ENGINEERING QUALITATIVE ANALYSIS AND ITS APPLICATION ON FATIGUE DESIGN OF STEEL STRUCTURES

By

Ye Zhou

B.A.Sc, The University of British Columbia, 1995
M.A.Sc, The University of British Columbia, 1998

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
Department of Civil Engineering

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
July, 2003
© Ye Zhou, 2003
In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.
Fatigue failures are often encountered in steel structures under heavy cyclic loadings. The advances in Linear Elastic Fracture Mechanics (LEFM) and Finite Element Analysis (FEA) provide engineers and researchers reasonably effective tools for solving fatigue problems. However, most structural and building design codes have not yet taken advantage of advanced theories in modern fracture mechanics due to the complexity of analysis as well as the large quantity of variables involved. The result is that practicing structural engineers are not able to fully utilize state-of-the-art research in fatigue analysis. This thesis considers fatigue problems in structural engineering using outcomes of recent advancements in numerical qualitative analysis. A major result of this research work is an integrated set of software modules for fatigue analysis and evaluation, which have the flexibility to be applied by practicing structural engineers in a variety of situations. The software combine techniques such as interval arithmetic and qualitative reasoning used in the fields of mathematics and information science, and apply them to fatigue analysis.

Conventional computation methods generally limit practicing engineers from using complex formulations or considering uncertainties in general. A method is needed that can be implemented regardless of the uncertainty or linearity of the design parameters and their constraints. More exotic methods such as qualitative reasoning provide an effective and sound technique for solving complex and uncertain scenarios. Uncertainties in fatigue designs can be formulated as variables in the application domain and processed by numerical constraint reasoning. The capability of representing design parameters and outcomes in a two-dimensional solution space provides a practical way for engineers to leverage their existing knowledge and experience. The software expresses the results of the analysis in variable ranges and diagrams showing a two-dimensional design space. As a result, the use of qualitative reasoning to define solution trends and ranges can assist in the difficult process
of making appropriate engineering assumptions and judgments when carrying out complicated
analysis procedures. In addition, interval constraint analysis can be used to derive controlling
parameters and design space, therefore giving engineers a good overall understanding of a problem
when practical experience is not available.
## Table of Contents

1. **Abstract** ................................................................. II

2. **Table of Contents** .................................................. IV

3. **Figures** ...................................................................... VII

4. **Acknowledgments** .................................................... IX

1. **Engineering Analysis with Uncertainties and Complexities** .......... 1
   1.1 Engineering Approaches to Design Problems .......................... 1
   1.2 Advances in Computational Methods .................................. 3
   1.3 Qualitative Reasoning with Engineering Uncertainties ............. 4
      1.3.1 Interval Constraints and Solution Space ...................... 5
   1.4 Prospect for Further Research ........................................ 12
      1.4.1 Fatigue Design of Steel Structures Using Qualitative Reasoning ....... 12
   1.5 Contribution of Current Research .................................... 13
      1.5.1 Qualitative Reasoning Software ................................ 14

2. **Theories and Algorithms** ............................................. 17
   2.1 Conventional Techniques for Modeling Uncertainties in Engineering .. 17
      2.1.1 Reliability/Probabilistic Methods .............................. 18
      2.1.2 Algebraic Algorithms for Nonlinear Constraints ............ 19
      2.1.3 Iterative Numerical Algorithms ................................ 20
      2.1.4 Stochastic Methods ............................................ 21
   2.2 Interval Arithmetic .................................................... 22
   2.3 Numerical Consistency Techniques ................................... 24
      2.3.1 General Consistency Techniques ............................... 26
      2.3.2 Preprocessing Methods .......................................... 27
      2.3.3 Consistency Techniques in Continuous Domains .............. 28

3. **System Design and Implementation** .................................. 39
   3.1 Techniques, Theory and Algorithm Description ..................... 39
      3.1.1 Introduction ................................................... 39
      3.1.2 Interval Iteration ............................................. 40
3.1.3 Computation as Proofs ................................................................. 45
3.1.4 Interval Constraint Reasoning, an Example ................................. 46
3.1.5 Potential Pitfalls ........................................................................ 47
3.1.6 Consistency Enforcement ............................................................ 49
3.1.7 Solution Space Plotting ............................................................... 52
3.2 SYSTEM ARCHITECTURE .............................................................. 60
3.2.1 Introduction ............................................................................. 60
3.2.2 Constraint Solver ..................................................................... 63
3.2.3 Solution Space Plotter ............................................................... 68
3.2.4 Result Sets Manager ................................................................. 72

4 FATIGUE DESIGN AND ENGINEERING APPLICATIONS WITH QES2 ........ 75
4.1 ENGINEERING APPLICATIONS USING QES2 ................................ 75
4.1.1 QES2 Procedures ................................................................... 76
4.1.2 QES2 Techniques .................................................................... 78
4.2 GENERAL ENGINEERING APPLICATIONS OF QES2 .................... 79
4.3 FATIGUE DESIGN IN STRUCTURAL ENGINEERING PRACTICE .......... 83
4.3.1 S-N Curve Method ................................................................. 84
4.3.2 Using Finite Element Method with S-N Curves ......................... 86
4.3.3 Using Linear Elastic Fracture Mechanics .................................... 88
4.4 STEEL FATIGUE DESIGN WITH QES2 ........................................ 89
4.4.1 Structural Details under Fatigue Loading .................................... 90
4.4.2 Conventional Method of Fatigue Design .................................... 91
4.4.3 Hotspot Stress Calculation Using FEM ..................................... 93
4.4.4 Using QES2 in Fatigue Calculations ........................................ 95

5 CONCLUSIONS .............................................................................. 100

6 APPENDIX A – QES2 USER MANUAL ............................................. 104
6.1 INTRODUCTION .......................................................................... 104
6.2 BACKGROUND ........................................................................... 104
6.3 SYNTAX ..................................................................................... 106
6.4 USER INTERFACE ....................................................................... 107
6.5 SAMPLE ANALYSES ................................................................... 108
6.5.1 Example 1 ............................................................................. 108
6.5.2 Example 2 ............................................................................. 109
III Figures

Figure 1-1 Common engineering design scenario ......................................................... 2
Figure 1-2 2D solution space of $\sin(x^2)+\sin(y^2)<-0.25$ ..................................... 6
Figure 1-3 3D solution space of function $f(x,y)=x^2+y^2$ in cylindrical coordinate system ......................................................... 7
Figure 1-4 Interval constraint and solution space ......................................................... 9
Figure 1-5 Domain narrowing algorithm enforcing consistency (NC3) ......................... 11
Figure 1-6 QES2 system object model ................................................................. 16
Figure 1-7 Net based architecture of QES2 ............................................................ 16
Figure 2-1 A sample numeric CSP ................................................................. 30
Figure 2-2 Box approximation ........................................................................... 33
Figure 3-1 Conventional fixed-point iteration: iterate for solutions of $y' = g(x)$ and $x' = f^{-1}(y')$ ........................................... 41
Figure 3-2 Interval iteration proofing no solution in interval $X$ of $y' = g(x)$ and $x' = f^{-1}(y')$ ........................................... 42
Figure 3-3 Interval iterations resulting in a solution on a narrowed interval of $y' = g(x)$ and $x' = f^{-1}(y')$ ................................................................. 43
Figure 3-4 Inconsistent initial intervals ..................................................................... 44
Figure 3-5 Adaptive 2nd-order and partial global consistency enforcement ............... 50
Figure 3-6 Sampling insufficiency of raster plotting ............................................. 55
Figure 3-7 Interval constraint plotting with different resolution ............................ 55
Figure 3-8 QES2 system architecture layout ....................................................... 60
Figure 3-9 Constraint solver components ......................................................... 63
Figure 3-10 Binary tree representation of a sample constraint set Equation 3-1 ... 64
Figure 3-11 Constraint solver UML ............................................................. 66
Figure 3-12 Constraint expression UML ......................................................... 67
Figure 3-13 Narrowing processor UML ........................................................... 68
Figure 3-14 Illustration of adaptive interval constraint plotting ....................... 69
IV ACKNOWLEDGMENTS

I am deeply indebted to my thesis supervisor, Professor Siegfried F. Stiemer, for his lucid advice and continual encouragement throughout my student career at UBC. He helped me keep my standard high and provided me with the opportunity to explore interesting paths. Special appreciation also goes to Mr. David J. Halliday, Vice President and Adjunct Professor, and Mr. Michael Gedig, P.Eng. at AMEC Dynamic Structures, Ltd. I would like to thank my wife, Jian, for all her support, and my parents, for directing me towards my goals in life.

Furthermore, I am grateful to Professors Ricardo O. Foschi, Mohamed S. Gadala and Reza Vaziri, for their careful directing and review of my thesis. A special thank also goes to Mr. Nathan Loewen for proof reading my thesis.

The financial assistance of Professional Partnership Program with AMEC Dynamic Structures, Ltd., as well as of the Science Council of British Columbia, through a Technology B.C. Grant to AMEC Dynamic Structures, Ltd., are both gratefully acknowledged.
1 ENGINEERING ANALYSIS WITH UNCERTAINTIES AND COMPLEXITIES

1.1 ENGINEERING APPROACHES TO DESIGN PROBLEMS

Good engineering design relies on a working combination of theoretical knowledge and practical experience. Young designers may be well trained in engineering theory, but lack the years of practical experience required to carry out an efficient and sound structural design. In the current environment of a rapidly changing work force, it is often difficult to transfer the invaluable breadth of knowledge gained by experienced engineers to the next generation of designers. Where past experience is not readily available, engineers have to depend on extensive analysis.

Engineers often tend to avoid detailed analysis, such as finite element methods, due to the extra time and effort required. Instead, they often prefer to use simplified design methods specified by codes or standards, which generally rely heavily on judgment calls and experience with complex problems. In addition to the complexity of the analysis itself, many of the input parameters required for a detailed analysis have a degree of uncertainty during the design stage. Handling this uncertainty again calls for engineering experience and knowledge.

Figure 1-1 shows two generalized design scenarios: the scenario on the left relies heavily on design experience, while the scenario on the right relies on extensive analysis. A design problem involving metal fatigue provides a good illustration of the two different approaches. An experienced engineer may carry out a successful fatigue design using simplified design code methods by making appropriate assumptions when selecting critical parameter values. However, for engineers with less experience, the problem can rarely be described by design code approaches with sufficient accuracy or detail, and the engineer may have to rely on extensive analysis. In this case, a suitable
computational design tool could allow the inexperienced designer to overcome the technical or financial constraints of a complicated analysis. Such an engineering tool would need to handle uncertainties as well as complexities, and provide the designer with enough understanding of the problem that they are able to make judgment calls when experience is scarce. The development of these types of tools is one of the current challenges of engineering research. Implementation would ideally include the following features:

- Practical application without detailed computer programming skills
- Inclusion of uncertainties in problem formulation
- Self-optimizing capabilities where desirable properties are inherited from previous analyses (this would replace human expertise to some extent)

Figure 1-1 Common engineering design scenario
1.2 ADVANCES IN COMPUTATIONAL METHODS

In order to solve common engineering problems efficiently, a design algorithm must be able to do the following:

- Handle uncertainties, both qualitatively and quantitatively
- Use a reliable and stable computation process and be logically sound
- Be easy to configure and customize for solving a wide range of different problems
- Require simple input

While many computational methods exist that comply with one or two of the above-mentioned criteria, none of them meet all of the listed requirements. Potential methods for handling these requirements include the application of probability theory, as well as methods used in the field of artificial intelligence (AI) such as image recognition, expert systems, neural networks, fuzzy logic, and qualitative reasoning. The following gives a brief overview of the applicability of each of these techniques to the problem described above:

- **Probability theory** is commonly used to deal with engineering uncertainties, but is difficult to apply to practical problems when statistical data is insufficient.
- **Image recognition** generally does not handle numerical calculations and is consequently not reviewed.
- **Expert systems** are a widely used AI technique for engineering applications, but reason mostly with static knowledge, and are therefore not well suited to dynamic or changing information and numerical calculations.
- **Neural networks** offer an effective way to deal with an analysis where input data are uncertain; however, they require extensive training before they can produce sensible and reliable output [Ref. 44].
• Fuzzy logic has experienced recent developments that enable it to handle uncertain numerical domains, but it still lacks proven and reliable algorithms to carry out numerical calculations [Ref. 27, Ref. 36, Ref. 63, Ref. 55].

