le 0, b_L = 0, b_U > 0) \\ [-\infty, 0, +\infty] & (a_L < 0, a_U > 0, b_L \le 0, b_U \ge 0) \\ [-\infty, \frac{a_L}{b_L}, \frac{a_L}{b_L}] & (a_L \ge 0, b_L < 0, b_U = 0) \\ [-\infty, \frac{a_L}{b_L}, \frac{a_L}{b_L}] \cup [\frac{a_L}{b_U}, \frac{a_L}{b_U}, +\infty] & (a_L \ge 0, b_L < 0, b_U > 0) \\ [\frac{a_L}{b_U}, \frac{a_L}{b_U}, +\infty] & (a_L \ge 0, b_L = 0, b_U > 0) \end{cases}.$$

The width of an interval is a real number, denoted as $\text{wid}(A) = a_U - a_L$. $\text{wid}(\varnothing) = 0$. Some other notations are $\text{ub}(A) = a_U$, $\text{lb}(A) = a_L$, and $\text{nom}(A) = a_N$.

3.2 Sampling Relation between Real Number and Interval Number

The intervals capture the uncertainty of design. A real number x is a *sample* of interval X, iff $x \in X$. The value of a parameter, which is generated by computer or selected by human designer, is a sample of the corresponding set of values within the interval. Therefore, one CAD interval model is allowed to generate different shapes because of parameter intervals. Implicitly, a CAD interval model defines a set of geometric shapes that automatically accommodate geometry variation.

Some *strict* relations exist among intervals, which are related to real number samples. $X\Re Y \Leftrightarrow \forall x \in X, \forall y \in Y, x\Re y$. $X\Re Y$ denotes that X has a *strict* relation $\Re$ with Y ($X \in \boldsymbol{IR}$, $Y \in \boldsymbol{IR}$).

- (*strict equivalence*): $A \sim= B \Leftrightarrow \forall x \in A, \forall y \in B, x = y$.
- (*strictly greater than or equal to*): $A \sim\geq B \Leftrightarrow \forall x \in A, \forall y \in B, x \geq y$.
- (*strictly greater than*): $A \sim> B \Leftrightarrow \forall x \in A, \forall y \in B, x > y$.
- (*strictly less than or equal to*): $A \sim\leq B \Leftrightarrow \forall x \in A, \forall y \in B, x \leq y$.
- (*strictly less than*): $A \sim< B \Leftrightarrow \forall x \in A, \forall y \in B, x < y$.

Besides strict relations, some *global* relations exist in interval arithmetic evaluation and problem solving. $X\Im Y \Leftrightarrow \forall x \in X, \exists y \in Y, x\Im y$. $X\Im Y$ denotes that X has a *global* relation $\Im$ with Y ($X \in \boldsymbol{IR}$, $Y \in \boldsymbol{IR}$).

Global relations ensure the *feasibility* of interval arithmetic operations and solutions. The goal of solving interval problems is to find a region that includes all feasible solutions. The corresponding process is to eliminate certainly infeasible points from a given region so as to make it as compact as possible. The global relations make global solution and optimization of interval analysis possible. For example, the four basic arithmetic operations of intervals follow the rule of global relation and generate the global solution with a compact bound. This is the default relation in interval analysis. Strict inequalities are special cases of global inequalities. Function evaluation and problem solving in interval analysis are normally based on global relations.

- (*global equivalence*): $A = B \Leftrightarrow \forall x \in A, \exists y \in B, x = y$.
- (*greater than or equal to*): $A \geq B \Leftrightarrow a_L \geq b_L$. Equivalently, $A \geq B \Leftrightarrow \forall x \in A, \exists y \in B, x \geq y$.
- (*greater than*): $A > B \Leftrightarrow a_L > b_L$. Equivalently, $A > B \Leftrightarrow \forall x \in A, \exists y \in B, x > y$.
- (*less than or equal to*): $A \leq B \Leftrightarrow a_U \leq b_U$. Equivalently, $A \leq B \Leftrightarrow \forall x \in A, \exists y \in B, x \leq y$.
- (*less than*): $A < B \Leftrightarrow a_U < b_U$. Equivalently, $A < B \Leftrightarrow \forall x \in A, \exists y \in B, x < y$.

In a multidimensional interval space, an interval vector can be defined in $\boldsymbol{IR}^n$ with each component as an interval value, and an interval matrix is defined in $\boldsymbol{IR}^m \times \boldsymbol{IR}^n$ with each element as an interval value. Corresponding to a real function f: $\boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$, if $f_{set}(\mathbf{X})$ denotes $\{f(\boldsymbol{x}) \mid \boldsymbol{x} = (x_1, x_2, \ldots, x_n), x_i \in X_i\ (i = 1, \ldots, n), \mathbf{X} = (X_1, X_2 \ldots, X_n), \mathbf{X} \in \boldsymbol{IR}^n\}$, the inclusion function for f at $\mathbf{X}$ F: $\boldsymbol{IR}^n \rightarrow \boldsymbol{IR}^m$ if $f_{set}(\mathbf{X}) \subseteq F(\mathbf{X})$. A natural inclusion function $f(\mathbf{X})$ for $f(\boldsymbol{x})$ is obtained by replacing each occurrence of the variable x_i by interval variable X_i. It is based on the inclusion isotonicity of the interval operations [26] and the property of pre-declared inclusions [27]. Generally, the natural inclusion function $f(\mathbf{X})$ for $f(\boldsymbol{x})$ is not tight enough because of dependency between variables and wrapping effect [28].

