Engineering Design Calculations with Fuzzy Parameters

Kristin L. Wood*
Department of Mechanical Engineering
The University of Texas at Austin

Kevin N. Otto†
and Erik K. Antonsson‡
Engineering Design Research Laboratory
Division of Engineering and Applied Science
California Institute of Technology

Received: March 1, 1991
Revised: July 13, 1992

Abstract

Uncertainty in engineering analysis usually pertains to *stochastic* uncertainty, *i.e.*, variance in product or process parameters characterized by probability (uncertainty in truth). Methods for calculating under stochastic uncertainty are well documented. It has been proposed by the authors that other forms of uncertainty exist in engineering design. *Imprecision*, or the concept of uncertainty in choice, is one such form. This paper considers real-time techniques for calculating with imprecise parameters. These techniques utilize interval mathematics and the notion of α-cuts from the fuzzy calculus. The extremes or anomalies of the techniques are also investigated, particularly the evaluation of singular or multi-valued functions. It will be shown that realistic engineering functions can be used in imprecision calculations, with reasonable computational performance.

1 Introduction

1.1 Uncertainty in Engineering Design

Engineering design may be viewed as a goal-directed process, with the generally recognized stages of synthesis, evaluation, and decision [1, 3]. Other authors [19, 21] define engineering design, instead, as a mapping of the functional description of an artifact or process to a form description. While these definitions focus on different aspects of design as a human activity, each implicitly assumes the need to compare and contrast alternative designs.

During the preliminary design stages, design alternatives are usually described only approximately or imprecisely. Precise geometry, material properties, manufacturing processes,

*Assistant Professor
†Graduate Research Assistant
‡Associate Professor, Mail Code: 104-44, Caltech, Pasadena, CA 91125

etc., are not specified or known. Conventional computer-aided design methods require highly precise descriptions in order to operate, making them difficult or impossible to use during the early design process. This research focuses on the development of methods for representing and manipulating imprecise descriptions of designs to provide the designer with more information to compare alternatives during the preliminary design phase.

Using this premise, what types of uncertainty are encountered in preliminary design? How can we compute under uncertainty? Engineering design uncertainty is comprised of *design imprecision* (uncertainty in choosing among alternatives) and *stochastic uncertainty* (usually pertaining to manufacturing and measurement limitations) [23]. Stochastic uncertainty arises from a lack of exact knowledge of a parameter due to some process the designer has no direct control or choice over. A manufacturing tolerance is an example of such an uncertainty. Stochastic uncertainties are usually represented and manipulated using the probability calculus (Cox [5], Jeffreys [11]).

Design imprecision, on the other hand, concerns the choice of design parameter values used to describe an artifact or process. For example, in the preliminary design of equipment for processing solid epoxy resin, pumps feed molten resin to an extrusion/applicator head. The applicator head conducts the molten resin to a belt assembly, which compresses and conveys the resin in sheet form through a chilling process. A thrasher then chops the sheets into "flakes" for storage and transportation. Although the process layout is well defined in this case, the applicator and thrasher designs are imprecisely described, depending on the geometry, thermal, fluid, and other material properties of the resin. Given alternative designs for the applicator and thrasher, how can these alternatives be compared with respect to the imprecision?

We use the designer's preference to quantify the imprecision with which design parameters are known. Preference, as used here, denotes either subjective or objective information that may be quantified and included in the evaluation of design alternatives. For instance, an applicator design for the resin flaking process may include a wide range of material choices for the applicator's head structural supports. Preference of the material choice will depend on the corrosion and thermal expansion properties. The following section describes a general method for representing and manipulating this type of information in engineering design calculations.

1.2 General Approach

In this research, we pursue a *semi-automated* approach to the design of such complex entities as a process. That is, we propose to aid the designer's ability to make decisions in the preliminary design stage. The salient feature of this part of the preliminary design stage is uncertainty in the model used. We model and manipulate such uncertainties in a computer-assisted environment, under the hypothesis that doing so will allow the designer to make faster and more-informed decisions.

The model implies the designer arrives at candidate solutions by the designer's own creative inferencing, using, for example, traditional techniques of divergent thinking [1, 22]. Yet these models so determined are not completely described, and the designer cannot determine which candidate offers the best alternative to pursue into the subsequent, expensive detailed-design stage.

Graphically, our computational model of the design process is depicted in Figure 1, called the *Method of Imprecision*. At the start of the process, the designer suggests possible solutions. These are subsequently transformed into formal candidate solution models upon which computations can be performed. Each model includes *design parameters*, d_i, variables to be determined during the design process, and *performance parameters*, p_j, variables (parame-

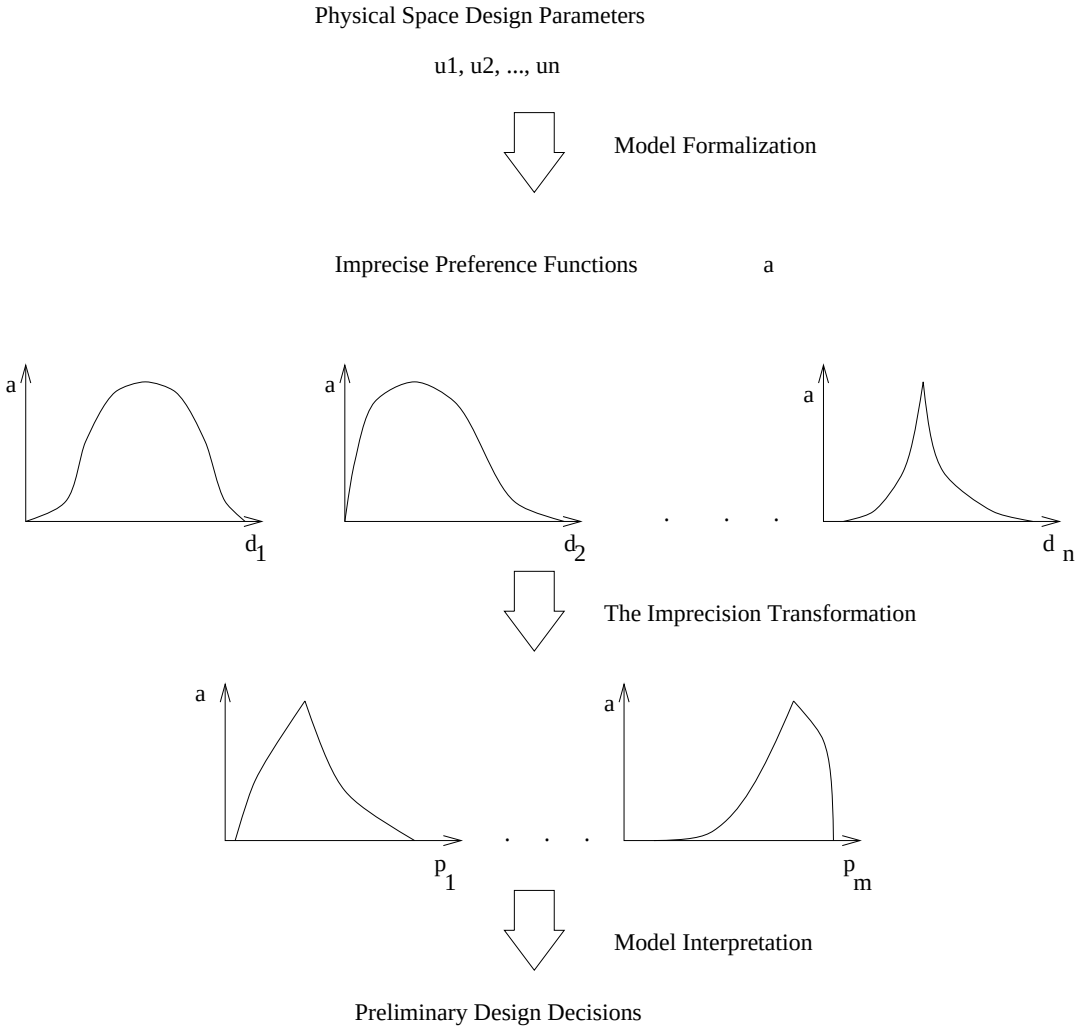

Figure 1: The Method of Imprecision.

terized as functions of the design parameters) used to give an indication of the performance achievable by the design.

Having determined conceptual models of the proposed candidates, the designer then specifies the design parameters imprecisely. The degree to which a designer prefers a value of a design parameter shall be denoted μ, termed *preference*. This is a normalized ranking of the designer's belief that the value will be used in the design. The specification of μ (how much a design parameter value will possibly satisfy the design) is based on the same foundations as used by the designer to hypothesize typical values. Usually this information comes from objective sources such as corrosion data, geometric constraints, or material properties. It also comes from subjective sources as well, such as the belief in availability of components or imprecise knowledge of reliability. Imprecise specifications explicitly represent these forms, allowing the designer to justify preliminary design decisions.

Having performed this formalization, the *Imprecision Transformation* is invoked to calculate the imprecise achievable performance. This calculation is performed over the *entire set* of imprecise values specified, allowing the designer to make more informative evaluations. It has been shown that using the fuzzy mathematics is a well suited method for imprecise design calculations [25], rather than, for example, using the probability mathematics. The fuzzy mathematics preserves peak preferences on the performance parameters with peak preferences on the design parameters, a required feature for the transformation. The imprecision transformation is based on Zadeh's *extension principle* [27] to calculate the preference of any performance parameter value p_j:

$$\mu(p_j) = \begin{cases} \sup\{\min\{\mu(d_1), \ldots, \mu(d_N)\} \mid d_1, \ldots, d_N : p_j = f_j(d_1, \ldots, d_N)\} \\ 0 \quad \text{if } \{d_1, \ldots, d_N \mid p_j = f_j(d_1, \ldots, d_N)\} = \emptyset \end{cases} \tag{1}$$

where $d_1, \ldots, d_N$ are values of the imprecise design parameters, $p_j = f_j(d_1, \ldots, d_N)$ is a performance parameter expressed as a function of the design parameters, and $\mu(d_i)$ is the degree of preference for the design parameter value d_i used.

After having made the performance parameter preference calculations, the designer can observe the imprecise performance achievable and proceed to judge the candidates. With single performance parameter designs, this process involves determining the preference of the performance parameter at the one required performance value (the *functional requirement*). With multiple performance parameter designs, a *figure of merit* must be introduced to combine the different performance parameter preferences at each performance parameter's functional requirement level. This involves trading off conflicting goals and is beyond the scope of this document, but the interested reader should refer to [23, 26].

The design paradigm presented so far assumes the preliminary design process can be carried out in a forward manner. Of course, there will be iteration among the steps presented. Observations in the achievable performance will induce changes in the specified preferences for design parameter values. Hence there is a need for real-time calculation of the performance parameter preferences induced from the design parameters. The next section will present a computational model for evaluating the preference functions of performance parameters, introduce a more efficient algorithm for a restricted class of problems, and introduce an new algorithm for more general functions involving extrema.

2 The Computational Model for Design Imprecision

2.1 Introduction

Previous publications [23, 24, 26] present a methodology by which uncertainties in preliminary engineering design can be represented and manipulated. This section discusses the component of the methodology that requires the computation of Zadeh's *extension principle* [27], *i.e.*, fuzzy arithmetic for the purpose of this study. The entire computational structure and requirements for implementing the methodology, including equation parsing, symbolic manipulation, and so on, is beyond the scope of this document.

2.2 Computation of the Extension Principle

Kaufmann and Gupta [12, 23] describe an analytical method of calculating a fuzzy output (application of Zadeh's *extension principle*) from imprecise inputs. This method is based on α-cuts [12] and interval arithmetic [14]. Although the method is straight-forward in its approach, the manipulation of symbols and the solution of expressions that include high order polynomials, both in the numerator and denominator (for extended division), make this method infeasible for computer-assisted design applications. This is compounded by the fact that the exact solution to the analytical application of the *extension principle* can be shown to be equivalent to an unwieldy non-linear programming problem [2]. A discrete numerical approach is therefore necessary to meet the computational requirements for handling many design parameters. This section will discuss a useful numerical technique, the LIA algorithm, and introduce a number of extensions.