• Qualitative reasoning techniques, although less known than other methods under the name of Artificial Intelligence, have been successfully adopted in many real-world engineering applications, such as the simulation of chemical plant operations [Ref. 39]. Using constraint satisfaction techniques, qualitative reasoning is capable of handling numerical logic and calculations. Its key difference from other AI methods can be summarized to its ability to handle numerical computation in a reliable manner.

Of the above methods, qualitative reasoning (QR) appears to have the highest potential for use in the development of practical engineering design tools. Some recent developments show examples of building useful engineering tools with QR techniques [Ref. 31, Ref. 60, Ref. 61]; however, these examples concentrate mainly on non-numerical problems. This thesis focuses on developing numerical applications of QR, and improving efficiencies by adopting new algorithms. Furthermore, a simple user interface will be developed in order to apply QR techniques to engineering problems.

1.3 QUALITATIVE REASONING WITH ENGINEERING UNCERTAINTIES

Any engineering design task can usually be decomposed into a set of relationships and constraints. A fatigue design problem, for example, may be decomposed into functional criteria such as structural life at an allowable stress level. These criteria can be easily represented in terms of inequalities. Since application of relevant geometric and engineering principles is always carried out within the scope of such functional criteria, most important engineering decision-making involves judgments regarding inequalities.
1.3.1 Interval Constraints and Solution Space

Inequality constraints define solutions in the form of solution spaces. Single point solutions are sought in engineering due to the fact that complete solution spaces are too difficult to compute and manage. Traditional media, such as engineering drawings, rely on fixed value assignments for variables. Nevertheless, using solution spaces has the advantage that it can be an extremely helpful basis for engineering judgment and decision-making. Figure 1-2 shows an example of the 2D solution space of \( \sin(x) + \sin(y) < -0.25 \). One can easily see that many non-connected solution spaces are possible. Slight deviation of one parameter value can move the design point from a permitted region to a non-permitted region. Complex 3D solution spaces are more common in real engineering analysis such as the example shown in Figure 1-3. These are even more difficult to understand when only single numerical answers are available to the designer.
Figure 1-2 2D solution space of $\sin(x) + \sin(y) < 0.25$
Qualitative reasoning is capable of deriving the complete solution space from a set of constraints. The key technique used in qualitative reasoning is constraint satisfaction. Engineering tasks are well suited to formulation as Constraint Satisfaction Problems (CSP), which are defined by a set of variables subject to constraints. The variables correspond to the relevant parameters of the design formulas. The constraints express design criteria by equalities or inequalities. The CSP approach uses search methods that detect single variable assignments that satisfy all the constraints, and then provide a description of solution spaces, i.e. the set of the entire solutions. Early applications of CSP research emerged from the field of image interpretation. Mackworth and Montanari [Ref. 46] generalized this concept to any kind of discrete data related by binary constraints, called discrete CSPs.
Consistency techniques are one of the methods used for solving CSPs. They provide filters that remove inconsistent values (i.e. values that cannot be part of a solution) from the search space, and thus make the search more efficient. Consistency techniques can be used as preprocessors in order to transform a given CSP into one that is simpler to solve. This approach has led to the identification of restrictions on the constraint syntax, the topology of the constraint network, and the solution space described by constraints. Most techniques for CSPs involve decomposing constraints into a tree with nodes representing variables and branches representing constraints. Solving a CSP is a process of reasoning through such a tree. Reasoning in the direction from end nodes to tree root is called backtracking. Research has shown that the degree of consistency reached in a CSP is related to the level of backtracking needed to solve it [Ref. 6]. Other work has concentrated on applying these techniques during the solution search process [Ref. 60]. In addition to the degree of consistency used, heuristics* on the order in which the variables are instantiated determine the execution time.

Initially, most research concentrated on discrete binary constraint problems [Ref. 46]. Only recently have search techniques developed for discrete problems been successfully applied to numeric CSPs. Value combinations in numeric CSPs are not enumerable; therefore, interval analysis is employed to achieve partial degrees of consistency. Sam-Haroud and Faltings [Ref. 61] extended the results of van Beek and Dechter to numeric CSPs [Ref. 16], and present convexity conditions under which a numeric CSP becomes backtrack free. Since this algorithm relies on an explicit representation of the solution spaces, the computing time required may be tremendous. In practical situations using this approach, it is typically applicable to problems involving less than twenty variables. Mittal and Falkenhainer [Ref. 48] have defined a generalization of the standard CSP, where the static CSP

*In computer science, a heuristic is an algorithm or procedure designed to solve a problem that ignores whether the solution is probably correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem. A heuristic is not guaranteed always to solve the problem, but often solves it well enough for most uses, and often does so more quickly than a more complete solution would.
formalism, in which the variables and constraints of a CSP are given in the problem definition, is extended to the definition of conditional CSP. However, no practical proposals exist for efficient handling of continuous variables, even though most engineering tasks are constrained by numeric CSPs.

Properly formulated local consistency techniques provide useful support for design tasks when many interdependent variables of a discrete and numeric nature are involved. The local consistency techniques are able to rapidly approximate consistent spaces with a reasonable degree of accuracy. When accuracy is essential, this approach provides an effective filter that is capable of yielding infeasible solution values. These results can be used as the input for subsequent search or optimization methods. However, extensions are needed to conditional CSPs allowing for a systematic enumeration of the solution spaces, leading to a clarification of the overall complexity of solution space sets. As in the context of standard CSPs, local consistency is applied to remove entire spaces that are inconsistent.

Design Variables

\[ V_1 = [V_{1,L}, V_{1,R}] \ldots V_n = [V_{n,L}, V_{n,R}] \]

Design equations/
constraints

\[ f_{1,L}(V_1 \ldots V_n) = f_{1,R}(V_1 \ldots V_n) \ldots f_{n,L}(V_1 \ldots V_n) = f_{n,R}(V_1 \ldots V_n) \]

Solution/Design
Space

Figure 1-4 Interval constraint and solution space
While typical applications of CSPs deal mainly with discrete or binary data, numeric CSPs (also known as continuous constraint systems, or interval constraint systems) are the focus of interest for engineering applications. Interval constraints were first introduced by J. G. Cleary [Ref. 12] in the 1980s. The initial goal was to address the error of floating-point numerical computations in the Prolog programming language, while placing formal arithmetic in closer relation to the formal language model. Interval constraint processing combines propagation and search techniques developed from artificial intelligence with methods from interval analysis. This idea is also known as interval propagation. Given a set of constraints $C$ involving variables $v_1 \ldots v_n$ and a set of floating-point intervals $V_1 \ldots V_n$ representing the domains of possible values, a reasoning procedure isolates a set of regions $R_1 \ldots R_n$ approximating the constraint system solution (Figure 1-4). To compute such a set, a search procedure navigates through the initial intervals $V_1 \ldots V_n$, alternating pruning and branching steps. The pruning step employs a relational form of interval arithmetic developed by R.E. Moore in the 1960s [Ref. 50]. Given a set of constraints imposed on real numbers, interval arithmetic is used to compute local approximations of the solution space for a given constraint. This results in the discarding of values from the initial variable domains. These domain modifications are propagated through the entire constraint set until reaching a stable state. This is closely related to the notion of arc consistency [Ref. 45], which is a well-known concept in artificial intelligence. The branching step consists of a bisection-based divide-and-conquer procedure on which a number of strategies and heuristics can be applied. The pruning step, also called interval narrowing or interval propagation, is a numeric implementation of local arc consistency. The pseudo-code in Figure 1-5 illustrates the typical algorithm for such a pruning step, which essentially narrows the input domain so that it satisfies all the constraints specified.
function NC3 (S : constraint set, B : domain) : domain
% Output domain is necessarily included in the input domain
begin
    Queue all constraints from S in Q
    repeat
        Select a constraint c from Q
        $B' \leftarrow \text{narrow}(B, c)$ % Narrow down B with respect to c
        if $B' = \emptyset$ then return $\emptyset$ % Inconsistency of the constraint system
        Let $S' = \{c \in S | \exists v \in \text{Var}(c), B'_v \subseteq B_v\}$
        % Queue the constraints whose variables' domains have changed
        $Q \leftarrow (Q \cup S') \setminus \{c\}$ % Delete c from Q
        $B \leftarrow B'$
    until Q is empty
    return B
end

Figure 1-5 Domain narrowing algorithm enforcing consistency (NC3)

A system of constraint interval arithmetic consists of three distinct layers. The top layer is concerned with the conversion from the external source language to an internal data structure or constraint network. The middle layer handles the interval iteration and relates the properties of the primitive operations to that of the constraint network as a whole. The bottom layer is a simple abstract theory for the implementation of primitive functions such as addition, subtraction, multiplication, etc. This thesis focuses on development of the top layer for the application of integrating computations of metal fatigue design, as well as the middle layer for the effective implementation of intensive engineering calculations that involve a large number of variables and constraints. A number of computational primitives necessary for engineering calculations are developed for the bottom layer.

The focus is placed on solving engineering design problems using numerical qualitative reasoning, i.e. numerical constraint satisfaction techniques. The advantage of using CSPs in this field is that consistency techniques represent an approximation of both input variables and solution spaces instead of single point values. In this context, the following methods are implemented:
• The approximation of solution spaces is achieved by an innovative local consistency method for numerical variables providing good results in pruning and execution time. This is achieved by using a local consistency operator for numerical constraints, which is superior in pruning power to existing methods.

• A novel search method using local consistency for numerical variables is implemented. In contrast to most existing approaches for solving mixed CSPs that are based on a cooperation between constraint solvers, this method integrates the local consistency methods for numerical variables into the search process, and also makes use of the mixed constraints to prune the search space.

1.4 PROSPECT FOR FURTHER RESEARCH

The latest developments in engineering design have shown a trend towards introducing high complexity into design formulations, with the intention of achieving high design reliability. However, the lack of tools to deal with such complexity, along with the uncertainties embedded in the design, often hinder the engineer’s insight into the true nature of the problem under investigation. Qualitative reasoning provides a potentially effective and sound technique through which complex scenarios, such as those commonly faced in metal fatigue design, can be solved regardless of uncertainty and linearity of the design parameters and their constraints. Additionally, the ability to represent design parameters and outcomes in a 2D space would provide engineers with a valuable design tool.

1.4.1 Fatigue Design of Steel Structures Using Qualitative Reasoning

Fatigue cracking problems occur in steel structures that are subjected to cyclical loading, and great care must be taken to develop fatigue-resistant design details. However, the complex nature of metal
fatigue prohibits intuitive design. Engineers must use a combination of experience and analysis techniques in order to design details that achieve good fatigue resistance. This usually involves starting with a design code check of stress ranges and allowable stress limits. Often the code cannot cover all of the particular details under consideration because of the limits in scope, in which case the engineer may decide to extend the analysis into an investigation of local stresses using finite element tools to better observe the effects of stress concentration. Even after this level of analysis, the design can still not be considered "fatigue-proof" because many essential factors (such as residual stress in a welded joint or weld defects) simply cannot be revealed by any analysis.