Interval vectors with same dimensions can be ranked and sorted.

- Interval vector **A** and **B** are in a non-decreasing order,
  $\mathbf{A} \prec \mathbf{B}$, where $\mathbf{A} = (A_1, A_2, \cdots, A_n), \mathbf{B} = (B_1, B_2, \cdots, B_n)$, if $A_n \le B_n$, and
  $\neg(A_i < B_i) \rightarrow (A_{i-1} \le B_{i-1})$ recursively apply, starting from $i = n$.
- Interval vector **A** and **B** are in a non-increasing order,
  $\mathbf{A} \succ \mathbf{B}$, where $\mathbf{A} = (A_1, A_2, \cdots, A_n), \mathbf{B} = (B_1, B_2, \cdots, B_n)$, if $A_n \ge B_n$, and
  $\neg(A_i > B_i) \rightarrow (A_{i-1} \ge B_{i-1})$ recursively apply, starting from $i = n$.
- $\text{maxwid}(\mathbf{A}) = \max_i(\text{wid}(A_i))$, where $\mathbf{A} = (A_1, A_2, \cdots, A_n)$.
- $\text{minwid}(\mathbf{A}) = \min_i(\text{wid}(A_i))$, where $\mathbf{A} = (A_1, A_2, \cdots, A_n)$.

3.3 Geometry Description

With the inherent capability of modeling variation, NICR has some special properties that make it different from current geometric modeling schemes. It models uncertainty and inexactness in the process of design. As the available ranges of parameters are narrowed down gradually, uncertainty is ruled out and decisions can be made throughout the design process until design is finalized. Changing current constraints or adding extra constraints would lead to different geometries. As illustrated in Figure 2, the shape of a 2D rectangular object may vary based on coordinates of four corner points within their allowable intervals.

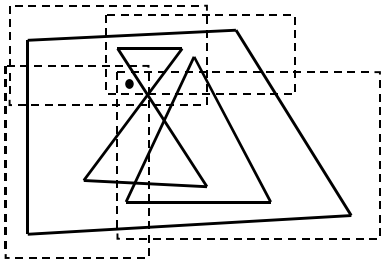

Figure 2: Constraint-driven geometry in interval modeling

Current parametric modeling scheme has strict requirements on the number of constraints. Only well-constrained geometry can be properly solved. The concept of under-constrained geometry in traditional parametric or variational design is not critical in NICR. *Soft* constraint is applied to geometry implicitly at every step of value specifications. The effect of adding more constraints is to reduce the allowable region of geometric entities systematically so that the final geometry can be fixed.

Over-constrained situation is also allowed in NICR. As illustrated in Figure 3, if the geometric constraints in a bracket design are specified as: the position of P_0; distances between P_0 and P_1, P_1 and P_2, P_2 and P_3, and P_3 and P_0; L_0 is perpendicular to L_1 as well as to L_3; and L_0 is horizontal, current CAD systems will complain that this geometry is over-constrained,

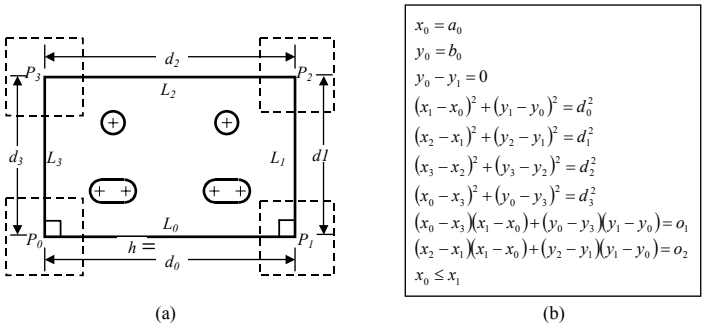

Figure 3: An example of over-constrained geometry in bracket design

In NICR, only those constraints that cause no feasible regions generate conflicts. Intervals loosen the current requirement on the number of constraints and give a different view of specifying parameters. Some of the previous over-constrained problems will no longer be over-constrained.

4. Solving interval constraints

To incorporate interval geometric modeling methodology into current CAD systems, several fundamental issues related to geometric computation should be addressed. These include linear and nonlinear equation representation and solution, which are essential for transformation operation, surface intersection, and constraint solving, etc. The process of solving systems of equations or inequalities is also called contraction. It starts with initial values of intervals, which are rough estimates of variable values. Subintervals that do not contain solutions are then eliminated, and intervals are "contracted". This process proceeds iteratively until there is no further improvement. Interval operations involve more steps and procedures than real arithmetic operations. Time and space efficient algorithms for solving interval constraint are needed.

4.1 Interval Linear Equations

Commonly used numerical methods for solving real-value linear equations can be extended to solve interval-value linear equations, such as Gaussian elimination and triangular factorization. But matrix-based methods only solve well-constrained problems. In contrast, iteration-based methods such as Jacobi iteration and Gauss-Seidel iteration have no requirement on the number of constraints. An algorithm for solving linear equations with nominal intervals is presented here, which is an extension of the Gauss-Seidel method, as listed in Figure 4. Different from methods of Alefeld and Herzberger [29], and Hansen and Sengupta [30], under-constrained and over-constrained linear systems are the major considerations here.