2.2.1 The LIA Algorithm

In reviewing the literature, many discrete and analytical methods exist for carrying out extended operations with fuzzy sets (or fuzzy numbers). The Fuzzy Weighted Average (FWA) algorithm, as presented by W.M. Dong and F.S. Wong in [7], outlines a simple and efficient algorithm that is useful for carrying out engineering design calculations. We extend this algorithm below for generalized real functions of fuzzy variables, referring to the extended form as the *level interval algorithm* (LIA). Comparing the algorithm to the analytical method outlined in [12], LIA uses the interval analysis techniques as described; yet, LIA simplifies the process extensively by discretizing the membership functions of the input fuzzy numbers into a prescribed number of α-cuts. Performing interval analysis for each α-cut and combining the resultant intervals, the output is a discretized fuzzy set, the performance parameter output of input preference functions for the case of a design calculation. Dong and Wong also include a combinatorial interval analysis technique in order to avoid the problems of the multiple occurrence of variables for division and multiplication in an algebraic equation expression. A condensed version of the algorithm from [7] has been provided below (where the terminology has been changed to reflect the application to design calculations). Later sections will describe the implementation and extension of this algorithm; hence its inclusion in this paper.

There are conditions which must be satisfied for application of the algorithm: the preference functions must satisfy the normality and convexity conditions and must be continuous over $\tilde{d}$; no singularities of the functions can occur over $\tilde{d}$ (*i.e.*, no division by zero can occur, and no zero arguments can occur in $f_j(\vec{d})$ for each $\tilde{d}_i$ for the unary operations, such as the natural logarithm and the square root); and only monotonic regions of multi-valued functions,

Table 1: Design Parameter Data for Column Equation.

DPs (units)	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$
n	1.0	3.0	5.0
l/r	60.0	100.0	160.0
E (GPa)	75.0	150.0	225.0
K (simply-supported)	1.0	1.0	1.0

e.g., sine and cosine, are computed for a given $\tilde{d}_i$.[1]

The algorithm is as follows: for N real imprecise design parameters, $\tilde{d}_1, \ldots, \tilde{d}_N$, let d_i ($i \in [1, N]$) be an element of $\tilde{d}_i$. Given a performance parameter represented by the mapping

$$p \;=\; f(d_1, \ldots, d_N) \qquad\qquad \forall d_i \in \tilde{d}_i$$

respectively, let $\tilde{P}$ be the fuzzy output of the mapping. The following steps lead to the solution of $\tilde{P}$.

1. For each $\tilde{d}_i$, discretize the preference function into a number of α values, $\alpha_1, \ldots, \alpha_M$, where M is the number of steps in the discretization.

2. Determine the intervals for each parameter $\tilde{d}_i, i = 1, \ldots, N$ at each $\alpha-$cut, $\alpha_j, j = 1, \ldots, M$.

3. Using one end point from each of the N intervals for each α_j, combine the end points into an N-ary array such that 2^N distinct permutations exist for the array.

4. For each of the 2^N permutations, determine $p_k = f(d_1, \ldots, d_N), k = 1, \ldots, 2^N$. The resultant interval for the $\alpha-$cut, α_j, is then given by

$$P^{\alpha_j} = [\min(p_k), \max(p_k)].$$

For N imprecise design parameters and M discrete preference levels, the complexity of this algorithm is given by

$$H \sim M \cdot 2^{N-1} \cdot \kappa \tag{2}$$

where H equals the number of operations and κ equals the number of multiplications and divisions in $f(\vec{d})$. Figure 2 shows the results of the analytical application of the *extension principle* to the column-stress equation (Equation 3), along with the results using LIA.

$$\sigma_a = \frac{\pi^2 E}{n\left(\frac{Kl}{r}\right)^2}. \tag{3}$$

The input parameters for the maximum allowable stress σ_a (Equation 3) are triangular preference functions as listed in Table 1. No noticeable differences can be seen in the figure; however, minimal error is of course encountered due to discretization and the precision of the computer used.

[1] Reference [23] discusses methods for relaxing the continuity and monotonicity conditions of $f_j(\vec{d})$.

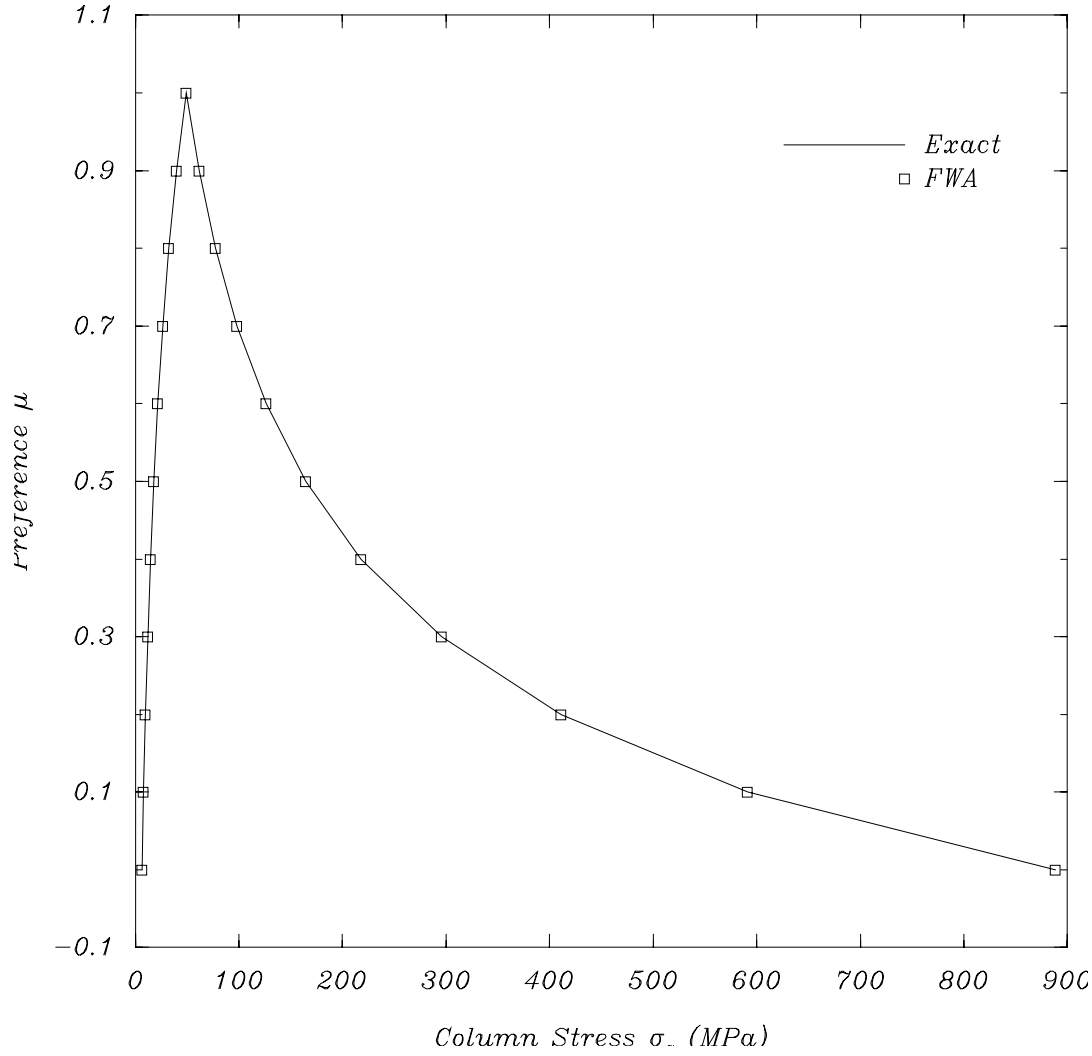

Figure 2: Example Column Calculation: Exact and LIA.

2.2.2 Implementation Scheme for LIA

For the LIA to be an efficient tool within real-time computer-assisted design, the implementation scheme must be considered in detail. A computationally efficient implementation of Dong and Wong's algorithm [7], written in pseudo-code, is shown in Figure 3.

The input array d contains the discretized elements for each input design parameter. For example, given two parameters, $\tilde{d}_1$ and $\tilde{d}_2$, and three α-cuts, $\alpha = 0.0$, $\alpha = 0.5$, and $\alpha = 1.0$, the d array is expressed as follows:

$$\begin{aligned} d = \quad & [d_{11}, d_{12}, d_{13}, d_{14}, d_{15}, d_{16}, \\ & d_{21}, d_{22}, d_{23}, d_{24}, d_{25}, d_{26}] \end{aligned}$$

where the first index corresponds to the parameter number and the second to the elements in the parameter's support due to the α-cuts. Figure 4 illustrates two (out of many possible) examples of fuzzy parameters (preference functions) that can be expressed in the d array.

Considering the implementation scheme further, another array (integer mask), is established in order to step through the d input array. The method consists of a bitwise system to determine the offset into the array. Employing this method conserves memory. Besides the masking system, the routine *func*() contains the actual algebraic expression, in the form of a binary tree, to be solved to obtain the fuzzy output. In *func*(), standard software engineering and symbolic programming rules are applied to reduce the number of multiplications. Such rules have lead to the *consts*() function seen later in the code, which multiplies the result by any constants in the original expression that were not necessary in evaluating the variables *fmin* and *fmax*. The routine *consts*() may save at least 2^N multiplications and/or divisions, where N equals the number of fuzzy parameter inputs. Finally, the *fwa*() scheme calls another routine, *table_vals*(). *table_vals*() constructs a look-up table of the input parameters that form the discrete outputs lying on the actual membership curve at the α-cut levels (used to determine which inputs combined to give a particular value in the output performance curve). Interpolation may be employed to determine the values of the corresponding inputs for other points on the curve.

The algorithmic complexity of the implementation (repeated from Equation 2) may be determined. For N design parameters, the complexity is of order

$$H \sim M \cdot 2^{N-1} \cdot \kappa$$

where H equals the number of operations and κ equals the number of multiplications and divisions in the routine *func*(). From a practical standpoint, a minicomputer or workstation executes this routine (binary operations only in *func*()) essentially in real-time for at least fifteen fuzzy input parameters for one equation. When adding trigonometric functions to the expression in *func*(), a time delay of seconds becomes apparent for around ten inputs.[2] Overall, the complexity result demonstrates the relative efficiency of LIA and its implementation scheme.

2.2.3 Revising the LIA Approach

Reducing the Complexity In the previous sections of this paper, the LIA algorithm for design calculations was presented in detail. Extensions to the algorithm in terms of implementation and reducing the complexity of the functional expression have been shown. Although the extended algorithm, as it stands, is usable for *real-time* engineering design calculations,

[2]These qualitative comments are generalizations of benchmarks taken from a Sun Microsystems 4/330 workstation. Of course, the comments here will vary somewhat depending on the expression in *func*().

```
Inputs:
(1) d, array of elements of input parameters.
(2) N, number of fuzzy input parameters.
(3) M, number of discrete points.

Outputs:
(1) p, array containing imprecise result, p = f(d_1, ..., d_N).

extended-calculation(d,N,M)

BEGIN
  integer ialpha_cuts, i, j,
      l, j_lim, icut, ioffset;
  integer mask[no_of_input parameters];
  real fvar[no_of_input parameters],
      f, fmin, fmax;

  ialpha_cuts = M / 2; /* No. of alpha cuts. */
  /* Create the bit masks to retrieve a value in the d array.*/
  for i = 0 to (N-1)
    mask[i] = 2^i;
  for icut = 0 to (ialpha_cuts-1)
  BEGIN
    for i=0 to N-1
      fvar[i] = d[i*M+icut];
    fmin = func(fvar,N);
    fmax = fmin;
    j_lim = 2^N - 1;
    for j=1 to j_lim
    BEGIN
      for l=0 to (N-1)
      BEGIN
        if((mask[l] AND j) == mask[l])
          ioffset = M - icut - 1;
        else
          ioffset = icut;
        fvar[l] = d[l*M+ioffset];
      END;
      f = func(fvar,N);
      fmin = min(fmin,f);
      fmax = max(fmax,f);
      table_vals(f,fmin,fmax,fvar,N,M,icut);
    END;
    consts(fmin,fmax);
    p[icut] = fmin;
    p[M-icut-1] = fmax;
  END;
  return results in p[], with implied α values from ialpha_cuts;
END;
```

Figure 3: Extended Operations Algorithm: Pseudo-Code.