A low reliability of the input variables and parameters used in fatigue analysis can potentially have a large effect on the design solution due to the complex relationships between the equations. If the uncertainty in the input data could be taken into account by allowing input variables to be presented as a range of values as opposed to a single value, it would enable representation of the solution as a design space, therefore allowing engineers to realize a more reliable design that satisfies the entire range of input values.

1.5 CONTRIBUTION OF CURRENT RESEARCH

This thesis develops a software framework capable of accommodating complexity and uncertainty in engineering calculations by using techniques in qualitative reasoning and adaptive graphing. Recent advances in qualitative analysis have begun to apply reasoning techniques to continuous domains, but still lack efficiency in dealing with large quantities of arbitrary numerical constraints, such as those given in engineering problems [Ref. 25, Ref. 61]. The contribution towards solving this problem as presented in this thesis reduces such inefficiency by retrofitting existing qualitative methods with interval arithmetic reasoning and optimized consistency algorithms. This presents a major step in the direction of applying qualitative techniques to everyday engineering.
Below is a summary of the main areas in which this thesis makes advances:

1. The techniques of qualitative reasoning, interval arithmetic, and adaptive graphing are integrated. This provides advancement over existing tools, which are inefficient for handling large numerical problems and are incapable of accommodating problems of generic forms with complexities and uncertainties.

2. Engineering problems are solved using qualitative reasoning. Conventional techniques are efficient only in binary and discrete domains. Existing techniques in interval arithmetic reasoning are effective, but may encounter difficulty in convergence for certain problems. This thesis introduces higher order consistency techniques, as well as additional rule sets, to the standard interval-based techniques, which use simultaneous constraints over logical and numerical operators to enable stability in solving large numerical problems. Common functions in engineering calculations, such as the $\text{if()}$ operator, are also developed and expand upon standard, theory inclined, interval arithmetic libraries.

3. A new method of metal fatigue design is introduced. This thesis proposes and illustrates a procedure to deal with fatigue design using multiple design codes, and potentially complex formulations including variable uncertainties. The developed tool provides an unprecedented way to increase the understanding of complexities in engineering conceptual design. [Ref. 20, Ref. 26]

1.5.1 Qualitative Reasoning Software

One of the key outcomes of this research is a software framework named Qualitative Engineering System 2 (QES2). QES2 is capable of analyzing engineering problems where the input is expressed in the form of equations, constraints, and design variable intervals, and the results are depicted in the
form of a solution space. Figure 1-6 shows the main object components and the structural arrangement within QES2. All variables and relationships that are expressed in equations are represented as objects with properties and behaviors. In addition to equations, parent and child links connect objects for knowledge exploration. Bounds and mean values are taken as the basis for numerical calculation and reasoning, but no weighting is given to individual values. Definitions and references in the form of objects and relationships are stored dynamically in the system. Through interacting with engineers, the system acts as a knowledge depot, in this case for metal fatigue design. All relevant results can be packaged so that they are easily accessed by inexperienced engineers.

Figure 1-7 illustrates the architecture of an engineering system based on QES2. The system server combines several solvers that deal with reasoning and computation, while manipulators are in charge of handling information. A net-based platform makes the stored knowledge and analysis available anywhere with Internet access, potentially enabling knowledge transfer between practitioners of various experience and knowledge levels.
Figure 1-6 QES2 system object model

Figure 1-7 Net based architecture of QES2
2 Theories and Algorithms

Resolution techniques for numerical reasoning with uncertainties encompass a number of different areas including numerical analysis, linear and non-linear programming, interval arithmetic, evolutionary programming, consistency algorithms and reliability analysis. To determine the best way in which to deal with uncertainties in engineering, a complete spectrum of analysis techniques must be investigated. The first part of this chapter reviews related works, and briefly sketches out the prominent techniques from each discipline, in order to set a comparative framework. The works reviewed have been chosen for their contribution to efficient algorithms and practical implementations. The latter part of this chapter will present the techniques and literature directly related to the work of this thesis in more detail.

2.1 Conventional Techniques for Modeling Uncertainties in Engineering

Engineering problems are essentially a combination of equations and variables that have embedded uncertainties. These equations can be generally classified as linear or non-linear, and are numerically solved with linear and nonlinear programming methods, algebraic and iterative numerical methods, as well as simulation processes.

Linear systems, with disposal efficiency as a main consideration, are often solved using Danzig's simplex algorithm [Ref. 15] from linear programming, and Gaussian elimination from linear algebra. Linear programming methods [Ref. 64] provide optimal solutions for a set of linear equations and inequalities with respect to an objective function. Linear algebra uses a systematic solution method that is based on iterative syntactic replacements, and works for any solvable linear problems. The method utilizes Gaussian elimination for equalities, and Fourier-Motzkin method for inequalities. These methods for solving linear systems have been widely implemented in many mathematical
packages such as Mathematica [Ref. 47] and Maple [Ref. 73], and in numerous reasoning or Constraint Logic Programming (CLP) languages.

Non-linear systems prove to be more difficult to solve using conventional engineering and mathematical treatments. Various methods have been developed for solving non-linear systems, and each method has some limitations: algebraic methods are restricted to polynomials, numerical methods offer no convergence guarantees and may exhibit chaotic behaviors, and reliable nonlinear programming methods (often classified as descent and direct methods [Ref. 43]) are restricted to convex solution spaces. These limitations prohibit the usefulness of these techniques. Certain practical systems, such as many CLP programming languages, avoid the limitations of these methods by introducing additional restrictions to the types of problems that can be solved:

- Constraints are restricted to be linear, such as in software packages CLPR and CHIP [Ref. 70],
- The evaluation of non-linear terms is delayed in hope that they become linear after the linear terms have been solved separately, such as in Prolog III [Ref. 14],
- A linear approximation is constructed for certain types of quadric constraints that are easy to enclose [Ref. 14].

2.1.1 Reliability/Probabilistic Methods

Probability theory is commonly used by the engineering community for dealing with uncertainties. This is one of the basic principles of most modern design codes, and provides the means to carry out analysis when input parameters are not accurate enough to be described by single values. In probabilistic methods, the input parameters are defined by statistical data such as the mean, standard deviation, and distribution type. Design codes use probabilistic data for general design cases. For critical design or research projects, resources are available to carry out sufficient testing and investigation to obtain the required statistical data for the specific project. However, typical design
projects in structural engineering are time critical and resource constrained; when these types of projects cannot be sufficiently handled by design codes, probability method may become difficult to apply due to lack of statistical data.

2.1.2 Algebraic Algorithms for Nonlinear Constraints

In the last three decades, the development of algebraic methods for solving numerical non-linear constraints has focused on two methods: the Grobner bases method for equalities [Ref. 33], and the partial cylindrical decomposition for inequalities. Both methods are intended for polynomial constraints or equations.

Grobner bases were introduced by Buchberger [Ref. 9] to transform a set of polynomials into a standardized form that has good computational properties. This concept facilitates the detection of insolvability in systems of equations. Several additional works have led to improvements and generalizations of the Grobner bases method. Sato and Aiba [Ref. 62] reported the successful application of Grobner bases to solve complex kinematics problems in tens of seconds.

Solving a system of nonlinear inequalities is formally defined as a quantifier elimination problem in the first order theory of real fields, for which Tarski [Ref. 32] proposed the first algorithm. The cylindrical algebraic decomposition algorithm has been proven to be the most practical and theoretically useful of these algorithms. Improvements in recent years have increased the efficiency of the algorithm by a factor of $3 \times 10^5$ over its initial formation.

Algebraic techniques offer the major advantage of compiling solutions for certain types of constraints, and also provide a run time significantly faster than iterative techniques. However, they are plagued by a poor scalability, and their application is limited to polynomials. This explains why
in practice algebraic techniques tend to be more useful as solution checkers rather than solution generators [Ref. 37].

2.1.3 Iterative Numerical Algorithms

The most compelling advantage of iterative numerical techniques is their generality. In effect, they are theoretically applicable to any problems that can be represented as a set of equations, and are therefore applicable to most engineering problems. Numerical iterative techniques deliver pointwise approximations calculated through a series of refinements that converge toward the exact solutions. The vast majority of iterative techniques are devoted to unconstrained minimization, of which the simultaneous nonlinear equation problem is a particular instance [Ref. 17]. Less is known about how the constrained problems can be solved where each variable is subject to restriction. Numerical iterative techniques typically require methods of evaluating the problem functions combined with an approximate solution that provides a starting point for the evaluation process. These evaluation methods can be of two forms:

- Global or bisection methods: these methods are used when the starting point is not close to the final solution. They are able to guarantee convergence to a local point of a nonlinear function, or some solution of a system of nonlinear equations from almost any starting point.
- Local methods (Newton, Broyden, Gauss-Jacobi, Gauss-Seidel [Ref. 66]): these methods can be used only when the starting point is in the vicinity of the final solution.

It is recognized that pure global methods are usually too inefficient for general use, and the issue of effectively combining local and global methods is critical for managing the trade-offs between efficiency and reliability. The most significant shortcomings of iterative numerical methods are as follows:

- The solutions found are correct within a convergence tolerance. The non-existence of a solution can only be inferred through a failure to converge after a fixed maximum number of iterations.
Predictability in terms of convergence and its speed cannot be guaranteed since the number of iterations required to solve a problem can vary substantially based on the starting point provided.

Iterative solvers cannot distinguish multiple solutions that may jump from one solution space to another, and possibly lead to chaotic behavior.

Iterative solvers require computation of partial derivatives generally obtained through finite difference methods.

In spite of the shortcomings, iterative methods are among the most widely used for dealing with nonlinear engineering problems because of their generality. Numerical techniques have been extensively studied for decades and many believe that although some improvement is still likely, quantum jumps in the quality of the algorithms are unlikely [Ref. 17].

2.1.4 Stochastic Methods

Stochastic methods, as the name implies, involve a certain degree of randomness. This refers to a class of generally incomplete and nonsystematic search techniques including heuristics and non-determinism. Examples of such techniques are hill climbing, simulated annealing and genetic algorithms. These techniques can be considered as adaptive in the sense that they start with a random focal point in the search space and modify it repeatedly using heuristics until it reaches a solution after a certain maximum number of iterations. These methods are generally rather robust when finding a global minimum in large and complex search spaces. A situation may occur where the searching process is occurring in the wrong portion of the solution space, which is dealt with by restarting the process from randomly selected points, therefore reducing the chance of convergence to local minima. However, a nonsystematic search of pointwise solutions does not allow a reliable method of locating feasible alternative solutions in the solution space. Successful use of such
methods has been reported to solve large systems of nonlinear inequalities derived from mechanical
design problems [Ref. 68].