To solve

$$\sum_{j=1}^{n} A_{ij} X_j = Y_i \quad i = 1,2,...m \tag{1}$$

where X_1, X_2, …, X_n are interval variables, A_{ij} is interval constant for each i and j, and Y_1, Y_2, …, Y_m are interval constants. If an empty interval is derived during the process, there is no solution within the given initial intervals.

```
INPUT:  Interval matrix A
        Interval vector Y
OUTPUT: Interval vector X

Interval V
int i, j, k
REPEAT until stop criterion is met
    FOR each 1 <= i <= m
        FOR each 1 <= j <= n
            IF Aij=0
                continue next j iteration
            ENDIF
            V = 0
            FOR each 1<=k<j
                V = V+Aik*Xk
            ENDFOR
            FOR each j+1<=k<=n
                V = V+Aik*Xk
            ENDFOR
            V = (Yi - V)/Aij
            Xj = Xj ∩ V
        ENDFOR
    ENDFOR
```

Figure 4: Algorithm of extended Gauss-Seidel method for solving linear equations (1)

4.2 Interval Nonlinear Equations

Nonlinear equation systems can be solved by the fix-point method, forward-backward propagation, Newton's method, and Krawczyk method, etc. Given the requirement that a constraint solving system should be flexible on the number of constraints yet with fast convergence, a linear enclosure method is presented here. This algorithm is more general than Kolev's method [31, 32]. Kolev's method only considers the degenerated case where the right-hand sides of constants are all *0*s. The evaluation based on linear enclosure has sharper bounds than the one based on the interval Newton's method if the widths of intervals are non-trivial. Methods using coefficient matrix inverse operation are not applicable for under-constrained and over-constrained problems. Let us consider the interval nonlinear equation system

$$F_i(\mathbf{X}) = C_i \quad i = 1,2,...l\,, \tag{2}$$

where $\mathbf{X}$ is the interval variable vector $[X_1, X_2, \ldots, X_n]^{\mathrm{T}}$ and C_i is a constant interval. The following steps are needed to solve the system:

STEP 1: Transform each equation of (2) to separable form to eliminate dependency among variables;

STEP 2: Find the linear enclosure of each of the univariate nonlinear functions and form a linear equation system;

STEP 3: Solve the linear system by the algorithm of Section 4;

STEP 4: If stopping criterion is met, stop. Otherwise, repeat from STEP 2 to STEP 4.

STEP 1:

According to Yamamura's algorithm [33], functions that are composed of four basic arithmetic operations (+, –, ×, /), unary operations (sin, exp, log, sqrt, etc.), and the power operation (^) can be transformed into the separable form by introducing necessary functions. For example, $f = f_1 \times f_2$ can be transformed to $f = (y^2 - f_1^2 - f_2^2)/2$ and $y = f_1 + f_2$; $f = f_1 / f_2$ can be transformed to $f = (y^2 - f_1^2 - 1/f_2^2)/2$ and $y = f_1 + 1/f_2$; and $f = (f_1)^{f2}$ can be transformed to $f = exp(y_1)$, $y_1 = (y_2^2 - (log(f_1))^2 - f_2^2)/2$, and $y_2 = log(f_1) + f_2$. In geometric modeling, most of constraints/functions can be transformed into the separable form.

Thus equations (2) can be transformed to

$$\sum_{j=0}^{n} f_{ij}\left(X_j\right) = D_i \quad i = 1,2,...,m\,, \tag{3}$$

where $X_1, X_2, \ldots, X_n$ are interval variables and $D_1, D_2, \ldots, D_m$ are interval constants.

STEP 2:
Linear enclosure of $f_{ij}(x_j)$ is found within the initial interval of $X_j^{(0)}$ for each i and j as follows. Let $X_j^{(0)} = [x_L^j, x_N^j, x_U^j]$, we can have

$$f_{ij}^S = f_{ij}\left(x_L^j\right), \text{ and} \tag{4}$$

$$f_{ij}^T = f_{ij}\left(x_U^j\right). \tag{5}$$

Let

$$a_{ij} = \frac{f_{ij}^T - f_{ij}^S}{x_U^j - x_L^j}. \tag{6}$$

The *linear enclosure* of $f_{ij}(x_j)$ can be defined as

$$E_{ij}(x) = B_{ij} + a_{ij}x \quad for\ x \in X_j^{(0)}, \tag{7}$$

such that

$$f_{ij}(x) \in E_{ij}(x) \quad for\ \forall x \in X_j^{(0)}, \tag{8}$$

as illustrated in Figure 5.