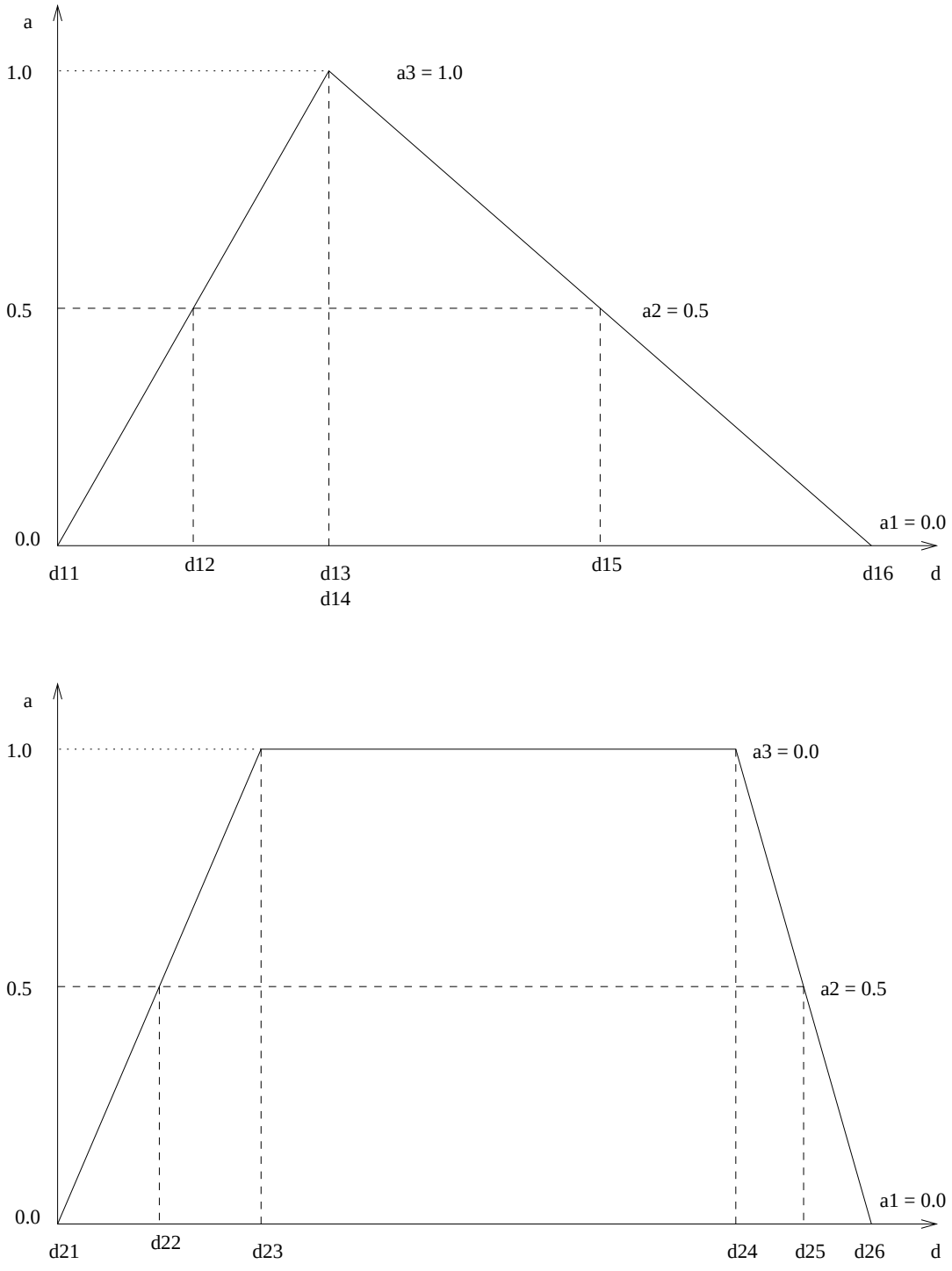

Figure 4: Example of the d Array.

its complexity can be reduced further. Adapting interval analysis methods [14] to the α-cut formulation of the algorithm is the key to the reduction in complexity.

Certain definitions and theorems as presented by Moore [14] are needed to construct and explain the reduction in complexity of the LIA algorithm (as developed in this document). These definitions and theorems are provided in Appendix A, where Moore [14] can be referenced for the details and proofs.

Applying the Methods of Interval Analysis Using engineering design terminology and nomenclature, the problem is to compute the performance parameter results $p_j, j = 1, 2, \ldots, m$ at each α-cut for a class of real-valued design functions $DF(\mathcal{X}_0)$, where $p_j = f(d_1, d_2, \ldots, d_n)$, and $\mathcal{X}_\alpha$ denotes the support of an α-cut across the design parameters. The $DF(\mathcal{X}_0)$ compose the arithmetic functions $\{+, -, /, \cdot\}$ and unary functions $\{\sin(\cdot)$,$\cos(\cdot)$, $\exp(\cdot)$,$\ln(\cdot)$,$\sqrt{\cdot}$, etc.$\}$, *i.e.*, all standard computing functions. Assuming that a function f in the class of functions $DF(\mathcal{X}_0)$ is defined for all $\mathcal{D} \in \mathcal{X}_0$, and assuming that no division by zero or unary operation of zero for $\{\ln(\cdot)$ or $\sqrt{\cdot}\}$ occur, the interval extensions $F(\vec{\mathcal{D}})$ of f will be Lipschitz and inclusion monotonic.

Under these conditions, Theorem A.1 can be applied such that the performance parameter output interval at each α-cut is obtained for arbitrarily sharp bounds (the excess width) for each discretized α-cut. Denote the output interval as $P_j^\alpha(\vec{D}^\alpha)$. When the d_i, $i = 1, 2, \ldots, n$, occur only once in f, only one computation is required (Theorem A.2) to obtain exact bounds for $P_j^\alpha(\vec{D}^\alpha)$ (within the precision of the computation). For multiple occurrences of the d_i, a number of refinements can be applied to determine arbitrarily sharp bounds. Alternatively, as shown by Dong and Wong [7], a combinatorial interval analysis scheme can be used to determine exact bounds for a function f with multiple occurrences of variables.

To reduce the complexity of the LIA algorithm, two changes to Dong and Wong's approach are required:

1. Apply Theorem A.2 and Theorem A.3 such that the permutations in the combinatorial step of the LIA algorithm are applied only to the variables that are repeated.

2. Use interval arithmetic to determine the bounds instead of normal single-valued arithmetic.

These changes affect the LIA algorithm as follows. For N real imprecise design parameters, $\tilde{d}_1, \ldots, \tilde{d}_N$, let d_i ($i \in [1, N]$) be an element of $\tilde{d}_i$. Given a performance parameter represented by the mapping $p = f(d_1, \ldots, d_N)\ \forall d_i \in \tilde{d}_i$ respectively, let $\tilde{P}$ be the fuzzy output of the mapping. The following steps lead to the solution of $\tilde{P}$:

1. For each $\tilde{d}_i$, discretize the preference function into a number of α values, $\alpha_1, \ldots, \alpha_M$, where M is the number of steps in the discretization.

2. Determine the intervals for each parameter $\tilde{d}_i, i = 1, \ldots, N$ for each $\alpha-$cut, $\alpha_j, j = 1, \ldots, M$.

3. Given p unrepeated design parameters in f and $N-p$ repeated DPs, separate the intervals for the $\tilde{d}_i$ as follows: $\vec{\tilde{d}} = \{\tilde{d}_1, \ldots, \tilde{d}_p, \tilde{d}_{p+1}, \ldots, \tilde{d}_N\}$, where the repeated variables are contained in $\vec{\tilde{d}}$ between $\tilde{d}_{p+1}$ and $\tilde{d}_N$.

4. Using one end point from each of the $N-p$ intervals for each α_j, combine the end points into an $(N-p)$-ary array such that 2^{N-p} distinct permutations exist for the array.

5. For each of the 2^{N-p} permutations, determine through interval computations $P_k = f(D_1, \ldots, D_N), k = 1, \ldots, 2^{N-p}$, where P_k denotes the interval at the k^{th} permutation for p_k, similarly for $D_1, \ldots, D_N$. The resultant interval for the $\alpha-$cut, α_j, is then given by
$$P^{\alpha_j} = [\min(\underline{p}_k), \max(\overline{p}_k)].$$

The complexity of this modified algorithm is

$$H \sim M \cdot 2^{N-p-1} \cdot \upsilon$$

where $N-p$ is the number of repeated design parameters in the performance expression and υ equals the number of interval operations in the expression $f(d_i)$. As p approaches N, the new algorithm is much more efficient than the original LIA; however, there does exist a trade-off in overall complexity because the modified version of LIA uses interval operations whereas the original does not. For implementations of the modified algorithm where the interval operations are carried out in assembly code, the effect will not be dramatic. But when the interval operations are implemented in subprogram calls, and when p is much less than N, the values of κ and υ should be calculated to determine whether the modified LIA or the original will perform better.

2.2.4 Extending LIA for Internal Extrema

The LIA algorithm and extensions presented above are valid only for real-valued functions f, and corresponding interval extensions $F(\mathcal{X}_0)$ that do not include internal bounded *extrema* for the intervals in question, $\mathcal{X} \in \mathcal{X}_0$. This is because only the endpoints (at a given α-cut) of the input parameters d_i $i = 1, \ldots, n$ are used in the computation. We now introduce an extension to the LIA algorithm to determine the correct bounds $P^{\alpha_j}(\vec{D}^{\alpha_j})$ for a given α-cut α_j with the following procedure:

1. For each α-cut α_j, determine if an internal extrema exists for the α-cut intervals of d_i $i = 1, 2, \ldots, n$, $p = f(d_1, d_2, \ldots, d_n)$. This may be accomplished by either analytically or numerically solving
$$\frac{\partial p}{\partial d_i} = 0$$
for each d_i.

2. Denoting the extrema by ξ_l, and denoting the values of the d_i that make up a given ξ_l by $\varepsilon_{l,i}$, if every $\varepsilon_{l,i}$ lies within $\mathcal{X}_\alpha$, calculate ξ_l.

3. If every $\varepsilon_{l,i}$ does not lie within the α-cut, let q span those that do not, $\varepsilon_{l,q}$, and m span those that do, $\varepsilon_{l,m}$. Denote the extrema within $\mathcal{X}_\alpha$ as $f(\varepsilon_{l,i}) = \xi_l$. Denote the extrema on the boundary of the $\mathcal{X}_\alpha$ caused by the extrema outside $\mathcal{X}_\alpha$ (but within $\mathcal{X}_0$) as $\xi_l = f(\varepsilon_{d_m,d_n})$, where d_m span the design parameters such that ε_{d_m} are within the α-cut, and d_n span the design parameters which are not within the α-cut, and use the values of d_n as the extrema of the α-cut on d_n.