2.2 INTERVAL ARITHMETIC

The first practical applications of interval arithmetic are dated back to Moore in the 1960s [Ref. 50].
While the original purpose of interval arithmetic was to quantify the errors introduced by finite
precision arithmetic such as floating-point computation in computers, it also appeared to constitute an
appropriate framework for dealing with uncertainty [Ref. 51]. Interval arithmetic techniques address
a wide variety of problems such as linear and nonlinear interval equations [Ref. 7], as well as
optimization [Ref. 58]. In interval arithmetic, real numbers are approximated by intervals, and
arithmetic expressions are computed by applying the operators of the formula to the end points of the
intervals of its arguments, for example:

\[ [2, 4] - [1, 2] = [0, 3] \]

Most properties of arithmetic operators such as commutativity and associativity carry over to their
interval counterparts. However, distributivity does not hold in general and is replaced by a weaker
property called sub-distributivity:

\[ A(B + C) \subseteq AB + AC \]

This explains why naive evaluations of interval arithmetic expressions often result in strong
overestimates. For the following example, the determination of the order of operations for obtaining
the best computable enclosure can be a quite complex task for arbitrary expressions.

\[ A(1-A) \subseteq A - A^2 \]

Systems of linear algebraic interval equations can be solved either by direct matrix inversion or by
using the interval counterparts of the standard iterative algebraic techniques. The former consists
merely of computing the direct inverse solution using interval matrix inversion. A variant of direct
inversion is also given by the Krawczyk's nonlinear operator [Ref. 38], which often yields sharper results when applied iteratively. These simple methods require the regularity of the matrix to be inverted, a condition which does not hold in general, and are consequently too restrictive for general use [Ref. 52]. A more general alternative consists of applying the extension of Gauss-Seidel or Gauss-Jordan [Ref. 59] iterations to interval systems, and eventually replacing the iterative solution process by direct solutions when regularity is detected. Systems of linear interval equations can also be solved using the interval extension of Gaussian elimination, but the algorithm requires preconditioning and can be carried out reliably only when the matrix is strongly regular. When these conditions hold, Gaussian elimination remains the preferred solution.

Most of the resolution methods for systems of nonlinear equations are direct extensions of the standard numerical iterative techniques as discussed in section 2.1.3. Interval arithmetic counterparts exist for many locally convergent numerical methods such as Newton methods [Ref. 2]. The Krawczyk's nonlinear operator is often used to solve a step of the Newton interval. Combined with interval Newton method, this operator has indeed proven to be useful for producing good evaluations of complex interval functions. Most of the state of the art interval arithmetic techniques are direct successors of the Krawczyk's method. Bisection remains the main globally convergent technique used in interval arithmetic. It consists of recursively bisecting the original domain in the direction corresponding to the largest interval, while trying to remove the sub-domains having no intersection with the interval enclosure of the effective solution space. Similarly to the case of globally convergent numerical techniques, the slow convergence of bisection precludes its utilization for solving practical problems alone. This method often complements locally convergent iterations, notably to identify multiple solutions.

Interval arithmetic methods compile compact enclosing approximations of the solution space. They are consequently appealing when it is needed to reason on a spectrum of possible feasible
alternatives, rather than on single point isolated solutions, which are often encountered in engineering applications. These methods tend to emphasize completeness rather than absolute soundness since the interval enclosing representation of the solution space loses pointwise relations between the values of the variables. From the standpoints of efficiency and robustness, interval arithmetic techniques present roughly the same advantages and shortcomings as iterative numerical techniques. Their major disadvantage remains that special care must be exercised in advanced applications in order to prevent unrealistic estimates:

- When a constraint expression contains multiple occurrences of the same variable, simple interval evaluation often results in excessively loose bounds. A simple expression such as $X - X$ can generally not be recognized as zero.
- In general there is no guarantee that the sequence of intervals generated by locally convergent operators will converge at all. Hence, the determination of appropriate initial domains is of crucial importance for nonlinear systems [Ref. 51]. This problem, generally addressed using bisection, is analogous to the choice of suitable starting conditions for iterative numerical techniques.

Finally, both linear and nonlinear algebraic interval methods require preconditioning steps or sophisticated applicability conditions.

### 2.3 Numerical Consistency Techniques

The approaches previously presented handle the entire set of constraints as a whole and therefore provide global solutions, whereas consistency techniques are based on the construction of partial solutions that are recurrently extensible, allowing more flexibility in the solution process. In effect, the advantages in terms of accuracy and soundness provided by global solution constructors are often mitigated in practice due to the following reasons:

- They require rigid representations that are difficult to handle by non-experts.
• They pose scalability problems.
• They often involve complex processes where the user has no control over the decision mechanisms.

Constraint based approaches in AI can be traced back to the development of the first interactive graphical interfaces using geometrical constraints in the 1960s. More recently, several CAD systems have used constraints as problem specifications. The first formalizations of constraint satisfaction problems and techniques were issued from research on line labeling in vision [Ref. 71] which resulted in a basic consistency technique called the Waltz filtering algorithm, later referred to as AC-2. The research on image processing led Montanari to propose the first systematic treatment of constraint networks along with a path consistency algorithm called PC-1. This work initiated abundant research in constraint satisfaction problems [Ref. 61].

Constraint satisfaction problems are generally represented using graphs in the computational sense, depending on whether the constraints are binary (pairs of variables) or n-ary (n-tuples of variables). In these graphs the nodes represent the variables involved in the problem while the arcs represent constraints between variables. The vast majority of research on CSPs is devoted to binary problems, and less is known about how n-ary problems should be solved. Non-binary discrete problems can be mapped to binary ones with generally more complex variable domains by using reduction methods. On the other hand, the simple transformation method based on projection applies to general problems but corrupts the effective solution space. A static CSP is a problem where both sets of variables and constraints are entirely pre-determined and do not vary during the solution process. In contrast, a dynamic CSP may see some constraints added or removed during the problem solving process, or may be subject to dynamic constraints where the constraints themselves are subject to additional constraints [Ref. 25]. In a CSP, a distinction is made between the domain of a variable and its label. The domain refers to the whole set of values the variable can take, while the label represents the
subset of the domain assigned to the variable at a given time. Both labels and domains can be either discrete or continuous.

2.3.1 General Consistency Techniques

Problem reduction techniques are often referred to as consistency techniques. Backtrack search, along with its numerous variants and enhancements, remains the principal mechanism for solving a CSP. However, since it is often the source of intractable complexities, problem reduction has been introduced to transform the initial problem into an equivalent problem that is easier to solve. Problem reduction consists of pruning from the variable domains that are locally inconsistent, thus reducing the solution space as well as the subsequent search effort. These methods exhibit polynomial time complexities, where the order of the polynomial is a linear function of the number of levels of local consistency to check for. This notion of problem reduction was introduced in the literature along with formal definitions of local degrees of consistency in constraint networks, and local consistency checking algorithms. Several degrees of local consistency have been defined: node, arc and path consistency as defined by Mackworth [Ref. 90] and generalized to k-consistency and strong k-consistency as defined by Freuder [Ref. 46].

**Definition 2-1: k-Consistency**

A network is k-consistent if and only if, for any given instantiation of any k - 1 variables satisfying all the direct relations among those variables, there exists an instantiation of the k-th variable so that the k values taken together satisfy all the relations among the k variables [Ref. 23].

**Definition 2-2: i-Consistency**

A network is strongly i-consistent if it is i-consistent for all i ≤ k. [Ref. 23]

Strong 2-consistency and strong 3-consistency correspond respectively to arc and path-consistency. From the algorithmic point, arc-consistency techniques have been the subject of particular focus and attention. Since the first arc-consistency algorithm was introduced by Waltz [Ref. 71], numerous
variants have been proposed such as AC-1, AC-2, AC-3 [Ref. 46], and have been further improved in the discrete domain to AC-4, generalized to AC-5 and culminating with AC-6 [Ref. 60]. The first path consistency algorithm PC-1, further refined to PC-2, was proposed by Montanari [Ref. 49]. In the discrete domain, PC-2 is again improved to PC-3.

Algorithms for higher degrees of consistency than arc consistency require tools for synthesizing constraints by composition (i.e. inferring constraints of higher arities than the ones originally present in the network). This notion has been introduced by Freuder [Ref. 23] who proposed the first synthesis algorithm, which was later refined by Cooper [Ref. 49]. The process of inferring the constraint of arity $n$ ($n$-constraint) for a problem of $n$ variables by extending partial solutions is also referred to as solution synthesis. Solution synthesis is generally demanding in terms of computational resources. An acceptable compromise between the effort devoted to local pruning and the cost of subsequent searches must be sought when using problem reduction techniques. In practice, the most commonly used algorithms are those enforcing arc and path consistency.

2.3.2 Preprocessing Methods

Simplifying properties and ensuring a backtrack-free search are essential requirements of an efficient search. Extracting a particular solution from a consistent labeling is a two-step iteration, in which values are assigned to variables sequentially. In the first step, an unassigned variable is selected and a value is assigned to its label. In the second step, the labels of all remaining unassigned variables are updated so that they contain only values that are consistent with those already assigned. If the initial labeling is globally consistent and non-empty, every partial assignment of variables can be extended to a full solution. Consequently, the assignment procedure will not require any backtracking. Computing globally consistent labeling amounts to synthesizing the $n$-constraint for an $n$-variable problem, a task that exhibits an exponential time complexity in the most general case. However, provided that certain conditions are verified, local consistency algorithms are sufficient either to
compute a globally consistent solution space or to set boundaries on the remaining search effort. For example, it has been shown [Ref. 49] that when the constraint network is a tree, arc consistency also guarantees global consistency. This is an example of how the topology of the constraint network can simplify the computation of a globally consistent labeling. Similarly, path-consistency is sufficient to ensure backtrack-free search in series-parallel networks. It has also been shown that path-consistency ensures global consistency irrespective of the topology of the network when the constraints themselves are convex and binary [Ref. 16].

The complexity of consistency algorithms may also be reduced by using domain specific characteristics [Ref. 46]. Ensuring node and arc consistency are stronger than required enables backtrack-free searching in tree-structured networks. Consequently, a weaker concept is proposed called directional arc-consistency that guarantees backtrack-free search in trees. This notion generalizes to higher degrees of consistency and is defined under total ordering of the variables.

In addition to problem reduction techniques, the complexity of constraint satisfaction problems can be reduced using the methods listed below [Ref. 60]:

- **Graph-reduction techniques**: the constraint graph is simplified prior to search using reduction techniques such as vertex elimination or vertex merging.
- **Abstraction techniques**: the size of the search space is reduced by clustering values, variables or both.
- **Decomposition methods**: topological properties of the constraint graphs are used to propose decomposition schemes that are able to reduce the complexity of the problems.

### 2.3.3 Consistency Techniques in Continuous Domains

Early attempts to establish constraint propagation as appropriate technology for continuous domains have shown little success. Traditional consistency techniques and propagation algorithms, such as the
Waltz propagation algorithm, provide relatively poor results when applied to continuous domains since they ensure convergence and completeness for only restricted classes of constraints. When nonlinear and transcendental constraints are involved, the behavior of these algorithms becomes unpredictable and is often plagued by two problems:

- Early quiescence: the refinement process stops prematurely and no consistency is ensured.
- Cycling: the refinement process enters infinite loops.

These negative results have historically been attributed to the analytical complexity inherent in solving general systems of equations and inequalities. Recent advances [Ref. 58] have proposed various remedies and significant improvements to mitigate these problems. Since composing continuous constraints numerically is a complex task, most of the techniques for continuous CSPs work with the unary projection of constraints rather than the constraints themselves, and the vast majority of propagation techniques are variations of the same theme: fixed-point iteration via arc-consistency algorithms with emphasis on tightening the outer domain bounds for variables. Constraints are generally approximated by the feasibility domains of the variables they involve, and the combination of constraints then reduces to elementary operations on intervals.