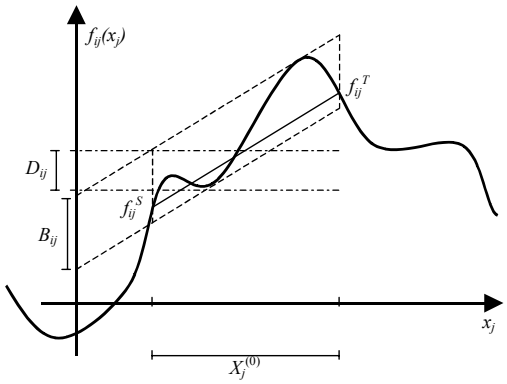

Figure 5: Linear enclosure of nonlinear interval function

To find a B_{ij} with the minimum width with the given a_{ij}, derivation of $f_{ij}(x)$ is used if $f_{ij}(x)$ is continuous and differentiable within interval $X_j^{(0)}$. The problem is reduced to solve real value nonlinear equation and find out solutions of

$$f'_{ij}(x) = a_{ij} \quad for\ x \in X_j^{(0)}. \tag{9}$$

Given that $f_{ij}(x)$ is continuous and differentiable for most geometric relations, equations (9) have at least one solution. The Secant method can be used to solve the equation efficiently. Having been transformed to the separable form, $f'_{ij}(x)$ is a univariate polynomial function or a function with unary operations for most geometric constraints. For polynomial functions, roots can be isolated within disjointed intervals individually based on Descartes' rule of signs before equations are solved. Descartes' bound gives the upper bound of the number of positive roots of a polynomial. Once polynomial functions are solved, solutions to unary functions such as *sin* and *cos* can be easily found.

Let $P(x)$ be a polynomial with real coefficients, the following transformations are defined:

- (*Reverse transformation*): $R[P(x)] = x^n P(1/x)$ where n is the degree of P.
- (*Translation transformation*): $T_t[P(x)] = P(x+t)$ for $t \in \boldsymbol{R}$.

- (*Homothetic transformation*): $H_c[P(x)] = P(cx)$ for $c \in \boldsymbol{R}$.

Based on the algorithm of Collins *et al*. [34, 35], $P_{ij}(x)$ for $x \in X_j^{(0)}$ is transformed to $P_{ij}^{0}(x)$ for $x \in [0, 1]$ by $P_{ij}^{0}(x) = H_{b-a}[T_a[P_{ij}(x)]]$ where a is the lower bound of X_j^0 while b is the upper bound of X_j^0. The roots of $P_{ij}(x)$ for $x \in X_j^0$ have one-to-one correspondence with the roots of $P_{ij}^{0}(x)$ for $x \in [0, 1]$. A list of root intervals or exact roots can be obtained by calling *RootIsolation*(P_{ij}^{0}, 0, 0) listed in Figure 6. For each root interval or exact root with information of (*depth, index*) in the list, there is an corresponding $x \in [\frac{(b-a)index}{2^{depth}} + a, \frac{(b-a)(index+1)}{2^{depth}} + a]$ for root intervals or $x = \frac{(b-a)index}{2^{depth}} + a$ for exact roots such that $P_{ij}(x)=0$.

```
INPUT:  Polynomial P with n degree
        int depth
        int index
OUTPUT: RootIntervalList

IF P(0) = 0
    RootIntervalList.addExactRoot(depth, index)
ENDIF
IF P(1) = 0
    RootIntervalList.addExactRoot(depth, index+1)
ENDIF
Polynomial Q = T1[R(P)]
IF DecartesBound(Q) = 1
    RootIntervalList.addRootInterval(depth, index)
ELSEIF DecartesBound(Q) >= 2
    Polynomial P1 = 2^n H1/2[P]
    RootIsolation(P1, depth+1, 2*index)
    Polynomial P2 = T1[P1]
    RootIsolation(P2, depth+1, 2*index+1)
ENDIF
```

Figure 6: `RootIsolation` procedure based on Descartes' rule of signs

Thus, interval $X_j^{(0)}$ can be subdivided into small intervals containing an individual root. Let $g(x) = f_{ij}'(x) - a_{ij}$. Solutions to (9) within interval $X_j^{(0)}$ can be found by (10) iteratively.

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{g(x_n) - g(x_{n-1})} g(x_n) \quad n = 1,2,3,... \tag{10}$$

Suppose x_{jp} (p=1, 2, …, P) is the p^{th} solution of equation (9), and $x_{j0} = x_L^j$. Let $B_{ij} = [b_L^{ij}, b_N^{ij}, b_U^{ij}]$, where

$$b_U^{ij} = \max_p \left\{ f_{ij}(x_{jp}) - a_{ij} x_{jp}, p = 0,1,2,..., P \right\}, \tag{11a}$$

$$b_N^{ij} = f_{ij}(x_{j0}) - a_{ij} x_{j0}, \tag{11b}$$

$$b_L^{ij} = \min_p \left\{ f_{ij}(x_{jp}) - a_{ij} x_{jp}, p = 0,1,2,..., P \right\}. \tag{11c}$$

From equation (8), we have

$$f_{ij}(X_j) \subseteq E_{ij}(X_j) \quad for\ i = 1,2,...,m, \tag{12}$$

thus,

$$\sum_{j=1}^{n} f_{ij}(X_j) \subseteq \sum_{j=1}^{n} E_{ij}(X_j) = \sum_{j=1}^{n} (B_{ij} + a_{ij} X_j) \quad for\ i = 1,2,...,m. \tag{13}$$

STEP 3:
Solving (3) thus is reduced to solving linear equations (14) iteratively.

$$\sum_{j=1}^{n}\left(B_{ij}+a_{ij}X_j\right)=D_i \quad for\ i=1,2,...,m. \tag{14}$$

This linear system can be solved using the algorithm described in Section 4. Because the coefficient a_{ij}'s are degenerated intervals, only one iteration is needed to solve the linear equations. Suppose Y_j is the j^{th} variable solution of (14) in the k^{th} iteration. By formula (15), the initial value of X_j in the $(k+1)^{th}$ iteration is calculated. If an empty interval is derived, the original system has no solution within the given initial intervals ($X_1^{(0)}, X_2^{(0)}, ..., X_n^{(0)}$).