4. If the above condition is true, compare the calculated extrema ξ_l with P^{α_j} from the LIA algorithm such that
$$P^{\alpha_j} = [\min(\underline{p}^{\alpha_j}, \xi_l), \max(\overline{p}^{\alpha_j}, \xi_l)]$$
for all l.

Alternatively, Skelboe [20] developed an algorithm for computing interval expressions with internal extrema. Skelboe's approach, which implements Theorem A.3 and Theorem A.4, relies on a subdivision of the argument intervals of an expression and a subsequent recomputation of the expression with the new intervals. His approach can compute refinements of an interval extension to arbitrarily sharp bounds. In the best case, the Skelboe algorithm can outperform the LIA algorithm, even with extensions, when internal extrema are present. However, the complexity of the Skelboe algorithm is dependent on the subdivision structure and the placement of the internal extrema with respect to the subdivisions, *e.g.*, a centered form for the interval expressions. Such a dependency causes the upper bound on complexity to greatly exceed that of the *guaranteed* complexity of the extended LIA algorithm.

3 Anomalies in Imprecise Calculations

3.1 Introduction

The extended-operations algorithm (LIA) and extensions presented above provide a basis for computing with sets of *imprecise* parameters in preliminary engineering design. Even though the algorithm and extensions, as discussed in this document, can be applied for standard computing functions, certain limitations apply (as presented above): the preference functions must satisfy the normality and convexity conditions and must be continuous over $\tilde{d}$; no singularities of the functions can occur over $\tilde{d}$ (*i.e.*, no division by zero can occur, and no zero arguments can occur in $f_j(\vec{d})$ for each $\tilde{d}_i$ for the unary operations, such as the natural logarithm and the square root); and only monotonic regions of multi-valued functions, *e.g.*, sine and cosine, are computed for a given $\tilde{d}_i$. This section will consider cases when one or more of these conditions are violated.

3.2 Discontinuous Preference Functions

The extension principle is used to operate and compute on input sets, using the standard computing functions. Because of the set operations, the question arises: do there exist computing functions which can operate on well-formed, continuous preference functions and create a preference function which is discontinuous? To study this question, consider the function $z : \mathbb{R} \rightarrow \mathbb{R} \mid z(x) = 2.927x^3 - 1.927x$, (to be interpreted as a performance parameter p) which is a cubic equation passing through $(-1,-1)$, $(0,0)$, and $(1,1)$ with two local extremum, as shown in Figure 5.

Let $\tilde{X}$ be a triangular imprecise number with preference function $\mu(x|\tilde{X})$, where $\mu(x|\tilde{X}) = 0$ outside $\{x|x \in (-1,1)\}$, equals 1 at $x = 0$, and linearly increases to $\mu(0) = 1$ from the endpoints $\mu(-1) = 0$ and $\mu(1) = 0$ ($\tilde{X}$ is to interpreted as a design parameter d). Substituting $\tilde{X}$ and the equation for z into the LIA algorithm, one obtains the resulting incorrect preference function for $\tilde{Z}$, shown in Figure 6. The problem is the two local extremum in the function z. Hence the algorithm of Section 2.2.4 must be applied.

Substituting $\tilde{X}$ and the equation for z into the extended LIA algorithm for functions with extrema, one obtains the resulting preference function for $\tilde{Z}$, shown in Figure 7. Jump discontinuities in preference occur at $z = \pm 0.602$. These result from the local extremum in the function z, shown in Figure 5 ($x = \pm 0.42$). At z values just above $z = 0.602$, only points in the neighborhood of $x = 0.926$ are in the pre-image of these z; the map is one-to-one. But at z values just below $z = 0.602$, the pre-image contains points in the neighborhood of both $x = -0.42$ and $x = 0.926$. Values of z below the local extremum in Figure 5 ($x = 0.42$,

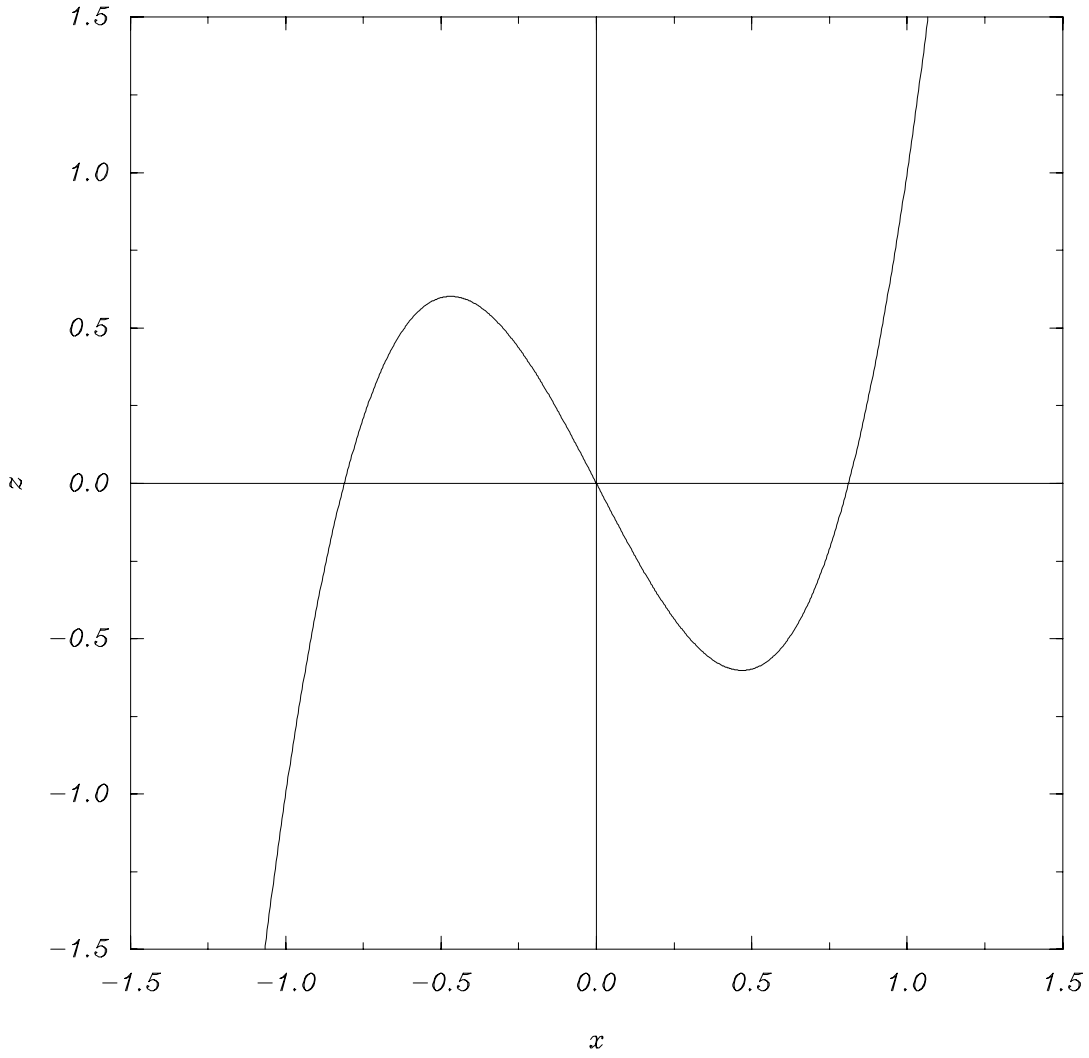

Figure 5: The cubic equation $z = 2.927x^3 - 1.927x$.

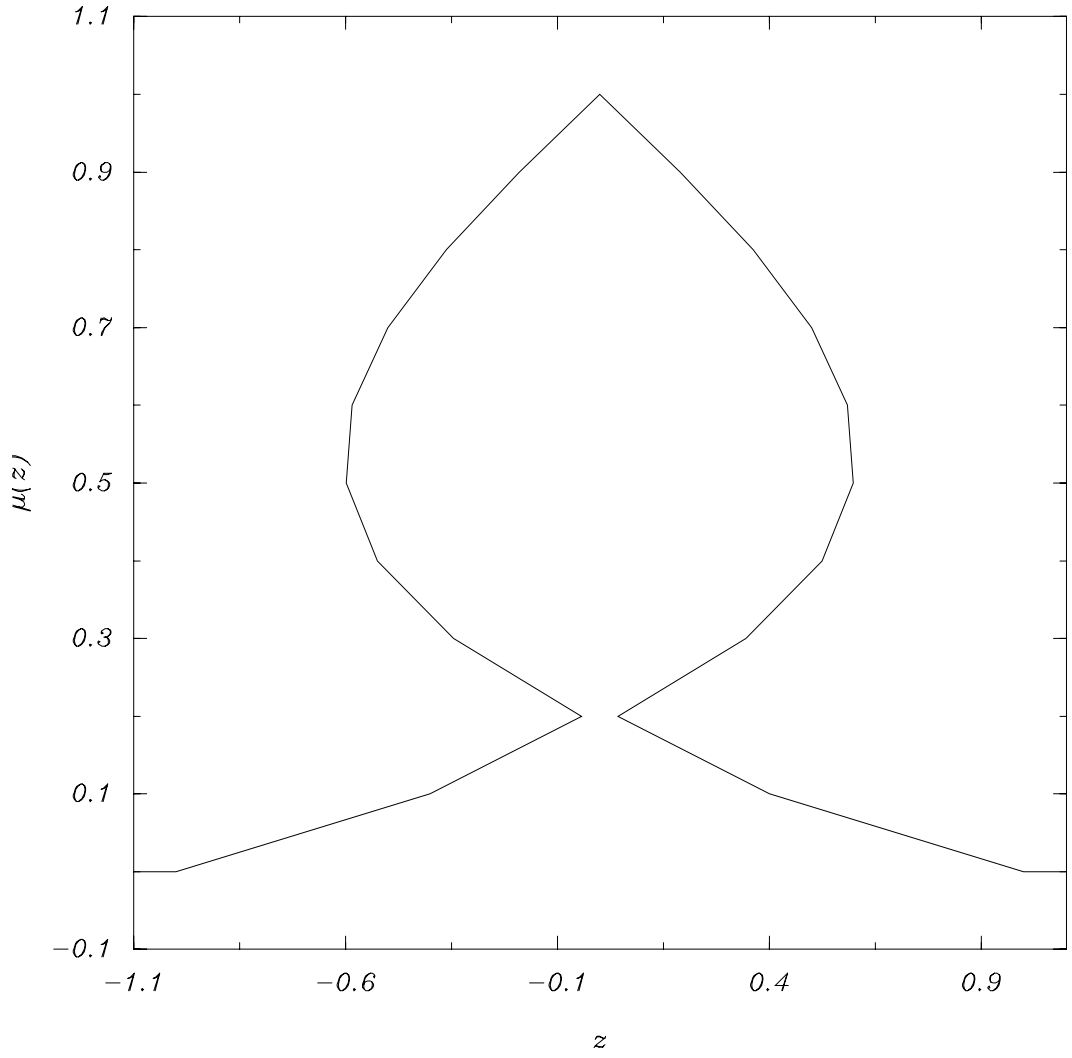

Figure 6: Incorrect LIA solution for $\mu(z|\tilde{Z})$.