A numeric constraint satisfaction problem (numeric CSP) is a conjunction of real constraints $c_1, \ldots, c_m$ on a set of variables $\{v_1, \ldots, v_n\}$ to which are associated interval domains of possible values $\{I_1, \ldots, I_n\}$. The solution space of the CSP (Figure 2-1) is the $n$-ary relation over $\mathbb{R}$, which is defined as the intersection of the relations and the Cartesian product of domains $I_1 \times \cdots \times I_n$. The solving process of a numeric CSP consists of isolating a set of $n$-ary boxes included in the search space, and approximating the solution space. To compute such a set, a search procedure navigates through the search space alternating pruning and branching steps. In the framework of interval constraints, the
pruning step, also called interval narrowing or interval propagation, is implemented by local consistency algorithms that are described in the next section.

\[ \begin{align*}
  y &\leq x^2 \\
  x^2 + (y-2)^2 &\leq 1 \\
  x &= [-1.5, 1.5] \\
  y &= [1, 1.5]
\end{align*} \]

Figure 2-1 A sample numeric CSP

The call to Function `narrow()` in NC3 (Figure 1-5) is an algorithmic narrowing process. A constraint narrowing operator (CNO) is associated with the underlying relation \( \rho_c \) of each constraint \( c \) in \( S \). CNOs are complete, contracting, monotone, and idempotent operators mapping boxes to boxes. For every box \( B, B' \) the following relations can be made:

\[
\begin{align*}
  B \cap \rho &\subseteq N_c(B) \quad \text{(completeness)} \\
  N_c(B) &\subseteq B \quad \text{(contractance)} \\
  B \subseteq B' &\Rightarrow N_c(B) \subseteq N_c(B') \quad \text{(monotonicity)} \\
  N_c(N_c(B)) &\subseteq N_c(B) \quad \text{(idempotence)}
\end{align*}
\]

The algorithm stops when a stable state is reached, i.e. no narrowing is possible with respect to any constraint. The result of the main step is to remove incompatible values from the domains of the variables occurring in \( c \). The main properties of NC3 are the following:
- It terminates in a finite time, since each step is contracting and the inclusion relation over an approximate domain is a well-founded relation.

- It is complete (the final box contains all solutions of the initial system included in \( B \)), since each narrowing operator is complete.

- It is confluent (selection of constraints in the main loop is strategy independent) and it computes the greatest common fixed-point of the constraint narrowing operators that is included in the initial box [Ref. 6].

Different constraint narrowing operators may be defined that result in different local consistency notions. A system is said to be locally consistent with respect to a family of constraint narrowing operators if the Cartesian product of its variables' domains is a common fixed-point of the constraint narrowing operators associated with its constraints. The main local consistency notions used in numeric constraint satisfaction problems are first order local consistencies derived from arc consistency (hull consistency or 2B consistency), box consistency, and higher order local consistencies derived from k-consistency (3B consistency, kB consistency, box-2 consistency) [Ref. 6].

2.3.3.1 Numerical Arc-consistency

The improvements of arc-consistency techniques all exploit features inherent to discrete solution spaces. The introduction of explicit counters and supports for the label values is an example of this. They are consequently not directly applicable to the case of continuous CSPs. This explains, from an algorithmic standpoint, why propagation algorithms have not evolved in continuous domains and still take their general forms from AC-3. Some research efforts have focused on the issue of enhancing the constraint revision operators used during the relaxation steps of consistency algorithms [Ref. 61]. The first practical implementations limited themselves to using basic interval arithmetic operations for refining the interval labels through fixed-point iterations. These implementations include BNR
(Prolog) and its successor CLP(BNR), CLP(R) [Ref. 54]. Since elementary interval arithmetic operations often produce unrealistic estimates, these implementations are relatively weak. They fail to narrow the labels for the following simple linear problem:

\[
\begin{align*}
X + Y &= 5 \\
X - Y &= 6 \\
X, Y &\in (-\infty, \infty)
\end{align*}
\]

Additionally, they cannot detect the following types of inconsistencies:

\[
\begin{align*}
A + 1 &= D \\
A + B &= D \\
A &\in (0, \infty), \ B &\in (-\infty, 0)
\end{align*}
\]

They also assume a single, non-splitting label, or an interval per variable. Variants of these systems are as follows:

- A specific adaptation is made to particular types of constraints, such as in CLP(F) which is devoted to functional constraints and ensures a better narrowing, in some cases, by putting interval constraints on the values of the constraint derivatives at points and intervals.

- The capability of handling multiple solutions is introduced. An example of this is the representation of real domains by interval hierarchies, which allows control of the constraint processing precision and provides a hierarchical version of consistency algorithms. The narrowing operators are based on elementary interval arithmetic, and the results provided for the two previous examples are no better.

Given a numeric CSP, an interval narrowing process generates a box approximating its solution set (Figure 2-2). In the interval constraint framework, this approximation is generally computed by applying the algorithm described by NC3 (Figure 1-5) to reflect its similarity to the arc consistency algorithm AC3.
2.3.3.2 Numerical search

A first remedy to the narrowing failure described in the last section is proposed by Cleary [Ref. 12] who uses a case analysis technique based on domain splitting. Cleary considers relational arithmetic on continuous domains by adapting techniques from interval arithmetic. These techniques are introduced into Prolog\(^\dagger\) for narrowing the intervals associated with constraints. Narrowing is done using specific algorithms adapted to each kind of constraint, and these techniques are coordinated.

\(^\dagger\) Prolog [Ref. 18] is a leading logical programming language. It was first created by Alain Colmerauer [Ref. 13] around 1970s, attempting to make a programming language that enables to express logic, instead of carefully specifying instructions on the computer.
with a backtracking tree search. Cleary's algorithms have been generalized for narrowing intervals constrained by any relation on a continuous domain and incorporated in the constraint logic programming languages CHIP and CLP(R).

2.3.3.3 Hybrid techniques

Since domain splitting and the subsequent backtracking tree search are computationally demanding, certain constraint logic programming languages, such as CIAL [Ref. 58] limit the use of propagation techniques to nonlinear constraints and use powerful mathematical tools for linear constraints. CIAL includes a linear solver based on preconditioned Gauss-Seidel method in addition to an interval narrowing module devoted to non-linear constraints. When some variables become impossible to narrow, they are considered as constant terms, which might render some nonlinear constraints linear and reactivate the linear solver. Such hybrid implementations offer significant improvements for certain types of problems. Some authors have also attempted to improve the results of interval-based consistency techniques by combining them with assortments of numerical tools such as relational and constant elimination arithmetic symbolic algebra, hierarchical strategies for handling inequalities, or domain-specific interval propagation tactics [Ref. 61].

2.3.3.4 The state of the art

The most significant improvements of arc-consistency techniques in continuous domains are given by the works of Lhomme [Ref. 40], Benhamou et al. [Ref. 6] and Faltings [Ref. 6]. The two former works present several analogies while the latter deals with the issue from a different perspective and analyzes the cause of poor performance of traditional arc-consistency techniques in continuous domains.

Lhomme [Ref. 48] proposes an interval propagation formulation based on bound propagation. The new consistency concept, referred to as 2-B-consistency, is weaker than arc-consistency, and assumes
the convexity of variables' domains. In the case where the effective domains are disjunctive, 2-B-consistency will consequently admit local inconsistencies within the labels. In Lhomme's approach, constraints are decomposed into primitive or basic constraints allowing an easy definition of extrema functions. The relaxation step of the 2-B-consistency algorithm, analogous to AC-3, uses these extrema functions to refine variables' domains with fixed-point iterations.

Benhamou et al. [Ref. 6] present a Prolog dialect called Newton, which is extended to handle systems of nonlinear equations and inequalities as well as constrained and unconstrained optimization. Newton implements a relaxed version of arc-consistency called box-consistency using the combination of interval arithmetic with a binary search technique. The goal of box-consistency is to determine an outer enclosing approximation of each variable's domain that is as tight as possible. Newton implements the arc consistency algorithm AC-3, where each relaxation step enforces box-consistency on the unary projections of the revised constraint. Enforcing box-consistency amounts to identifying the left- and right-most quasi-zeros (a zero that cannot be distinguished from a zero due to machine precision) on an interval function. The search for a quasi-zero uses the main step of Newton interval method. Since the Newton interval method is not sufficient to guarantee box-consistency, it applies an internal splitting operation used to locate the left-most and right-most quasi-zeros: each interval is split recursively into two other intervals of the same size. If no quasi-zero is found in the left part, this part is simply pruned and the algorithm is restarted on the right part.

Both 2-B and box-consistency assume the convexity of the variables' domains, and focus on optimizing the tightness of the feasibility space outer bounds. Their principal difference lies in the fact that the former requires interval tools for computing the unary projections of constraints, while the latter works directly on the original constraints and approximates the projections using interval Newton iterations. Finally, note that the Newton method generally produces sharper results than
2-B-consistency and is able to handle larger problems since no decomposition and no additional variables are required. However, it imposes more restrictive applicability conditions.

Some arc-consistency techniques use total constraints. Faltings [Ref. 58] shows that some undesirable features of propagation algorithms with interval labels must be attributed to the inadequacy of the propagation rule and to a lack of precision in the solution space description. Faltings demonstrates that by using a network of total constraints, along with a relaxation rule, based on the identification and classification of local extrema, sound and locally complete propagation becomes possible. Faltings arc-consistency algorithm is defined for binary constraints. Generalization to higher arity constraints has been implemented but requires more sophisticated mathematical preprocessing of constraints.

Since arc-consistency techniques enforce a weak degree of consistency, situations where no significant refinement of the variables' labels is obtained are unavoidable. Fixed-point algorithms are in effect sensitive to the variable dependency problem that produces a loss of precision in the refinement process. The variable dependency problem refers to the fact that when the same variable occurs in two distinct constraints, the refinement obtained through fixed-point iterations is generally less precise than if the two constraints were synthesized into a single constraint prior to relaxation. Both Benhamou et al. and Faltings' algorithms work on the original constraints of the problems rather than on primitive constraints. This explains why the approximations they provide are sharper than those delivered by the standard implementations presented above, such as CLP(BNR) [Ref. 54]. In the presence of cyclic constraint networks, fixed-point iterations cannot guarantee convergence. Lhomme [Ref. 40] addresses the problem by introducing the notion of 2-B(w)-consistency, a partial form of 2-B-consistency, characterizing the degree of consistency obtained when the refinement process terminates abnormally (looping problems). The parameter $w$ characterizes the imprecision of the computed bounds and 2-B-consistency is equivalent to 2-B(w)-consistency when $w=0$. Using
2-B(w)-consistency, the complexity of the consistency technique can also be tuned by fixing the precision desired for the resulting interval bounds. Faltings stresses the need for algorithms computing labels of higher degrees of consistency in order to avoid the problem of infinite iterations.

2.3.3.5 Algorithms for higher degrees of consistency in continuous domains

Path-consistency algorithms such as PC-1 and PC-2 have been applied successfully to some restricted classes of continuous constraints such as temporal constraints and spatial constraints. However, in the general case and in the absence of reliable tools for combining numerical constraints, it is difficult to reach even arc-consistency. As a result, only a few authors have investigated the issue of algorithms for higher degrees of consistency in continuous domains.

Hyvonen [Ref. 35] attempted first the notion of tolerance propagation (TP), which generalizes the idea of exact numerical propagation to interval propagation. In the TP approach, numerical problems are represented as spreadsheet tables where each variable is initially set to an arbitrary interval. A forward chaining rules schema propagating solution functions is then used to refine the intervals further by determining the least common general solution for the problem. Elementary solution functions are evaluated using interval arithmetic. A local TP algorithm, closely related to the Waltz filtering algorithm, can be generalized into global TP for determining globally consistent solutions. The idea behind global TP is to use global solution functions instead of the local ones during propagation. Global solution functions can be theoretically evaluated using algebraic manipulations or interval arithmetic. Hyvonen proposes the combination of numerical and algebraic techniques for computing the actual value of an interval function extension, and applies interval arithmetic algorithms for determining the most optimal extension. Since the evaluation of complicated interval functions is often computationally demanding, Hyvonen also proposes a partial globalization schema producing more optimal solutions than local tolerance propagation with less computation than global TP. The idea here is to determine global solution functions only with respect to some critical
variables and some sub-networks of the original constraint network. Lhomme [Ref. 58] generalizes the notion of 2-B-consistency to K-B-consistency. K-B-consistency algorithms are proposed that can reach higher degrees of consistency than 2-B-consistency but uniquely refine the outer bound of the feasibility domains.