$$X_j^{(k+1)} = X_j^{(k)} \cap Y_j \quad for\ j=1,2,...,n. \tag{15}$$

STEP 4:
When the stopping criterion, such as the width of intervals has no further improvement (16a) or the intervals are sharp enough (16b), is met, the iteration is stopped. Otherwise, go back to (3) to find out the new linear enclosures within the updated intervals and repeat the procedure starting from STEP 2.

$$\left|\sum_{j=1}^{n}\mathrm{wid}(X_j^{(k+1)})-\sum_{j=1}^{n}\mathrm{wid}(X_j^{(k)})\right|<\varepsilon_1 \quad \text{for iteration } k. \tag{16a}$$

$$\left|\sum_{j=1}^{n}\mathrm{wid}(X_j^{(k)})\right|<\varepsilon_2 \quad \text{for iteration } k. \tag{16b}$$

4.2 Interval Inequalities

Inequalities can be solved by the methods for equations. Consider a set of linear or nonlinear inequalities

$$F_i(\mathbf{X})\le C_i \quad i=1,2,...l, \tag{17}$$

where $\mathbf{X}$ is the interval variable vector and C_i is a constant interval, inequalities are transformed into equations

$$F_i(\mathbf{X})+S_i=C_i \quad i=1,2,...l, \tag{18}$$

where S_i is a slack variable with initial value of $[0,0,+\infty]$. Similarly,

$$F_i(\mathbf{X})\ge C_i \quad i=1,2,...l, \tag{19}$$

can be transformed into

$$F_i(\mathbf{X})+S_i=C_i \quad i=1,2,...l, \tag{20}$$

where S_i is a slack variable with initial value of $[-\infty,0,0]$. Inequalities can be easily integrated into systems of equalities, which is another property of interval constraint representation.

5. Design refinement

One important aspect related to interval representation of allowance is the over estimation of allowance. An interval vector simply encloses the allowable region by a hyper cube, which usually includes some infeasible region. During the function evaluation, inclusion functions are likely to give a set that is larger than the actual solution set. Design refinement is needed to generate more delicate design if desirable details have not been achieved yet. There are two ways to refine design: interval subdivision and constraint re-specification. Interval subdivision is to divide existing interval regions into unions of subintervals to achieve the refined views of current design. Constraint re-specification is to modify some of constraints or to add extra valid constraints to contract feasible regions.

5.1 Interval subdivision

Interval subdivision (also called *subpaving*) substitutes an interval vector with multiple interval vectors such that the corresponding real space region is subdivided into multiple smaller regions to cover the actual solution set more compactly. As shown in Figure 7, the interval vector **X** can be bisected recursively and subintervals are tested individually if they belong to the actual solution set. The actual solution set is approximated by the union of subinterval regions.

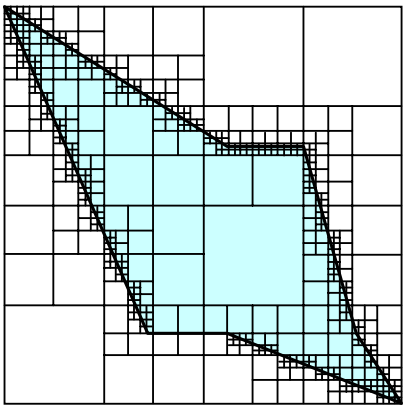

Figure 7: Two-dimensional interval vector subdivision

To represent subdivision of intervals concisely, a *power interval* can be used. An *n*-dimensional power interval with degrees of *m*, denoted as $\mathbf{P}^{(m,\,n)}$, is an ordered list of *m* non-overlapped interval vectors of *n*-dimensional, i.e., $\mathbf{P}^{(m,\,n)} = [\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_m]$, where $\mathbf{X}_i \in \boldsymbol{IR}^n$ ($i = 1, \ldots, m$), $\text{minwid}(\mathbf{X}_i \cap \mathbf{X}_j) = 0$ ($i \neq j$), and $\mathbf{X}_i \prec \mathbf{X}_{i+1}$ ($i = 1, \ldots, m-1$).

Consider a design problem $\boldsymbol{f}(\mathbf{X}) = \mathbf{Y}$. The target is to find the actual solution set $\mathbf{S} \subseteq \mathbf{X}$ with the minimal size such that $\boldsymbol{f}(\mathbf{S}) = \mathbf{Y}$. Interval arithmetic only gives a valid solution **D** with $\boldsymbol{f}(\mathbf{D}) \supseteq \mathbf{Y}$. If the valid solution is represented by power intervals, refinement can be looked as degree elevation of power intervals. If the original solution to a problem is found as an *n*-dimensional vector $\mathbf{X} = [X_1, X_2, \ldots, X_n]$, the corresponding power interval is $\mathbf{P}_{(0)}^{(1,n)} = [\mathbf{X}]$. One elevation operation will bisect **X**, with each interval vector being deleted and new subintervals inserted. Feasibility of each new subinterval then can be tested. The procedure of subdivision is shown in Figure 8.

```
INPUT:  Power Interval P^(m,n)
        Interval vector Y
        Mapping function f
OUTPUT: Power Interval P^(k,n)

IF stop criterion is met
    Return P^(m,n)
ELSE
    j = m*n
    Q^(j,n) = Bisect(P^(m,n))
    FOR 1 <= i <= m*n
        IF f(Q^(j,n)(i)) ⊄ Y
           Delete(Q^(j,n)(i))
        ENDIF
    ENDFOR
    Subdivide(Q^(j,n),Y,f)
ENDIF
```