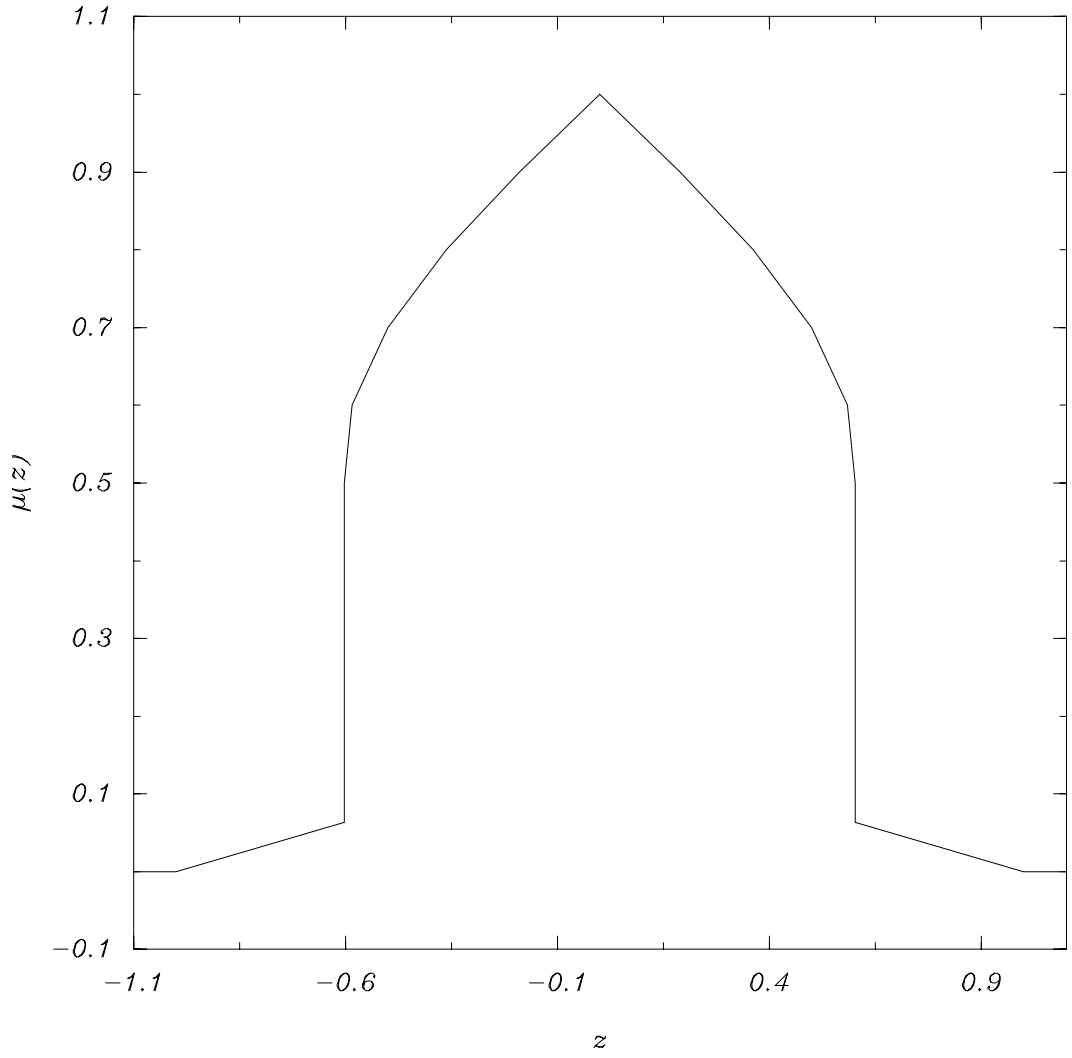

Figure 7: Output preference function $\mu(z|\tilde{Z})$.

$z = 0.602$) have 3 points x in their pre-image. Points in the neighborhood of $x = -0.42$ have higher preference than points in the neighborhood of $x = 0.926$, and hence at $z = 0.602$ there is a jump discontinuity in preference.

This type of discontinuity in output preference is easily understood; it arises from the multiplicity of points in the map's pre-image. The multiplicity here is finite, and can be dealt with. In particular, this is a demonstration of when the algorithm introduced in Section 2.2.4 must be applied. Real concerns arise, however, when the output preference functions exhibit unboundedness, either in support or in the number of discontinuities. Representation and calculation of the imprecise preference functions then becomes difficult, as will be demonstrated below.

3.3 Unbounded Preference Functions

Consider the possibility of functions which operate on well-formed imprecise numbers and create output preference functions whose support become unbounded. If such functions exist, they would fail when applying the methods of Section 2.

Consider an imprecise number $\tilde{X}$ of "about $1/4$, and possibly negative", *i.e.*, let $\tilde{X}$ be a triangular imprecise number with preference function $\mu(x|\tilde{X})$, where $\mu(\frac{1}{4}) = 1$, and $\mu(x|\tilde{X}) = 0$ outside $\{x|x \in (\frac{1}{4} - s, \frac{1}{4} + s)\}$, and s will be gradually increased to observe the variation in calculation results. One expects such an imprecise number will become ill-defined with inversion (when s becomes large enough to encompass zero), since the inverse of points in the neighborhood of zero become unbounded.

The imprecise variable $\tilde{X}$ centered at $1/4$ was systematically increased in support to include negative numbers to observe the effects of inclusion of zero within the support. The effect on the output of the inverse function $z : \mathbb{R} \rightarrow \mathbb{R} \mid z(x) = 1/x$ was observed. Substituting $\tilde{X}$ and the equation for z into the extension principle, one obtains the equation for the preference of any z as

$$\mu(z) = \mu(x), \text{ where } x : x = 1/z \tag{4}$$

since $z = f(x) = 1/x$ is one to one.

The results are shown in Figure 8. As s increases, the preference functions for z begins to extend to include non-zero preference for unbounded values; all calculations are performed with the nominal $\tilde{X}$ remaining at $1/4$. This is a case of a function which operates on a well-formed imprecise number and produces a preference function with unbounded support. This occurs due to $\tilde{X}$ including 0, a singularity of the function z. Hence the output preference includes $z(0) = 1/0 = \infty$. When the input preference function $\mu(x|\tilde{X})$ includes both positive and negative points in the neighborhood of zero, the output preference function $\mu(z)$ has support over both positive and negative values in the "neighborhood" of infinity. Therefore the output preference function will appear as shown in Figure 8.

The LIA will fail, of course, since zero is a singularity of z, and hence cannot occur in $\tilde{X}$. This result seems to indicate that using preference curves to represent designer uncertainty will sometimes fail. But consider what the result indicates when the preference function is interpreted as the degree of designer preference. The unbounded results on z are due to the preferences stated on x. The problem has been incorrectly formulated, and must be reformulated to hedge the preferences on x away from the singular point zero, just as the designer would have to do if imprecision were not used. This method warns the designer of the problem, whereas using crisp or single valued calculations (in the example, just using $x = 1/4$) would not.

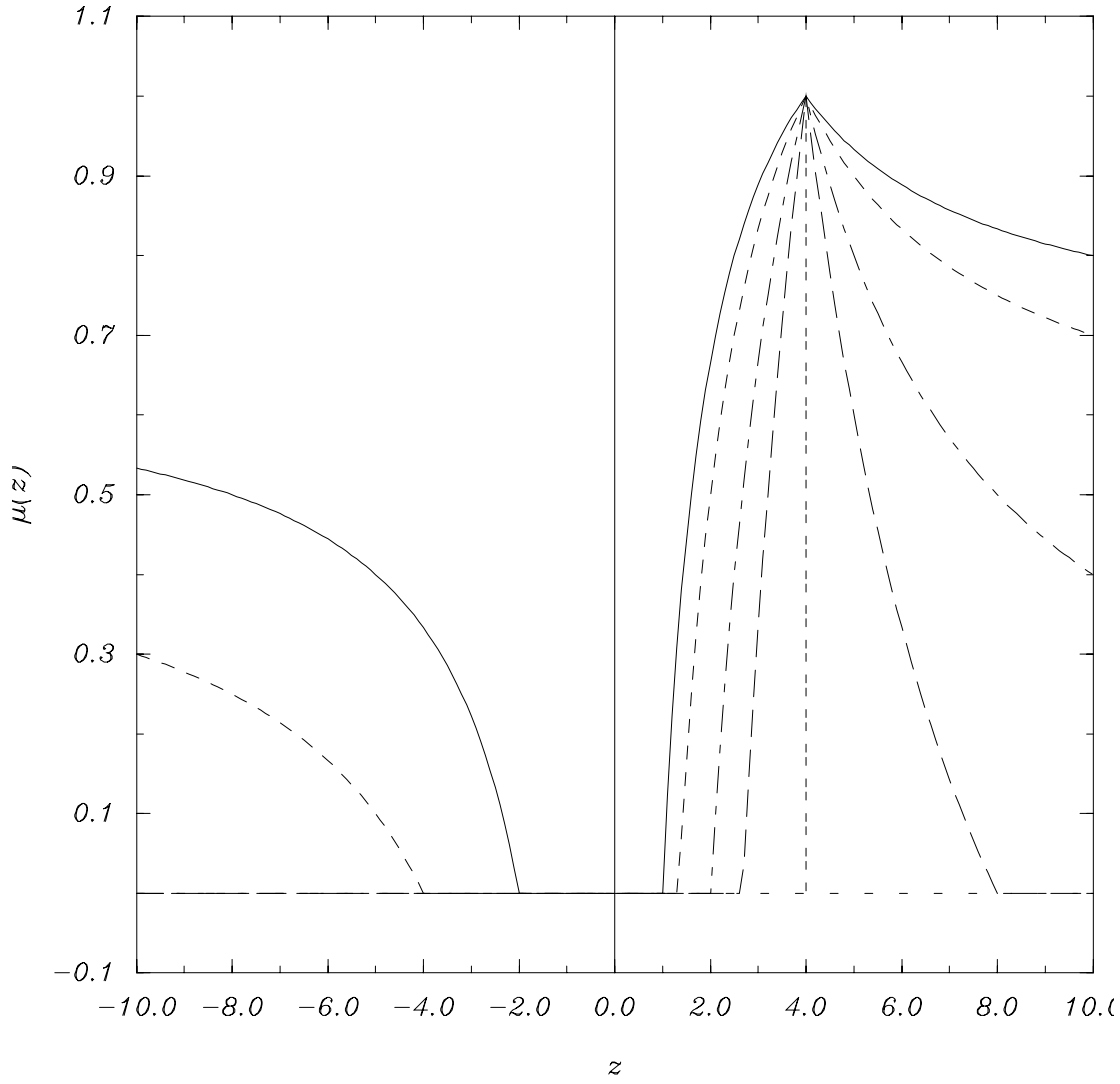

Figure 8: Output preference functions $\mu(z)$.

3.4 Singular Points and Preference Functions

Consider any function which operates on well-formed imprecise numbers and creates an output preference function requiring a limit process for evaluation. Such could be the case of a function with a singularity within the support of the design parameter's preference. To demonstrate this, consider

$$z : \mathbb{R} \to [-1, 1] \subset \mathbb{R} \mid z(x) = \begin{cases} \sin(1/x) & x \neq 0 \\ 0 & x = 0 \end{cases}$$

This function has a non-removable singularity at $x = 0$ (multiplying by $(x-0)^n$ will not create a locally regular function for any n). As $x \to 0$, $z(x)$ oscillates an unbounded number of times between -1 and 1. Refer to Figure 9. The question then arises as to the output preference behavior when the input imprecise number $\tilde{X}$ includes in its support the singular point $x = 0$.

Let $\tilde{X}$ be an imprecise number with preference function $\mu(x|\tilde{X})$, where $\mu(x|\tilde{X}) = 0$ outside $\{x|x \in (-\pi + s, \pi)\}$, $\mu(\frac{s}{2}) = 1$, and s is gradually decreased from $\frac{3}{2}\pi$ to 0 to observe the variation in calculation results. Substituting $\tilde{X}$ and the equation for z into the extension principle, one obtains the equation for the preference of any z as

$$\mu(z) = \sup\{\mu(x) \mid x : z = \sin(1/x)\} \tag{5}$$

Figure 10 shows the corresponding output preference functions resulting from application of Equation 5 to the imprecise numbers $\tilde{X}$.