2.3.3.6 Summary

Many practical problems need to reason over ranges of feasible values, and thus require tools for constructing the most exhaustive description possible of the feasible space. The discussion presented here has shown that although there are numerous methods devoted to numerical constraint checking, only a few fit these requirements. Interval arithmetic and constraint satisfaction techniques, often used in conjunction, are the most appropriate, yet important issues remain unexplored in interval-based consistency techniques, and further research is necessary to render them suitable for practical use.

Interval constraint solving techniques have been shown to be very efficient for solving problems involving nonlinear constraint systems of equations when the desired solutions are concentrated in a small region of solution space [Ref. 6]. However, many problems in the fields of simulation, control design, robotics, and camera control result in complicated design spaces, and consequently cannot benefit from present day interval constraint solvers. It is possible that interval constraint solving techniques may be extended in order to deal with these problems, and many have already been used to a limited extent. Non-standard interval arithmetic such as Kaucher and Markov [Ref. 58] and modal arithmetic appear to offer promising tools for dealing with both the computation of inner approximations and quantified variables. However, their use will require the implementation of efficient libraries and new interval algorithms.
3 SYSTEM DESIGN AND IMPLEMENTATION

3.1 TECHNIQUES, THEORY AND ALGORITHM DESCRIPTION

Most constraint reasoning developments have had difficulty achieving suitability to general applications. This work focuses on methods of solving quantitative engineering problems with uncertainties, and thus there is an emphasis on the practical application of numerical reasoning. Consequently, reasoning for binary and discrete variables is neglected in QES2 to streamline the handling of numerical constraints. In addition, the introduction of innovative high order consistency enhances the stability and efficiency for reasoning over numerical constraints.

3.1.1 Introduction

The focal idea of interval arithmetic constraints is to view computing problems as constraint systems that relate a set of variables or functions in real numbers. The variables whose values are to be computed are initially unbounded in the model. The goal of the computation, or reasoning, is to shrink the intervals of the variables in such a way that no solution to the original system is removed. The shrinking, or variable narrowing, is done iteratively by applying various contraction operators that typically apply only a subset of the entire constraint set. The implementation of interval constraints focuses on the development of contraction operators, which apply a narrowing set of constraints to the variables without removing any solutions, while also retaining as much efficiency as possible. Consistency techniques are used to select contraction operators in a sequence that may potentially accelerate the solution convergence.

Constraint interval arithmetic is a relatively new approach to the existing problem of deriving numerical results from algebraic models. Since it is simultaneously a numerical computation technique and a proof technique, it bypasses the traditional dichotomy between numerical calculations
and symbolic proofs. The combination of proof and calculation is used to handle practical problems, which neither method can handle alone. The underlying semantic model is based on the properties of monotone contraction operators on a lattice, an algebraic setting in which fixed point semantics take a particular sequential form.

A good numerical algorithm for constraint interval arithmetic should preserve the following properties [Ref. 54]:

- Narrowing (the answer is a substitution of the question).
- Idempotence (executing the answer adds nothing to the existing solution).
- Monotonicity (the more specific the question is, the more specific the answer will be).

Relational arithmetic in constraint logic programming languages typically applies to either finite or integer domains (CHIP), rationals (Prolog-III) or floating-point numbers (CLP(!)) [Ref. 6]. In these systems, the set of constraints is effectively restricted to linear equations or inequalities, for which there are well-known solution algorithms. Another approach suggested by Cleary [Ref. 12] and also proposed independently by Hyvonen [Ref. 35], considers relational arithmetic on continuous domains by adapting techniques from interval arithmetic. These ideas for constraint interval arithmetic were first fully developed and integrated into BNR Prolog in 1987 at Bell-Northern Research (BNR) [Ref. 54].

3.1.2 Interval Iteration

Basic fixed-point iteration is one of the oldest and most widely used general methods for numerically solving nonlinear equations. Figure 3-1(a) shows a simple example for finding a solution to the following pair of equations:

\[ y = f(x) \]
\[ y = g(x) \]

where \( f \) is decreasing and \( g \) increasing. Successive evaluations according to the following:
\[ y' = g(x) \]
\[ x' = f^{-1}(y') \]

are expected to converge to a unique solution of the two equations.

![Diagram](image)

(a) A successful iteration for solution  
(b) A failed iteration for solution

Figure 3-1 Conventional fixed-point iteration: iterate for solutions of \( y' = g(x) \) and \( x' = f^{-1}(y') \)

Unfortunately, unstable iterative behavior like that shown in Figure 3-1(b) is often encountered in practice. After appearing to converge initially, the iterations move away from the desired solution. Eventually, the iterations may find another unexpected solution, or may become periodic or chaotic. Convergence depends on many things: the shape of the curves, the starting point, which function is first applied, and the choice of coordinate system. Selecting all of these factors so as to guarantee convergence may require considerable analysis; in practical applications it may seem to be expedient to make random selections for these initial factors until convergence is achieved. However, the reliability of this approach is limited since the appearance of convergence is not necessarily true convergence, as illustrated by the previous example. There have been some cases of serious errors caused by accepting such approximate solutions [Ref. 66].
More sophisticated techniques, such as Newton's method, consist of a symbolic transformation of the problem, followed by iteration of the transformed problem. This transformation serves to accelerate the convergence, provided it occurs, but since the iteration technique is the same it still retains the difficulties of fixed-point iteration.

![Figure 3-2 Interval iteration proofing no solution in interval $X$ of $y' = g(x)$ and $x' = f^{-1}(y')$](image)

These difficulties vanish if the fixed-point iteration is performed on variables that have interval values instead of floating-point values. An interval iteration is shown in Figure 3-2. The procedure starts with an interval $X$ of values for $x$. It then finds the image interval $Y' = g(X)$ and its pre-image $X' = f^{-1} g(X)$, and then intersects it with the original interval. In the case shown in Figure 3-2 the intersection is empty, and it follows that there is no solution in the original interval $X$. It is important to see that this interval iteration is a proof that no solution exists in this interval, which is justified by the fact that the functions $f$ and $g$ are monotonically increasing or decreasing. For an initial interval that does contain a solution, Figure 3-3 shows the first two steps of a converging iteration in which the starting interval includes the solution and at each step the interval gets narrower. Ideally this iteration could be
continued indefinitely and converge to the solution. However, a few iterations often suffice to narrow the initial interval to a small final interval. An argument similar to Definition 2-2 shows that if there is a solution in the initial interval then there is a solution in the final interval. Of course, in this case there is an inclination to believe that there is a solution because of the completeness of the real numbers, but the constructive proof does not assume completeness and thus cannot draw such a definite conclusion. If this procedure is carried out using finite precision arithmetic, such as ordinary floating-point methods, it does not generally work since rounding errors will often cause the intervals to miss each other. In order to avoid this problem with finite precision it is necessary to make each interval slightly bigger by outward rounding after every arithmetic operation. Having done this, the iteration stops automatically when it reaches the limits imposed by the precision being used.

![Diagram showing interval iterations resulting in a solution on a narrowed interval of \( y' = g(x) \) and \( x' = f^{-1}(y') \)]

**Definition 3-1 Interval solution proof**

Assume \( x_L \leq x \leq x_H \) and \( x \) is a solution to

\[
x = f^{-1}[g(x)]
\]

then

\[
g(x_L) \leq g(x) \leq g(x_H)
\]

since \( g \) is monotonically increasing, and

\[
x_L' = f^{-1}g(x_L) \geq f^{-1}g(x) \geq f^{-1}g(x_H) = x_H'
\]
since $f$ is monotonically decreasing. Hence
\[ x \in [x_L, x_H] \cap [x'_L, x'_H] \]
but since the intersection is empty, the assumption that $x_L \leq x \leq x_H$ is a solution must be false.

\[ \]

Figure 3-4 Inconsistent initial intervals

Figure 3-4 shows a more complex case leading to a contradiction. In this case both $X$ and $Y$ have initial intervals. There are a number of different ways the iteration can be applied. One can start with $X$ and fold in the interval for $Y$ after the first half iteration, as indicated by the arrows in the figure, or start with $Y$ and fold in the interval for $X$, or do both in parallel and merge the results. Whichever way one chooses, the result (failure in this case) is the same. Even in survival cases, the final intervals will have the same bounds regardless of the sequence chosen. After studying some examples like this, one begins to suspect that unlike traditional fixed point iterations there are no pathological cases. The worst thing to happen is that the initial interval does not narrow at all, which simply indicates that the original hypothesis is too weak to reach any useful conclusion.
Constraint interval arithmetic obviously has similarities to the classical interval arithmetic developed by Moore [Ref. 50] in the 1960s. However, as a whole, they are extremely different paradigms. The first major difference is that constraint interval arithmetic is a relational language while the classical interval arithmetic is functional. This difference affects not only the formal structure of the language, but has a major impact on problem formulation. The second difference is the notion of a constraint as something which can be imposed once, but which continues to operate so as to maintain its truth. Classical interval arithmetic merely provides an abstract data type for intervals, and each individual calculation must be explicitly coded.

3.1.3 Computation as Proofs

The other aspect of constraint interval arithmetic that uniquely distinguishes it from traditional numerical techniques is that the computations represent proofs of the non-existence of solutions. This method is very different from traditional exact rational arithmetic, in which computations can be thought of as constructive proofs of existence. Constraint interval arithmetic, being a proving technique, carries a degree of logical force generally absent from traditional floating-point numerical computing, which is, by comparison, concerned only with heuristics. Interval proofs can be used to refer to real numbers since only bounded precision constants actually appear in the proofs, and the constraint system itself has no notion of real numbers in the full mathematical sense. In practical applications, this one-directional bounding technique is not contradictory to the consistency techniques described in the preceding chapter. The absolute consistency is a requirement in strict logical sense, but one-directional bounding is sufficient in most applications of engineering numerical calculations.

To take full advantage of the proving aspect of the technique, it is sometimes necessary to formulate problems negatively, so that a failure indicates a successful proof. Thus in the problem of formal system verification, the following question needs to be answered: If all components lie within their
specified tolerance intervals, can a system parameter lie outside its specification? A no answer would indicate that a successful proof of compliance has been constructed, achieving a formal verification for all systems characterized by the model and initial intervals. If no contradiction is found, then the final intervals indicate conditions in which the specifications might not be met, and thus provide a direct indication of where the design may be marginal.

3.1.4 Interval Constraint Reasoning, an Example

The following example illustrates the implementation of arithmetic constraint reasoning in QES2:

\[ x = \cos(x) \quad x = [-\infty, \infty] \]

In QES2, this equation is converted into two constraints:

Constraint 1: \( y = \cos(x) \)

Constraint 2: \( y = x \)

To carry out the narrowing process, the constraint must be converted into the following forms:

\[ y \leftrightarrow \cos(x) \]

\[ x = \cos^{-1}(y) \]

The implementation of the narrowing operators on such constraints is subtle, since \( \cos^{-1}(x) \) can be an infinite disjoint union of intervals, and the converted form must efficiently compute intervals that contain the mathematical intersections and are as small as possible. Subjecting the initial domain of \( x \) through the first constraint, the resulting domains are \( x = y = [-1, 1] \) due to the range of the cosine function. By applying constraint 1 again, the domain is further reduced to \( x = y = [-0.54, 1.0] \). Repeating the narrowing process a number of times, the solution of \( x = [0.739, 0.739] \) can be obtained.
If each primitive contraction has been proven to remove no solutions due to the original constraint, we know that the result is contained in the final interval. Convergence can be accelerated by using techniques such as Newton contraction of the following form:

\[ x = x \leftrightarrow \cos(x) \leftrightarrow a + \frac{\cos(a) - a}{1 + \sin(x)} \]

where \( a \) is chosen to be the midpoint of the interval \( x \). Many techniques such as this exist for further development.