Figure 8: `Subdivide` procedure for power interval elevation

5.2 Constraint Re-specification

Another way to contract a solution is to modify or add valid constraints to narrow down feasible regions. Feasibility and effectiveness should be considered simultaneously. Constraint modification depends on sensitivity analysis, while adding constraints is largely dependent on user's specification. One basic question is how to differentiate *active* and *inactive* constraints. Active constraints scope the actual range of solution while inactive constraints have certain level of slackness. At the beginning of interval computation, all constraints are active if a sufficiently large initial region is given. As the iteration proceeds, some constraints turn to be inactive. The decision of which constraints to be modified is based on the selection of active constraints.

Lemma: For a constraint set $\boldsymbol{p} = \{\boldsymbol{f}(\mathbf{X}) = \mathbf{Y} \text{ and } \boldsymbol{g}(\mathbf{X}) = \mathbf{Z}\}$, the subset $\boldsymbol{f}(\mathbf{X}) = \mathbf{Y}$ with respect to a solution $\mathbf{D} \subset \mathbf{X}$ is inactive if $\boldsymbol{f}(\mathbf{D}) \subset \mathbf{Y}$ and $\boldsymbol{g}(\mathbf{D}) \supseteq \mathbf{Z}$.

Proof:

Suppose $\mathbf{S}_1$ and $\mathbf{S}_2$ are actual solution sets of $\boldsymbol{f}$ and $\boldsymbol{g}$ respectively, and $\mathbf{S}$ is the actual solution set of $\boldsymbol{p}$. Given that $\boldsymbol{f}(\mathbf{S}_1) = \mathbf{Y}$ and $\boldsymbol{f}(\mathbf{D}) \subset \mathbf{Y}$, because of the property of inclusion monotonic, $\mathbf{S}_1 \supset \mathbf{D}$. Similarly, $\mathbf{D} \supseteq \mathbf{S}_2$. Thus, $\mathbf{S}_1 \supset \mathbf{S}_2$. □

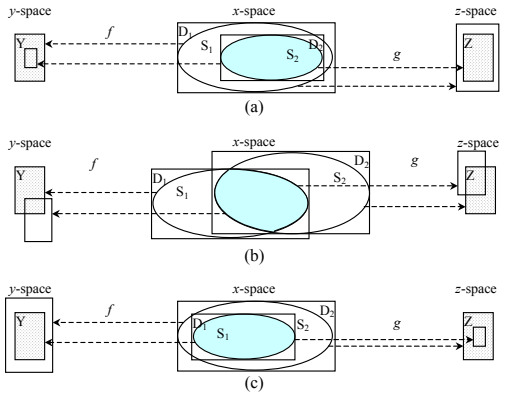

Figure 9: Relations of two constraint subsets

As illustrated by Figure 9, subset $\boldsymbol{f}$ is inactive and $\boldsymbol{g}$ is active in case (a); both $\boldsymbol{f}$ and $\boldsymbol{g}$ are active in case (b); and $\boldsymbol{f}$ is active and $\boldsymbol{g}$ is inactive in case (c).

6. A numerical example

The NICR kernel is implemented in C++ with an object-oriented programming style. The kernel includes the fundamental structure and arithmetic operations of the nominal intervals. It also includes the implementation of the algorithms described in previous sections for solving linear and nonlinear constraint systems as well as design refinement. The NICR kernel is integrated and tested in a geometric modeling system, which is based on ACIS® kernel.

As a demonstration, the design of the bracket in Figure 3 is used as a numerical example. The designer specifies the nominal value, lower bound, and upper bound of each coordinate and parameter. Geometric constraints are assigned to generate the outline of the bracket, which is over-constrained in the sense of the traditional parametric modeling. The interval geometric modeler then calculates the ranges of geometric points based on the algorithms of solving interval linear and nonlinear equations.

Figure 10 lists the constraint equations in Figure 3 (b), which are transformed to separable form. Based on the algorithm in Section 0, this over-constrained nonlinear equation system is solved. The numerical results are listed in Table 1.

$x_0 = a_0$	$y_2 - y_3 + v_3 = 0$
$y_0 = b_0$	$u_4^2 + v_4^2 = d_3^2$
$y_0 - y_1 = 0$	$-x_0 + x_3 + u_4 = 0$
$u_1^2 + v_1^2 = d_0^2$	$-y_0 + y_3 + v_4 = 0$
$x_0 - x_1 + u_1 = 0$	$u_1^2/2 + v_1^2/2 + u_4^2/2 + v_4^2/2 - w_1^2/2 - w_2^2/2 = o_1$
$y_0 - y_1 + v_1 = 0$	$-u_1 - u_4 + w_1 = 0$
$u_2^2 + v_2^2 = d_1^2$	$-v_1 - v_4 + w_2 = 0$
$x_1 - x_2 + u_2 = 0$	$u_1^2/2 + v_1^2/2 + u_2^2/2 + v_2^2/2 - w_3^2/2 - w_4^2/2 = o_2$
$y_1 - y_2 + v_2 = 0$	$-u_1 - u_2 + w_3 = 0$
$u_3^2 + v_3^2 = d_2^2$	$-v_1 - v_2 + w_4 = 0$
$x_2 - x_3 + u_3 = 0$	$-x_0 + x_1 = c$