When the support for the input $\tilde{X}$ does not include the singular point 0 (*i.e.*, $s > \pi$), the output preference function remains perfectly well defined. When the input preference function does include the singular point for any support range, one would expect that the output preference function would become not well defined, since, after all, the function z itself becomes oscillatory (unstable). But note that the output preference function remains well defined. This is due to the supremum in the extension principle definition, and the selected input preference function $\mu(x|\tilde{X})$. There is always a point x in each z value's pre-image such that the preference of that x is greater than the singular point preference. Since this supremum preference is greater than the singular point preference, the preference for all z are well defined. Hence the preference $\mu(z)$ is well defined when $x = 0$ does not have the peak preference among all x. When $x = 0$ has the peak preference, the *sup* definition must be explicitly used, since the point with maximum preference in the pre-image of any z does not exist, only a least upper bound exists. All values of z have in their pre-image points x arbitrarily close to zero. In this case, the peak preference of $x = 0$ would be the least upper bound of preference for all z, *i.e.*, $\mu(z) = 1$ $\forall z$.

Calculation of the preference for $z = \sin(1/x)$ poses difficulty. There are numerous and possibly an infinite number of internal extrema within the support of the design parameter (depending if $x = 0$ is in the support). Hence the methods of Section 2.2.4 are impractical as posed. They must be extended using a limiting process. That is, the support of $\mu(x)$ must be split into monotonic intervals of z between the zeros of the slope of z as discussed and solved in Section 2.2.4. The difference here is that there are an unbounded number of such intervals. However, these intervals will become smaller in a limit as will their contribution to $\mu(z)$. The limiting process can be terminated with a convergence criteria. Alternatively, Equation 5 can be directly solved using a limit process as well.

Given that functions exist which exhibit such computational difficulty, one must have an ability to characterize them to know when the methods of Section 2 will fail. This problem's key feature which causes the real time methods to fail is the presence of the singularity of the

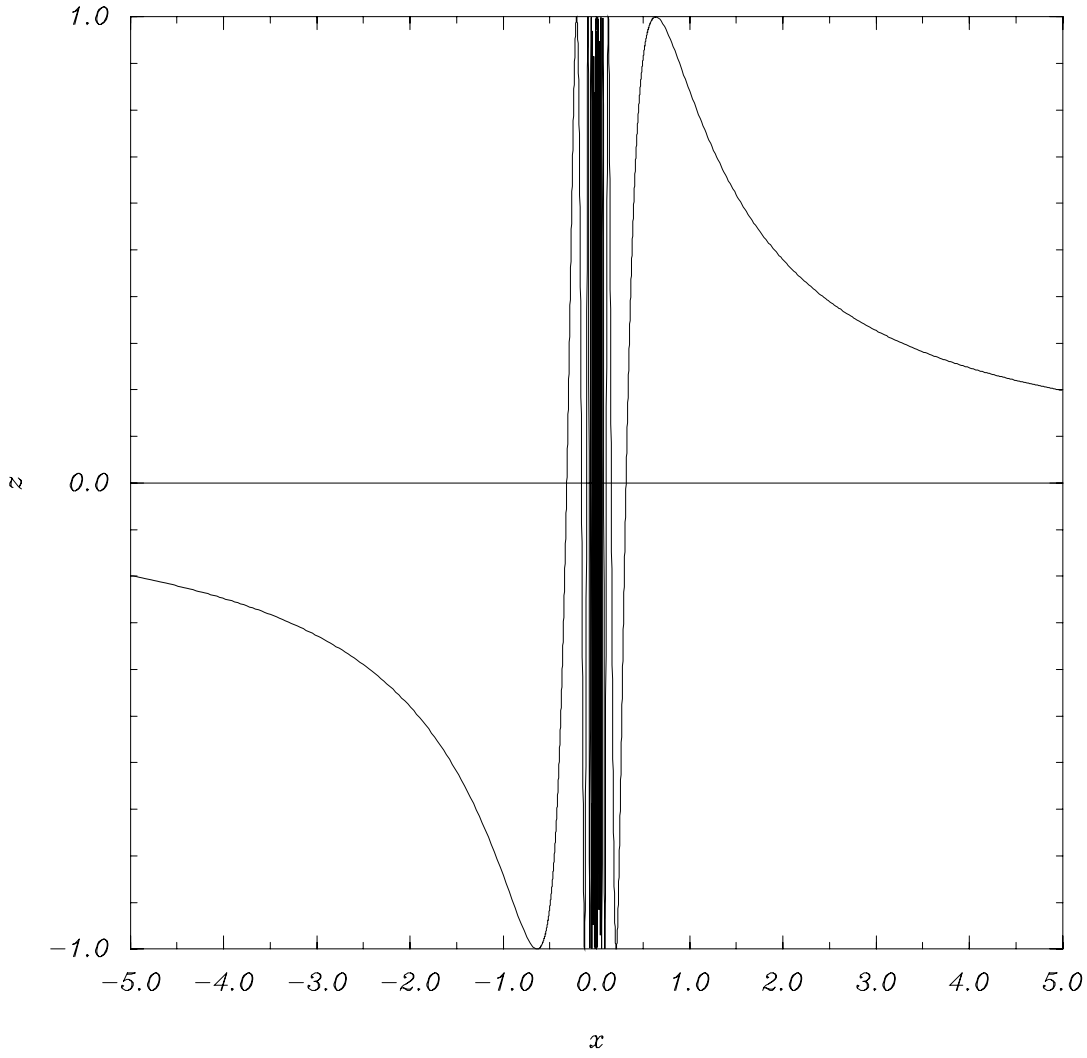

Figure 9: The function $z = sin(1/x)$.

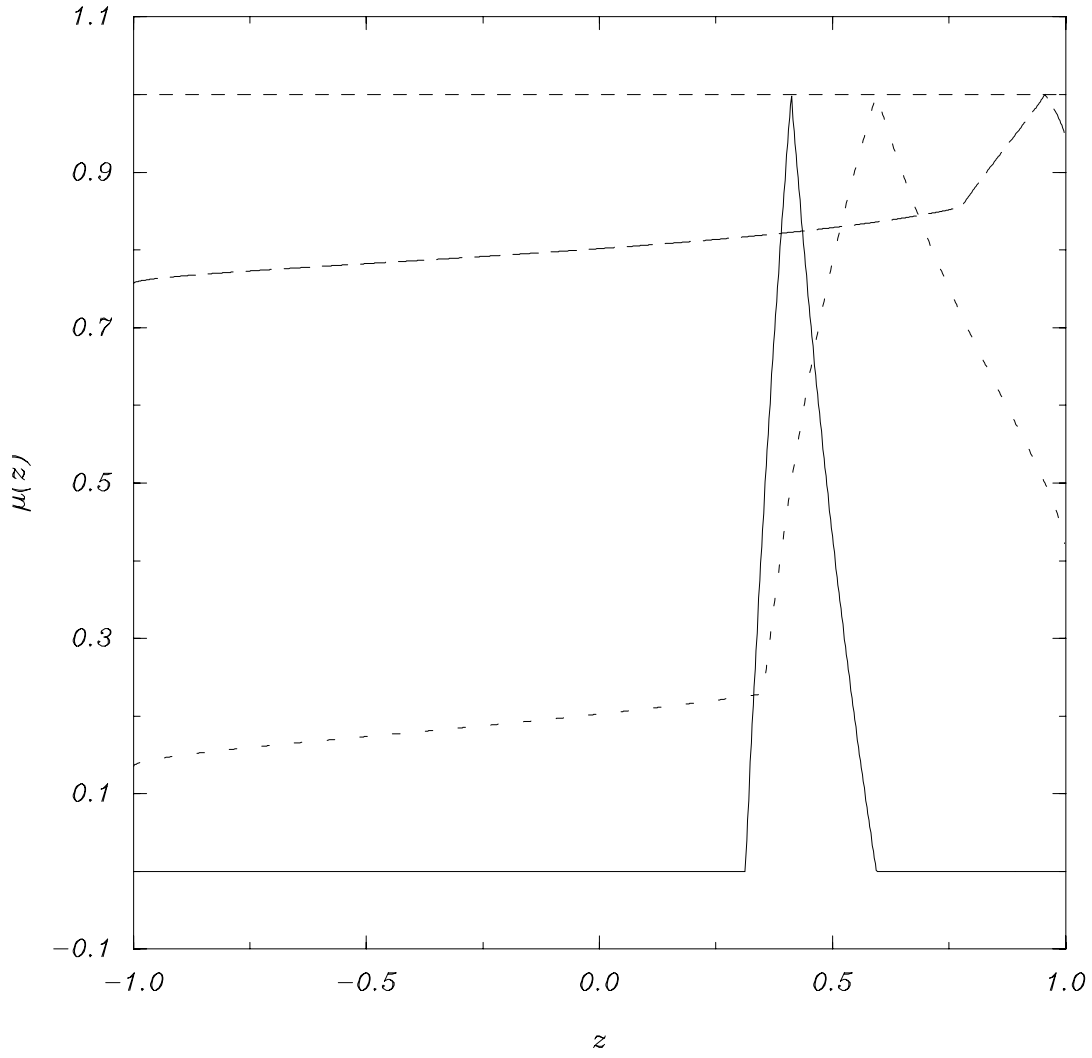

Figure 10: Output preference functions $\mu(z)$.

function z in the design parameter $\tilde{X}$'s support. The same can be said of the previous example involving unbounded support ($z = 1/x$). In both cases, the output preference functions are defined and exist, but they exhibit difficulty in representation or evaluation. These effects are a direct result of the singularity of the function being within the support of the design parameter.

Note in the case of $z = \sin(1/x)$, the singularity requires particular care, in that $\mu(z)$ is defined, even though there is a singularity of z within $\tilde{X}$. Any conclusions drawn by a designer when observing such a resulting preference curve for z may be misleading. The point is that failure to meet the criterion of using the real-time techniques of Section 2 indicate that the functions being considered (z) are incompatible with the specified design parameter preference functions ($\mu(x|\tilde{X})$), and hence is a warning to the designer that the values desired for $\tilde{X}$ should be re-considered.

4 Interpretability of Imprecision Results

Consider the possible existence of continuous functions which operate on well-formed design parameter preference functions and with them create output performance parameter preference functions which oscillate in preference from 0 to 1 as the output z approaches a limit value $\overline{z}$. The existence of any such function would question the viability of the entire proposed method, since such a function would be an interpretable mapping from a design parameter set, where the preferences are known, to an output set where the preferences are un-interpretable. Via construction, it will be shown (though not formally proven) that this is only possible for functions which are infinitely multi-valued (have an infinite number of branches), but need not be singular. Any such functions which map specified preferences into un-interpretable preferences are not due to the fuzzy extension principle, but are entirely due to multi-valued character of the function, and hence would exist whether the fuzzy mathematics were used or not.

Such a function z must convert a well-formed, convex input preference function $\mu(x|\tilde{X})$ into a preference function of the form $\mu(z) = 1/2\sin(1/z) + 1/2$, or more simply behave like $\sin(1/z)$, *i.e.*, behave un-interpretable in the neighborhood of $z = 0$. Denoting $z = f(x)$ and using the extension principle (Equation 1), this produces $f(x) = 1/\arcsin(\mu(x|\tilde{X}))$. Now define $\mu(x)$ as a convex function over all real values with range $[0, 1]$, such as e^{-x^2}. These definitions construct $z : \mathbb{R} \rightarrow \mathbb{R} \mid z = 1/\arcsin(e^{-x^2})$, based on $\mu(z)$ and $\mu(x)$. Using different functions which exhibit the same behavior as the chosen $\mu(z)$ and $\mu(x)$ will construct a similar z, though it may not be as easily expressible. The created z is defined over all x and contains no singularities. It is, however, multi-valued. Refer to Figure 11. Take z to be all of the branches. Then the pre-image of each value z has either zero or two points x.