3.1.5 Potential Pitfalls

For constraint networks that are acyclic it is possible to bound the number of operations required to achieve a fixed point. The bound is linearly proportional to the size of the network. Restoring a fixed point after a change requires much less work, and in particular cases may require only a few operations. There is a wider class of constraint networks that are essentially functional in that if the individual constraints were to be executed functionally in some particular order, they would produce the required fixed point.

It has been observed empirically that such networks seem to evaluate linearly, or near linearly, in time, but this probably depends on the specific details of the internal scheduling mechanism being used to guide propagation, and is not a general property of the technique. One example of this kind is linear systems that are both row and column permutations of a triangular system, with some interesting special cases in polynomial multiplication, reversion of series, and matrix factorization. A more subtle example is given for PERT/CPM scheduling problems [Ref. 43], where the algorithm seems to "find" its way to the classical canonical decomposition into separate upper and lower-bound potential construction problems regardless of the order in which the problem is stated. This ability to find and exploit hidden functional pathways, which seem to be quite common in industrial problems, may be one of the main explanations for the otherwise puzzling fact that this technique has worked
better than expected on realistic problems. It should also be noted that the existence of functional pathways and the direction of data flow depend on the relative sizes of intervals. Moreover, the optimal order of evaluation may be quite different depending on the initial conditions.

For general constraint networks containing loops, it is very difficult to predict performance theoretically even for specific problems, just as it is generally difficult to predict the number of iterations to convergence in conventional fixed-point iterations. It is also unclear how to formulate a useful complexity measure for constraint interval arithmetic in general. Since termination is governed by the actual precision being used, a complexity model must take the precision into account. Worst-case performance for fixed precision in general has an upper bound that depends exponentially on the number of variables but is independent of the problem being solved. This bound, which is on the order of the number of different states of the system, is so large as to be practically useless, and there are some relatively simple problems that come close to reaching it. For example:

\[ \text{abs}(X) \leq M, \text{abs}(Y) \leq M, \]
\[ A \times X + B \times Y = C \]
\[ A \times X + B \times Y = C' \]

where \( C \) and \( C' \) are disjoint intervals which differ by some small \( \delta \). This eventually results in failure, but effectively does so by counting from \(-M\) to \(+M\) by steps of size \( \delta \), which can be an enormously large number. On the other hand, in practice, industrial sized problems with hundreds of variables and constraints will in most cases converge to a fixed point in a reasonable amount of time. If the constraints are inconsistent, failure usually occurs fairly quickly unless the initial conditions are very near the boundary of the failure region.
3.1.6 Consistency Enforcement

Narrowing a system of variables through a set of constraints is the process of consistency enforcement. These techniques are key to all numerical constraint solvers, including QES2. They govern the efficiency, stability and generality of a constraint solver.

As introduced in Section 2.3, techniques for numerical consistency enforcement can be categorized into local, high order and global consistencies. All constraint reasoning techniques initiate a reasoning process from certain types of local consistency enforcement, with various implementations resulting for different capability and applicability. Research in the last decade shows that interval arithmetic is most applicable in real number reasoning [Ref. 60, Ref. 6], as adopted in the QES2 solver. Enforcing local consistency through interval arithmetic offers generality to numerical problems, but if used alone may be slow and plagued by complexity issues described in the previous section. The efficiency and stability may be potentially improved by applying high-order and global consistencies. However, most of the latest implementations of high-order consistency require significant increases in reasoning time that may offset the improvement in global reasoning stability. None of the current implementations of global consistency techniques are capable of handling general numerical problems [Ref. 61].

In the reasoning solver of QES2, local consistency is enforced by recursive iterations through a binary tree of converted constraints, similar to hull consistency in many other numerical constraint solvers. In contrast to other solvers, QES2 discards consistency manipulators in binary and discrete domains, consequently increasing the efficiency by preserving only interval arithmetic manipulators. To support global efficiency and avoid complexities, QES2 uses adaptive 2\textsuperscript{nd}-order consistency enforcement, as well as a direct partial enforcement on global consistency. Apart from being compatible with all reasoning scenarios targeted by other developments on consistency techniques,
the introduction of these two methods yields dramatic improvements on global efficiency and stability for most numerical engineering problems. Such improvements are the result of sacrificing absoluteness in strict logic applicability, while focusing on numerical solving by the same methodology as finite element structural analysis, as described in the next section.

![Diagram](image)

### Figure 3-5 Adaptive 2\textsuperscript{nd}-order and partial global consistency enforcement

With the QES2 solver, a reasoning process is always started by converting the constraint set into a binary tree, in which the enforcement of first-order consistency, or hull consistency, can proceed. A narrowing iteration consists of a series of recursive requests calling the root node, propagated down to the end nodes, and then backtracked with narrowing functions that finish at the root node. The
narrowing functions are first supported by interval arithmetic operations, composing the enforcement of 1st-order local consistency. The second step of a narrowing function is an adaptive 2nd-order local consistency check, which is realized by evaluating the level of further narrowing provided by the upper level consistency against a preset parameter \( nr \), defined as:

\[
nr = \frac{\text{Amount narrowed in the first constraint among all 2nd order constraints}}{\text{Amount narrowed in the 1st order constraint}}
\]

While a complete 2nd-order consistency requires all related constraints to be checked for narrowing on the upper level consistency, the adaptive 2nd-order consistency calls for full operations of 2nd-order enforcement only when the potential narrowing prospect is promising, i.e. when the limiting \( nr \) is reached. When the limiting parameter \( nr \) is properly set, the adaptive 2nd-order consistency can realize most of the global stability and efficiency achieved by a complete 2nd-order consistency, while avoiding the high associated computational cost. In QES2, \( nr \) is experimentally set to 10\%, yielding good performance for engineering calculations with hundreds of variables and constraints. Further research is needed to derive the optimum value of \( nr \). In Figure 3-5, an adaptive 2nd-order consistency enforcement is illustrated at the consistency between node \( a \) and \( b \).

With conventional interval constraint reasoning, the only means to accomplish global consistency is by enforcing local consistencies over all constraints. Since converged approximation of variables is well accepted in engineering calculations, QES2 implements the enforcement of global consistency at all instances on narrowing over the 1st-order consistency. Figure 3-5 illustrates such a process. While narrowing on the consistency between node \( c \) and \( d \), the current table of variables is substituted into the global constraint set for a consistency check. The normal narrowing process continues when the consistency exists. For the case that the global consistency is not achieved, bisection is called in the current narrowing step before proceeding to the next local consistency level. The outer bounds of the precision level used with the reasoning solver are the global consistency check to ensure logical soundness. In order to avoid potentially excessive computation associated with the iterative global-
local consistency interaction, the process of global consistency checking is a one-way process, i.e. a maximum of one iteration is allowed. While the enforcement over partial global consistency works on the assumption of approximation, it does eliminate the complexity issues of local consistencies acting alone.

3.1.7 Solution Space Plotting

The reasoning techniques presented in previous sections only give an outer approximation to the true solution of a design variable (Figure 2-2). For example, when the true solution of design variable $V$ exists in two disjoined regions as follows:

$$D_1 = [v_1, v_2], \text{ and } D_2 = [v_3, v_4]$$

an outer approximation will only result in a solution if:

$$D_0 = [v_1, v_4]$$

Research on continuous interval constraints has been attempting to achieve inner approximation, i.e., to identify the invalid region $D_i = [v_2, v_3]$. To date, constraint systems with universally quantified variables have mainly been handled by symbolic methods, such as the cylindrical trigonometric decomposition by Pau and Schicho [Ref. 56]. Some numerical methods do exist, such as using a bisection scheme [Ref. 61]. However, all these methods have strong requirements on the form of the constraints, and are limited to problems with few variables.

QES2 approaches the solution of inner approximation in a manner analogous to the finite element method used in structural analysis. A FEM analysis is a process of approximation, in which a solution is never completely accurate. However, higher accuracy can be achieved by using smaller element sizes. If a solution space is plotted with boxes that are small enough to reveal the invalid region $D_i$, an equivalent inner approximation is achieved. This thesis combines interval constraint reasoning with adaptive graphing to achieve this ability in solution space plotting.
Mathematical modeling in science and engineering often requires the following items:

- Solving a system of numerical equalities or inequalities (referred to collectively as constraints).
- Visualizing the solution to the system as a two-dimensional graph. The solution is in general a relation between the two variables selected as coordinates that may be functional or not.

Existing software concentrates on the case where the relation is a functional one specified by

\[ y = f(x) \]

The simplest approach is to sample \( f \) at equidistant values of \( x \) and to connect the resulting points by straight lines. This has the disadvantage that detail is lost where the function changes rapidly. Adaptive plotting attempts to allocate the available number of points in such a way that this problem is avoided as much as is possible with the given number of points. Even when adaptive sampling is used, significant features may fall between successive points and are missed in the plot. Independent to the use of adaptive sampling, two difficulties remain. Firstly, numerical rounding errors may accumulate unexpectedly when evaluating complex equations, and thus cause the plot to stray from the correct path. Secondly, implicitly defined curves, such as

\[ x^2 + y^2 = 1 \]

and

\[ \sin(x \cos y) = \cos(y \sin x) \]

need to be converted to the functional form

\[ y = f(x) \]

Without conversion to the functional form, one must resort to local methods such as described in Ref. 65, but generic success is not guaranteed.
The method of interval constraint plotting has the property of showing a hull of the function or relation. This property holds independently of rounding errors. In cases where the precision of the underlying floating-point system is insufficient, this limitation is shown as the hull being wider than necessary, but it remains a hull. Otherwise, the plotted hull is the smallest allowed by the resolution resulting from the selected size of pixel.

In this thesis the plotting scheme has adopted the method of interval constraint plotting. The following sections will review conventional plotting methods. Interval constraint plotting is contrasted against so-called raster-scan plotting. Raster-scan plotting is extremely simple to implement and can handle any relation, but also suffers from inefficient plotting, deformations, and is susceptible to rounding errors. Many of the problems faced by raster-scan plotting can be eliminated by using interval constraint plotting.

3.1.7.1 Conventional raster-based plotting

Conventional techniques for plotting a function \( y = f(x) \) usually utilize an algorithm similar to the following:

```plaintext
for (x = a; x != b; x = x + dx)
    plot(x, f(x));
```

In its naive form, \( \Delta x \) is a constant which determines the spacing that the function \( y = f(x) \) will be sampled at. This is not satisfactory for curves that oscillate much more strongly in some areas than in others. In adaptive plotting, \( \Delta x \) is a variable determined by the program to decrease the spacing between sampling points in regions of strongly oscillating curves.
Sampling is the major drawback in the conventional methods. Even adaptive sampling does not protect against an injudicious choice of points. The plotter may still choose an insufficient number of points to faithfully render part of a curve. This problem is illustrated in Figure 3-6. The left plot was produced with naive plotting, while the right plot employed adaptive plotting. Notice that the graph
becomes erratic when \( x \to 0 \), even when adaptive plotting is used, as the sampling density cannot adapt to the increasingly rapid oscillation of \( \sin(1/x) \). Figure 3-7 shows the hull of the graph plotted by interval constraint plotting at two resolutions, corresponding approximately to the effective resolutions used in the plots of Figure 3-7. The area of rapid oscillation near \( x = 0 \) is represented in these latter graphs as a rectangular block of size \([0, x_0] \times [-1, 1]\) which indicates that the function takes values in \([-1, 1]\) when \( x \in [0, x_0] \), but provides no other information.