Figure 10: Constraint equations of Figure 3 (b) in separable form

Table 1: Numerical results of the bracket example

Variables	**Initial values**	**Final values (after 20 iterations)**	**Descriptions**
	X_0 = [0, 0.25, 0.5]	*X_0 = [0, 0, 0]*	*x* coordinate of P_0
	Y_0 = [0, 0.25, 0.5]	*Y_0 = [0, 0, 0]*	*y* coordinate of P_0
	X_1 = [0.5, 0.75, 1]	*X_1 = [0.5, 0.505012, 0.510024]*	*x* coordinate of P_1
	Y_1 = [0, 0.25, 0.5]	*Y_1 = [0, 0, 0]*	*y* coordinate of P_1
	X_2 = [0.5, 0.75, 1]	*X_2 = [0.5, 0.516686, 0.533372]*	*x* coordinate of P_2
	Y_2 = [0, 0.25, 0.5]	*Y_2 = [0.23886, 0.249714, 0.260569]*	*y* coordinate of P_2
	X_3 = [0, 0.25, 0.5]	*X_3 = [0, 0.0116355, 0.0232709]*	*x* coordinate of P_3
	Y_3 = [0, 0.25, 0.5]	*Y_3 = [0.238869, 0.249677, 0.260485]*	*y* coordinate of P_3
Parameters	*A_0 = [0, 0, 0]*		fixed position of P_0
	B_0 = [0, 0, 0]		fixed position of P_0
	D_0 = [0.49, 0.50, 0.51]		distance d_0
	D_1 = [0.24, 0.25, 0.26]		distance d_1
	D_2 = [0.49, 0.50, 0.51]		distance d_2
	D_3 = [0.24, 0.25, 0.26]		distance d_3
	O_1 = [-0.001, 0, 0.001]		perpendicularity
	O_2 = [-0.001, 0, 0.001]		perpendicularity

Figure 11 shows the convergence speed when solving the nonlinear equations. After 15 iterations, the widths of intervals are stabilized.

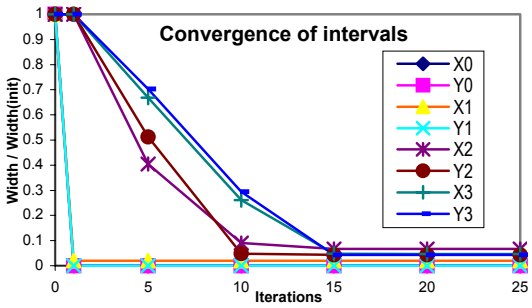

Figure 11: Convergence of Interval calculation in the bracket example

7. Conclusion

This paper presents a nominal interval constraint representation scheme based on nominal interval for CAD applications. It provides a generic numerical parameter scheme to represent inexactness at early design stages, uncertainty in detailed design, as well as the boundary information for design optimization. It relaxes the restriction of under-constrained and over-constrained situations for variational geometry. A generalized iterative nonlinear constraint solving method based on linear enclosure is developed for fast convergence. Inequalities are transformed into equations and can be solved uniformly. Interval subdivision and constraint re-specification methods are developed for design refinement. Active and inactive constraints are differentiated in sensitivity analysis.

Acknowledgement

The author would like to thank Dr. G. William Walster and Mark Woodyard of Sun Microsystems and Nate Hayes of Sunfish Studio for discussions and comments, as well as the anonymous referees for the suggestions.

References

1. Mudur, S.P. and Koparkar, P.A., “Interval Methods for Processing Geometric Objects”, *IEEE Computer Graphics and Applications*, Vol.4, No.2 (February 1984), pp.7-17
2. Toth, D.L., “On Ray Tracing Parametric Surfaces”, *Computer Graphics*, Vol.19, No.3 (July 1985), pp.171-179
3. Kalra, D. and Barr, A.H., “Guaranteed Ray Intersections with Implicit Surfaces”, *Computer Graphics*, Vol.23, No.3 (July 1989), pp.297-304
4. Moore, M. and Wilhelms, J., “Collision Detection and Response for Computer Animation”, *Computer Graphics*, Vol.22, No.4 (August 1988), pp.289-298
5. Von Herzen, B., Barr, A.H. and Zatz, H.R., “Geometric Collisions for Time-Dependent Parametric Surfaces”, *Computer Graphics*, Vol.24, No.4 (August 1990), pp.39-48
6. Duff, T., “Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry”, *Computer Graphics*, Vol.26, No.2 (July 1992), pp.131-138
7. Snyder, J., *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design Using Interval Analysis* (Cambridge, MA: Academic Press, 1992)