To observe the effects of z on an imprecise number $\tilde{X}$, consider $\tilde{X}$ where $\mu(x|\tilde{X}) = 0$ outside $\{x|x \in (-1, 1)\}$, $\mu(0) = 1$, and $\mu(x|\tilde{X})$ smoothly increases to the peak at 0 between -1 and 1. Substituting $\tilde{X}$ and the equation for z into the extension principle, the resulting $\mu(z)$ is shown in Figure 12.

Note that for z values in the neighborhood of zero, the preference function becomes arbitrary, from 0 through 1 in preference, as was desired to exhibit. That is, moving a δ amount in the neighborhood of $z = 0$ will result in large changes in $\mu(z)$, from zero to one. Also note the intervals on z where the preference function is not mapped from any x (for example, all z values between $-\infty$ and $\sin(1/\pi)$). These z values require complex x; there are no real x in the pre-image of these z. The number of such intervals in a neighborhood of z becomes unbounded as z approaches zero (and the length of the each interval approaches zero length).

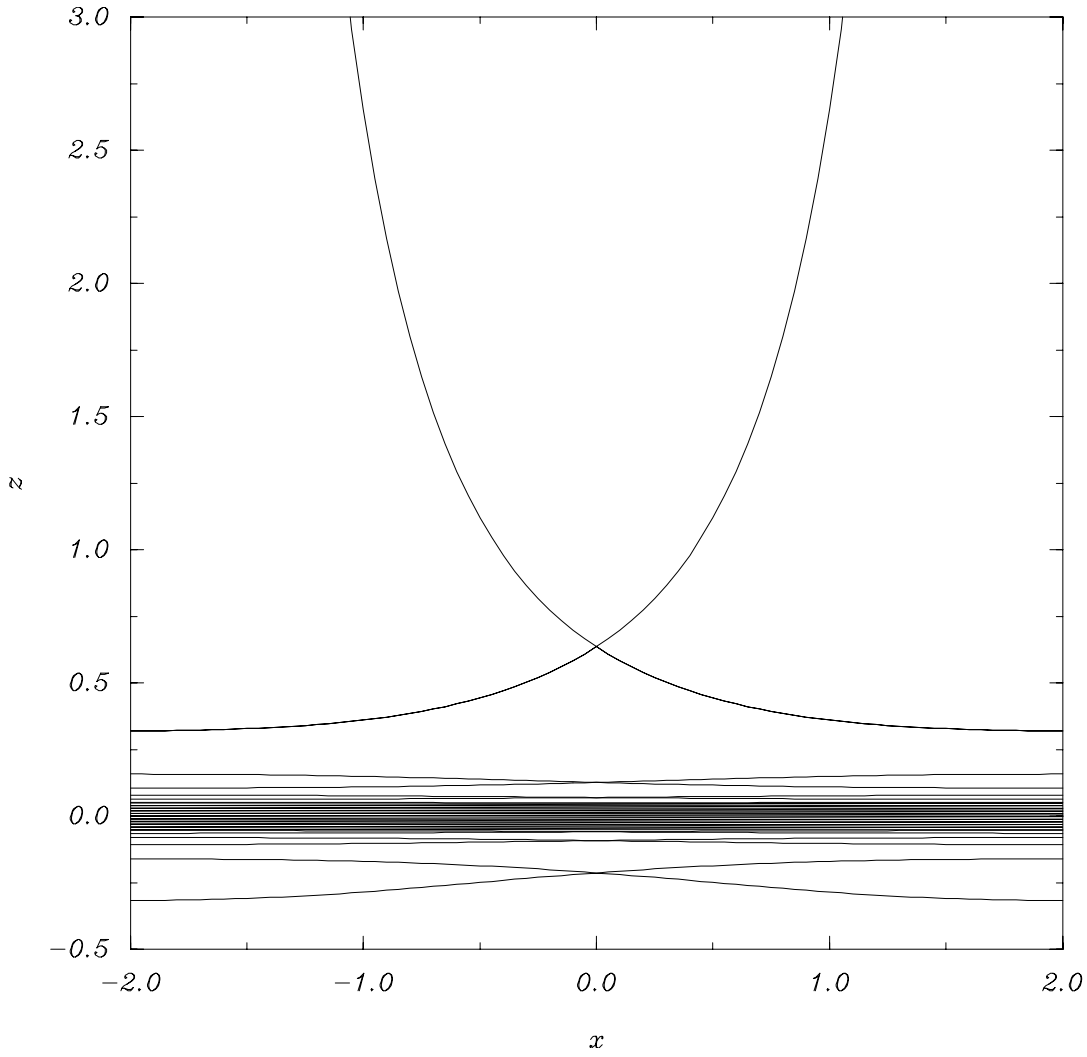

Figure 11: $z = \frac{1}{\arcsin\left(e^{-x^2}\right)}$.

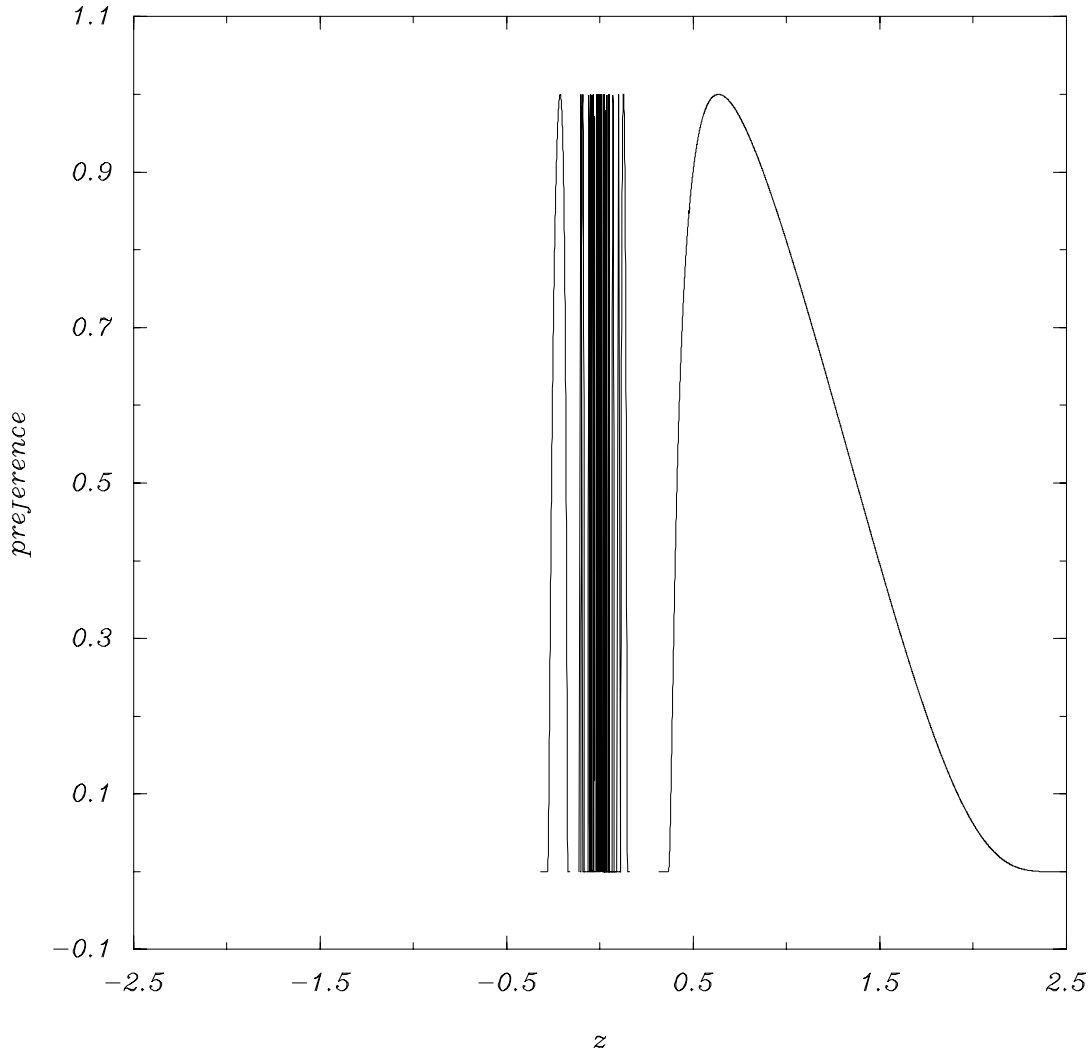

Figure 12: Output preference function $\mu(z)$.

These intervals arise since f is not a surjection; *i.e.*, $Z = f(\mathbb{R}) \not\simeq \mathbb{R}$, but $Z \subset \mathbb{R}$. According to our interpretation of preference, $\mu(z) \simeq 0$ for the values of z having no real points x in the pre-image. We therefore define $\mu(z) = 0$ for such points to define $\mu(z)\ \forall z$, as reflected in Equation 1.

These results would suggest that a preference function could become un-interpretable, since this preference function became un-interpretable as $z \to 0$. But this was entirely due to the unbounded number of branches considered in z: z is not a function. For any particular branch, however, the mapping is well defined. This result (infinitely many choices as $z \to 0$) exists using crisp numbers as well as imprecise numbers.

Note z was constructed based on the desired output function $\mu(z)$ (being un-interpretable at a value). The only way to exhibit this behavior was with a multi-valued function, where the multi-valued character arose as a direct consequence of the desired un-interpretability. Thus the conclusion: only multi-valued functions can lead to un-interpretability, entirely due to the indecision on which branch one is using in the function z itself, not in the fuzzy mathematics. Once a branch is selected, the problem reduces to a simple case which the methods of Section 2 can accommodate. Replacing the specific *sine* used for $\mu(z)$ by any general periodic function and replacing the specific $\mu(x)$ by a general convex imprecise number in the construction would demonstrate the conclusion for the general case (though this is not a formal proof). Overall, if the performance expression is interpretable, then the use of the fuzzy mathematics in design calculations will also produce interpretable results.

5 Related Work

Other authors have considered general methods of applying the extension principle to fuzzy numbers. Kaufmann and Gupta [12] propose a method for analytical arithmetic with fuzzy numbers. They introduce the properties of fuzzy numbers, along with an α-cut formulation of the operations on number sets.

In [9] and again in [10], Dubois and Prade review the correctedness of using α-cut calculations. This has been shown in earlier work by Negoiţă and Ralescu [16], and as well by Nguyen [17] for 2 variable problems, and in [18] by Novák. It is presented clearly in a recent work by Buckley and Qu [4].

Dong and Wong [7], as described above, present a computational algorithm for implementing the extension principle. The application domain for this algorithm emphasizes purely algebraic expressions, with a particular focus on fuzzy weighted averages. In [7], Dong and Wong also review the non-linear programming technique developed by Baas and Kwakernaak [2], an analytical procedure for L-R membership functions due to Dubois and Prade [8], an investigation of fuzzy number algebraic properties by Mizumoto and Tanaka [13], and an investigation of fuzzy algebra in terms of interval operations by Nakamura [15]. These techniques and investigations will not be discussed here.

Dong and Shah [6] also present a computational algorithm for implementing the extension principle. The application domain for this algorithm emphasizes purely algebraic expressions. They also introduce an algorithm for computing the extension principle with functions with bounded extrema, similar to the algorithm of Section 2.2.4. The algorithm as presented in [6], however, will compute incorrect bounds in dimensions higher than 1. The problem is that one must consider combinations of the components of the vertex points with the components of the singularity points, as is done in the algorithm of Section 2.2.4.

In a recent work, Buckley and Qu [4] address the use of α-cuts to evaluate fuzzy equations,

as stated above. They rigorously demonstrate that (1) the multiple occurrence of fuzzy variables in a functional expression and (2) fuzzy variables raised to an integer power may result in a larger interval than the exact solution for each α-cut. These anomalies will occur *only* when using the strict interval operations defined by Moore [14]. However, as shown in the previous sections, a combinatorial technique can be used to evaluate a function where either case (1) or (2) occur. The result, for each α-cut, will be an interval with arbitrary sharp bounds in the support of the fuzzy parameter. This approach, as explained in this paper, treats case (2) as a multiple occurrence of a fuzzy variable.

6 Conclusions

A method to perform engineering design calculations under imprecision is described in this paper. Interval and fuzzy calculi are used to realize an efficient algorithm for such calculations, a necessity in real-time preliminary design activities. The LIA algorithm, proposed by Wong and Dong [7], forms the basis for the computational approach. This algorithm has been extensively modified in terms of the implementation scheme, the capability to handle internal extrema, and the reduction of combinatorial complexity through interval analysis techniques.

A number of simple examples are also included to illustrate possible anomalies with imprecise design calculations. These simple examples demonstrate the ill-conditioning that can occur when operating on a fuzzy variable with simple functional equations. For all of the cases considered, an ill-conditioned state occurs when either there exists a singularity within the support of the input membership function, or the function itself is infinitely multi-valued. While these anomalies represent limiting conditions on the calculation method, design problems will normally not exhibit either a singularity or infinitely multi-valued character in physical space. In the first case, a real-world design problem will be ill-defined or unstable if a singularity occurs in a performance parameter calculation. For example, when the slenderness ratio l/r in Equation 3 approaches zero, the allowable stress in the column approaches infinity. A column with zero slenderness ratio is obviously ill-defined for supporting a structural load. In the second case, a multi-valued function will be constrained by choosing the appropriate branch in physical space. For example, when the designer uses an equation involving the square root function, the designer implicitly means the positive branch. Thus, the method described above for calculating under imprecision will be well-behaved and interpretable, provided that the limiting cases are avoided through careful problem formulation. The "care" needed is, of course, no more than is needed when a designer employs crisp (single-valued) calculation methods.

Acknowledgments

This material is based upon work supported, in part, by: The National Science Foundation under a Presidential Young Investigator Award, Grant No. DMC-8552695; The Caltech Program in Advanced Technologies, sponsored by Aerojet General, General Motors, and TRW; and an IBM Faculty Development Award. Mr. Otto is currently an AT&T-Bell Laboratories Ph.D. scholar, sponsored by the AT&T foundation. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] M. Asimow. *Introduction to Design*. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1962.

[2] S. Baas and H. Kwakernaak. Rating and ranking of multiple-aspect alternatives using fuzzy sets. *Automatica*, 13:47–48, 1977.

[3] J. M. Becker. *A Structural Design Process Philosophy and Methodology*. PhD thesis, University of California, Berkeley, 1973.

[4] J. Buckley and Y. Qu. On using α-cuts to evaluate fuzzy equations. *Fuzzy Sets and Systems*, 38:309–312, 1990.

[5] R. T. Cox. *The Algebra of Probable Inference*. Johns-Hopkins University Press, Baltimore, MD, 1961.

[6] W. Dong and H. Shah. Vertex method for computing functions on fuzzy variables. *Fuzzy Sets and Systems*, 24(1):65–78, 1987.

[7] W. M. Dong and F. S. Wong. Fuzzy weighted averages and implementation of the extension principle. *Fuzzy Sets and Systems*, 21(2):183–199, February 1987.

[8] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, 1980.

[9] D. Dubois and H. Prade. Fuzzy numbers: An overview. In *Analysis of Fuzzy Information*, volume 1, pages 3–39, Boca Raton, FL, 1987. CRC Press.

[10] D. Dubois and H. Prade. *Possibility Theory: An Approach to the Computerized Processing of Information*. Plenum Press, New York, 1988.

[11] H. Jeffreys. *Theory of Probability*. Clarendon Press, third edition, 1961.

[12] A. Kaufmann and M. M. Gupta. *Introduction to Fuzzy Arithmetic: Theory and Applications*. Electrical/Computer Science and Engineering Series. Van Nostrand Reinhold Company, New York, 1985.

[13] M. Mizumoto and K. Tanaka. Some properties of fuzzy numbers. In M.M. Gupta et. al., editor, *Adavances in Fuzzy Sets: Theory and Applications*, Amsterdam, 1984. North Holland.

[14] R. E. Moore. *Methods and Applications of Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[15] Y. Nakamura. Extension of algebraic calculus on fuzzy numbers using alpha-level sets. In *Fuzzy Information Processing Symposium (FIP-84)*, Kauai, Hawaii, 1984.

[16] C. V. Negoiţă and D. A. Ralescu. *Applications of Fuzzy Sets to Systems Analysis*. Halsted Press, New York, 1975.

[17] H. T. Nguyen. A note on the extension principle for fuzzy sets. *Journal of Math. Anal. Appl.*, 64:369–380, 1978.

[18] Vilém Novák. *Fuzzy Sets and Their Applications*. Adam Hilger, Philadelphia, 1989. Published in English by IOP Publishing Ltd.

[19] G. Pahl and W. Beitz. *Engineering Design*. The Design Council, Springer-Verlag, New York, 1984.

[20] S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.

[21] K. T. Ulrich and W. P. Seering. Synthesis of schematic descriptions in mechanical engineering. *Research in Engineering Design*, 1(1):3–18, 1989.

[22] I. Wilson and M. Wilson. *From Idea to Working Model*. J. Wiley and Sons, New York, 1970.

[23] K. L. Wood. *A Method for Representing and Manipulating Uncertainties in Preliminary Engineering Design*. PhD thesis, California Institute of Technology, Pasadena, CA, 1989.

[24] K. L. Wood and E. K. Antonsson. Computations with Imprecise Parameters in Engineering Design: Background and Theory. *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, 111(4):616–625, December 1989.

[25] K. L. Wood, E. K. Antonsson, and J. L. Beck. Representing Imprecision in Engineering Design – Comparing Fuzzy and Probability Calculus. *Research in Engineering Design*, 1(3/4):187–203, 1990.

[26] K. L. Wood, K. N. Otto, and E. K. Antonsson. A Formal Method for Representing Uncertainties in Engineering Design. In P. Fitzhorn, editor, *First International Workshop on Formal Methods in Engineering Design*, pages 202–246, Fort Collins, Colorado, January 1990. Colorado State University.

[27] L. A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, 1975.

A Interval Analysis Definitions

Definition A.1 *Let Λ and Υ be arbitrary sets, and let $g : \Lambda \to \Upsilon$ be an arbitrary mapping (function) from Λ into Υ. Denoting $S(\Lambda)$ and $S(\Upsilon)$ as the families of subsets of Λ and Υ respectively, the* **set-valued mapping**, *$\overline{g} : S(\Lambda) \to S(\Upsilon)$,*

$$\overline{g}(\mathcal{X}) = \{g(x) : x \in \mathcal{X}, \mathcal{X} \in S(\Lambda)\}$$

is the **united extension** *of g. This can also be written as*

$$\overline{g}(\mathcal{X}) = \bigcup_{x \in \mathcal{X}} \{g(x)\}.$$

Definition A.2 *A* **rational interval function** *is a function whose interval values are defined by a specified finite sequence of interval arithmetic operations.*

Definition A.3 *Using Definition A.1, and the assumptions therein, the united extension $\overline{g} : S(\Lambda) \to S(\Upsilon)$, has the* **subset property***:*

$$\mathcal{X}, \mathcal{Y} \in S(\Lambda) \;\; \textit{with} \;\; \mathcal{X} \subseteq \mathcal{Y} \implies \overline{g}(\mathcal{X}) \subseteq \overline{g}(\mathcal{Y}).$$

Definition A.4 *Let f be a real valued function of n real variables $x_1, x_2, \ldots, x_n$. An* **interval extension** *of f is an interval valued function F of n interval variables $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ with the property*

$$F(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n), \quad \textit{for real arguments}.$$

Definition A.5 *An interval valued function F of the interval variables $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ is* **inclusion monotonic** *if*

$$\mathcal{Y}_i \subseteq \mathcal{X}_i, \quad i = 1, 2, \ldots, n,$$

implies

$$F(\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_n) \subseteq F(\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n).$$

United extensions, which all have the subset property, are inclusion monotonic. Interval arithmetic is inclusion monotonic, as are rational interval functions and the natural interval extensions of all the standard functions used in computing.

Definition A.6 *An interval extension F is* **Lipschitz** *in $\mathcal{X}_0$ if there is a constant L such that $w(F(\mathcal{X})) \leq L \cdot w(\mathcal{X}) \;\; \forall \, \mathcal{X} \subseteq \mathcal{X}_0$.*

Theorem A.1 *If $F(\mathcal{X})$ is an inclusion monotonic, Lipschitz, interval extension for $\mathcal{X} \subseteq \mathcal{X}_0$, then the excess width of a refinement, $F_{(N)}(\mathcal{X})$, the union of interval values of F on the elements of a uniform subdivision of $\mathcal{X}$, is of order $1/N$. This gives*

$$F_{(N)}(\mathcal{X}) = \bigcup_{j_i=1}^{N} F(\mathcal{X}_{1,j_1}, \ldots, \mathcal{X}_{n,j_n}) = \overline{f}(\mathcal{X}_1, \ldots, \mathcal{X}_n) + \mathcal{E}_N$$

and there is a constant K such that

$$w(\mathcal{E}_N) \leq K \cdot w(\mathcal{X})/N,$$

where $\mathcal{E}_N$ is the error compared with the exact solution. If $F(\mathcal{X})$ is in centered form, the corresponding width of the error interval of the N^{th} refinement is

$$w(\mathcal{E}_N) \leq K_c \cdot w(\mathcal{X})/N^2,$$

for some K_c.

Theorem A.2 *If there does not exist a multiple occurrence of the real variables $x_1, x_2, \ldots, x_n$ in a given real valued function $f(x_1, x_2, \ldots, x_n)$, then the interval extension $F(\mathcal{X})$ corresponding to $f(x)$ is the united extension of f for all $\mathcal{X} \subseteq \mathcal{X}_0$.*

Theorem A.3 *Let $F(\mathcal{X}_1, \ldots, \mathcal{X}_j, \ldots, \mathcal{X}_n)$ be a rational interval function. If $\mathcal{X}_j$ occurs only once in F and*

$$\mathcal{X}_j = \bigcup_{i=1}^{N} \mathcal{X}_j^{(i)},$$

then

$$F(\mathcal{X}_1, \ldots, \mathcal{X}_j, \ldots, \mathcal{X}_n) = \bigcup_{i=1}^{N} F(\mathcal{X}_1, \ldots, \mathcal{X}_j^{(i)}, \ldots, \mathcal{X}_n)$$

for all $\mathcal{X} \in \mathcal{X}_0$.

Theorem A.4 *Let $F(\mathcal{X})$ be a rational interval function written in centered form. Each of the interval variables $\mathcal{X}_{p+1}, \ldots, \mathcal{X}_n$ occurs just once in F. Subdividing each of the interval variables $\mathcal{X}_1, \ldots, \mathcal{X}_p$ so that*

$$\mathcal{X}_i = \bigcup_{j=1}^{N} \mathcal{X}_i^{(j)} \text{ with } w(\mathcal{X}_i^{(j)}) = \frac{1}{N} w(\mathcal{X}_i),$$

there is a positive number K such that

$$\bigcup_{i_1=1}^{N} \cdots \bigcup_{i_p=1}^{N} F(\mathcal{X}_1^{(i_1)}, \ldots, \mathcal{X}_p^{(i_p)}, \mathcal{X}_{p+1}, \ldots, \mathcal{X}_n) = \overline{f}(\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n) + \mathcal{E}_N$$

where

$$w(\mathcal{E}_N) \leq \frac{K}{N^2} w(\mathcal{X}).$$