8. Snyder, J.M., Woodbury, A.R., Fleischer, K., Currin, B., and Barr, A.H., "Interval Methods for Multi-Point Collisions Between Time-Dependant Curved Surfaces", *ACM Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, , September 1993, New York, NY*, pp.321-334
9. Bliek, C., "Computer methods for design automation", *Ph.D. Thesis,* Massachusetts Institute of Technology, 1992
10. Rao, S.S. and Berke, L., "Analysis of uncertain structural systems using interval analysis", *AIAA Journal*, Vol.35, No.4, pp.727-735
11. Rao, S.S. and Cao, L. "Optimum design of mechanical systems involving interval parameters", *ASME Journal of Mechanical Design*, Vol.124, (September 2002), pp.465-472
12. Muhanna, R.L. and Mullen, R.L., "Formulation of Fuzzy Finite-Element Methods for Solid Mechanics Problems", *Computer-Aided Civil and Infrastructure Engineering*, Vol.14, (1999), pp. 107-117
13. Muhanna, R.L. and Mullen, R.L., "Uncertainty in mechanics problems --- interval-based approach", *ASCE Journal of Engineering Mechanics*, Vol.127, No.6 (June 2001), pp. 557-566
14. Finch, W.W. and Ward, A.C., "A set-based system for eliminating infeasible designs in engineering problems dominated by uncertainty", *ASME Proceedings of DETC97/dtm-3886*, Sept.14-17, 1997, Sacramento, CA, USA
15. Sederberg, T.W. and Farouki, R.T., "Approximation by Interval Bezier Curves", *IEEE Computer Graphics and Applications*, Vol.12, No.5 (September 1992), pp.87-95
16. Maekawa, T. and Patrikalakis, N.M., "Computation of Singularities and Intersections of Offsets of Planar Curves", *Computer Aided Geometric Design*, Vol.10, No.5 (Oct. 1993), pp.407-429
17. Maekawa, T. and Patrikalakis, N.M., "Interrogation of differential Geometry Properties for Design and Manufacture", *The Visual Computer*, Vol.10, No.4 (March 1994), pp.216-237
18. Hu, C.Y., Patrikalakis, N.M., and Ye, X., "Robust Interval Solid Modeling, Part II: Boundary Evaluation", *Computer-Aided Design*, Vol.28, No.10 (October, 1996), pp.819-830
19. Hu, C.-Y., Maekawa, T., Patrikalakis, N.M., and Ye, X., "Robust Interval Algorithm for Surface Intersections", *Computer-Aided Design*, Vol.29, No.9 (1997), pp.617-627
20. Abrams, S.L., Cho, W., Hu, C.Y., Maekawa, T., Patrikalakis, N.M., Sherbrooke, E.C., and Ye, X., "Efficient and Reliable Methods for Rounded-Interval Arithmetic", *Computer-Aided Design*, Vol.30, No.8 (July 1998), pp.657-665
21. Shen, G. and Patrikalakis, N.M., "Numerical and Geometric Properties of Interval B-Splines", *International Journal of Shape Modeling*, Vol.4 (1998), pp.31-62
22. Tuohy, S.T., Maekawa, T., Shen G., and Patrikalakis, N.M., "Approximation of Measured Data with Interval B-Splines", *Computer-Aided Design*, Vol.29, No.11 (1997), pp.791-799
23. Wallner, J., Krasauskas, R., and Pottmann, H., "Error Propagation in Geometric Constructions", *Computer-Aided Design*, Vol.32, No.11 (September, 2000), pp.631-641
24. Chen, F. and Lou, W., "Degree Reduction of Interval Bezier Curves", *Computer-Aided Design*, Vol.32, No.10 (Sept. 2000), pp.571-582
25. Lin, H., Liu, L., and Wang, G., "Boundary Evaluation for Interval Bezier Curve", *Computer-Aided Design*, Vol.34, No.9 (August 2002), pp.637-646
26. Ratschek, H. and Rokne, J., *New Computer Methods for Global Optimization* (New York: Ellis Horwood Limited, 1988), Ch.2, pp.28-29
27. Hansen, E., "An Overview of Global Optimization Using Interval Analysis", In: Moore, R.E., eds., *Reliability in Computing: The Role of Interval Methods in Scientific Computing* (Boston: Academic Press, 1988), , pp.289-305

28. Nickel, K., "How to Fight the Wrapping Effect", In: Nickel, K., eds., *Proceedings of the International Symposium on Interval Mathematics, September 23-26, 1985, Freiburg i. Br., Germany*, pp. 121-132
29. Alefeld, G. and Herzberger, J., *Introduction to Interval Computations* (New York: Academic Press, 1983)
30. Hansen, E. and Walster, G.W., *Global Optimization Using Interval Analysis* (New York: Marcel Dekker, 2004)
31. Kolev, L.V., "A New Method for Global Solution of Systems of Non-linear Equations", *Reliable Computing*, Vol.4, No.2 (May, 1998), pp.125-146
32. Kolev, L.V., "Automatic Computation of a Linear Interval Enclosure", *Reliable Computing*, Vol.7, No.1 (February, 2001), pp.17-28
33. Yamamura, K., "An Algorithm for Representing Functions of Many Variables by Superpositions of Functions of One Variable and Addition", *IEEE transactions on Circuits and Systems – I: Fundamental Theory and Application*, Vol.43, No.4 (April, 1996), pp.338-340
34. Collins, G.E. and Johnson, J.R., "Quantifier Elimination and the Sign Variation Method for Real Root Isolation", *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, July 17-19, 1989 Portland, Oregon*, pp.264-271
35. Collins, G.E. and Akritas, A.G., "Polynomial Real Root Isolation Using Descarte's Rule of Signs", *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, August 10-12, 1976, Yorktown Heights, New York*, pp.272-275