Theoretically one can plot any equality relation \( f(x, y) = 0 \) by evaluating the relation at each point of a raster. The typical raster plotting algorithm is:

```plaintext
for (x = a; x != b; x = x + dx)
    for (y = c; y != d; y = y + dy)
        if (abs(f(x,y)) != eps) plot(x,y);
```

Inequality relations are handled in the same way. In fact, the condition in the `if` statement can be any boolean expression, a freedom that has been exploited to obtain plots of fractals, where the condition depends on the behavior of an iteration. Several plotting packages offer a plotting feature in which different shades of gray are used to represent the raster plots for different values of epsilon, such as the Mathematica `DensityPlot` function. Although there are several problems with raster-scan plotting, it has the important advantage of being able to handle definitions in functional as well as in relational form. Some of the problems are as follows:

- If the graph is a line, then the same `eps` will give a thicker line in some places than in others. It takes some adjustment of `eps` to get an acceptable plot. Conventional plotting already has the drawback that the choice of `dx` has such far-reaching consequences, and the arbitrary choice of `eps` introduces an additional drawback.
- It is slow due to the fact that it is quadratic in the number of points along a linear dimension.
- It does not adapt to the nature of the plot: featureless areas take as much time as those with detail.
3.1.7.2 Interval constraint plotting

By combining the idea of raster scan plots with interval constraints, the above three problems can be avoided. The idea is to replace the above code by the following:

```c
for (x = a; x != b; x = x + dx)
    for (y = c; y != d; y = y + dy) {
        solve the constraint system:
        f(u,v) = 0 && x <= u <= x+dx && y <= v <= y+dy;
        Paint the rectangle with sides [x,x+dx] and [y,y+dy] black or white according to success or failure.
    }
```

This has the following advantages. Firstly, the arbitrary choice of $\text{eps}$ is no longer needed. Secondly, a rectangle is plotted white only if the constraint solver can prove that it contains no solutions to the constraint. As a result, the areas plotted in black represent a hull of the curve. In this way no features are missed in the sense that the hull is shown. Of course, certain features are missed in the sense that no detail is shown within the elementary rectangles $[x, x + dx] \times [y, y + dy]$. The consistency method only removes inconsistent values and may not remove all such values, which could result in intervals containing no solution. Thus, results in the consistency method mean that if a solution exists, then it is in the intervals found.

Conventional plotting, because of the single `for` loop, is linear; however, it is restricted to functions. Raster scan plotting handles relations, but is quadratic. One might expect that quadraticity is inherent in the ability to handle relations. This is not the case with interval constraint plotting, which is able to dispatch large areas of the plotting space due to failure of a single interval constraint system. Thus, interval constraint plotting is adaptive in a similar sense to adaptive plotting of functions: featureless areas of the plotting space require less computing effort. An algorithm that exploits this adaptivity might look as follows:

```c
void function plot(Rectangle rect) {
    create and solve the interval constraint system
    r(x,y) && point (x,y) is in rect;
    if (failure)
```
Execution of this algorithm follows a quad tree, but only as far as necessary. Interval constraint systems corresponding to large areas can fail and can then be painted in one step. Some researchers suggest that the growth rate per resolution is linear [Ref. 69], which is a worthwhile improvement over the quadratic nature of the version directly derived from the raster scan method. There is also a heuristic argument for linearity in the interval constraint plotting method. One should note that the graph of a monotone function that passes through an $N \times N$ rectangular grid of cells can intersect at most $3N$ of the cells. Indeed, such a graph will cross at most $N$ horizontal grid lines and at most $N$ vertical grid lines before leaving the grid. Consequently, it can enter into the interior of at most $2N$ grid cells. If $A$ of these crossings are grid cell vertices, i.e., intersections of horizontal and vertical grid lines, then each such crossing accounts for an intersection with at most three new grid cells. Thus, the graph can intersect at most $3A + 2(N - A) \leq 3N$ grid cells. Assuming that the constraint solver will fail on all cells that do not intersect the graph, and that the relation being plotted eventually becomes locally monotone after a certain number of subdivisions, all further refinements of the graph will require linear execution time in the number of new subdivisions.

Because interval constraints only eliminate inconsistent values for the variables concerned, it will never miss any solution, assuming that the underlying interval constraint solving hardware/software system is correctly implemented. Hence, when the outcome includes an empty interval for a variable, there is no doubt that there is indeed no solution. In other words, there are no false negatives in interval constraint plotting. On the other hand, stabilization with all intervals nonempty only means that if there are solutions, they must be in the remaining intervals. In other words, false positives are
possible. The latter arises only in situations where rectangles are too large or where the precision of the underlying floating-point system is inadequate.

In interval constraint plotting the asymmetry between absence of false negatives and the possibility of false positives translates to the fact that a hull of the true plot is shown. False positives become less likely the finer the plotting area is subdivided. Therefore, it is encouraging that computation time appears to grow linearly with the number of subdivisions of the plotting axes. The asymmetry between the absence of false negatives and the possibility of false positives may appear to be a fundamental limitation. In practice, plotting this hardly appears to be the case. Suppose we are interested in the $z = 0$ contour of a continuous function $z = f(x, y)$. In that case we should invoke interval constraint plotting twice: once for $f(x, y) < 0$ and once for $f(x, y) > 0$. In each case the white areas, where interval constraints failed, guarantee the absence of solutions of the equality. Rounding errors have the effect of making these areas slightly too small. Thus, for contours, interval constraint plotting gives with certainty an area of negative values and an area of positive values. In between there is a narrow area of uncertainty. With the usual proviso, this narrow area is a perfectly precise rendering of the contour.
3.2 SYSTEM ARCHITECTURE

3.2.1 Introduction

QES2 (Qualitative Engineering System 2) is a software implementation of the theories and algorithms described in the preceding sections. QES2 enables an engineering design or analysis to account for uncertainties in the form of ranges of input design parameters, and illustrate analysis results in the form of plots of solution spaces. QES2 is based upon the techniques of numerical constraint reasoning, interval arithmetic and adaptive plotting.
Java, an Object Oriented Programming (OOP) language, was selected to develop QES2. The nature of OOP leads the development of QES2 as a component based design. As depicted in Figure 3-8, the core component of QES2 is a solver for reasoning with numeric constraints. The reasoning solver works with a collection of interval arithmetic libraries to handle various mathematical as well as logical functions such as \( \sin() \), \( \cos() \) and \( \text{iif()} \). The versatile input/output (IO) interface uses an input parser and tree-node representation of output information. In order to directly accept various engineering calculations in the forms of variables, equalities and inequalities, QES2 adopts an industry standard library named JavaCUP/JLex [Ref. 34] to parse user input into a tree-like form on which the reasoning engine operates directly. A parser table is set up to allow JavaCUP/JLex library to interact with QES2 specified syntax. The local consistency of the reasoning process is computed by interval arithmetic, as described in the preceding text. Various functions implemented in interval arithmetic compose the Interval Arithmetic Library of QES2. The IA library expands upon a set of routings called \text{ia_math} \) [Ref. 31] to include all functions commonly used in engineering calculations.

While the reasoning solver and its input parser and IA library provide QES2's ability to analyze problems, the post processing modules, including a design space plotter and a result set manager, allow users to effectively interpret the analysis results, visually and numerically. The design space plotter illustrates graphically the 2-D relationship between any selected sets of design parameters. The result set manager may be used to archive, retrieve and superimpose any sets of analysis results for various purposes such as result set comparisons. Engineering design plug-ins can be custom built to streamline input for specific engineering problems. Optionally, the post processing modules can be modified for various design plug-ins.

Java is chosen as the development platform of QES2, but other choices are available. Most of the systems, as well as legacy codes of constraint reasoning and interval arithmetic, are written in C++
due to its efficiency, robustness and wide availability across different hardware and operation
systems. While being able to deliver highly efficient and optimized products, C++ is often tedious to
apply in the cases of fast prototyping and concept proving. At the other end of the spectrum, there are
many platforms for fast prototyping or Rapid Application Development (RAD), such as Borland®
Delphi and Microsoft® Visual Basic. While being efficient in quick prototyping for simple problems
as well as user interface construction, RAD platforms lack the support of legacy and proven codes
available to C++ platforms.

For the development of QES2, Java offers the best aspects of both the C++ and RAD platforms.
Widely used in enterprise system development and academic research, there are many proven and
converted legacy code bases. Java as a programming language adopts most of the best features of
C++ while avoiding the high development cost in implementing primitive functionalities. Its ability in
developing complex systems is far superior to conventional RAD platforms. Being a relatively recent
and highly optimized language, Java is also capable of fast prototyping and RAD as well. Other
beneficial aspects of Java to the development of QES2 are its multi-platform capability and tight
integration with TCP/IP/Internet applications. The multi-platform support allows the Java-based
applications to be usable in multiple platforms such as Microsoft Windows, Apple Mac operating
systems, and UNIX systems. The support of Internet technology makes it possible to operate QES2
via the network. At the time of development, the only shortcoming of developing QES2 in Java is the
relatively slow runtime efficiency of Java, due to Java's dependency on runtime Virtual Machines to
interpret codes. However, it is expected that this shortcoming will be resolved with the development
of Java technology.

UML, the Unified Modeling Language, is a standard notation for modeling object-oriented systems.
UML, in its simplest form, is a language that graphically describes a set of elements. In its most
complicated from, it is used to specify, visualize, construct, and document not only software systems
but also business models and non-software systems. Much like a blueprint for constructing a building, UML provides a graphical representation of a system design that can be essential to assure architectural soundness of the system. UML diagrams are widely used for visualizing code and browsing classes and packages with Java programs. The key Java classes of QES2 are presented in UML in the following discussion.

3.2.2 Constraint Solver

The core of QES2 is its numeric constraint solver, consisting of Java classes UIterface, Exp and Interpreter, as shown in Figure 3-9. The UML diagrams of these classes are shown in Figure 3-11 to Figure 3-13.

UIterface is the interface between the user input and the analysis process, i.e. the constraint reasoning process, and also facilitates the data flow between all the reasoning components. All user input of variables and constraints are provided by user interface modules in the form of text strings. UIterface uses a parser to convert these input strings to Exp, a binary-tree expression, on which constraint reasoning can be employed. Upon requests from UIterface, Interpreter does the constraint reasoning, or interval narrowing, across all the variables/tree nodes of Exp. Interpreter initiates a recursive narrowing process on Exp, with the ia_math module handling all the interval
arithmetic calculations. The narrowing process continues until the convergence criteria are satisfied; then UInterface directs the narrowed EXP back to user interface modules, which will present EXP in the forms of numbers and solution space plots.

**Equation 3-1**

\[
\begin{align*}
  a + b &= 10 \\
  a &= [1,3]
\end{align*}
\]

![Binary tree representation of a sample constraint set Equation 3-1](image)

Equation 3-1 is used to illustrate a narrowing process by QES2. First QES2 breaks down the constraint set into a binary tree form as shown in Figure 3-10 (a). The relationships, such as (logical AND), +, -, etc. in the constraint set provide the linkages between two sub nodes. The sub nodes keep breaking down until each node is composed of a single variable. At the same time, a variable table is being assembled including all the variables in the constraint set (Figure 3-10 (b)). In QES2, every variable is defined as a real interval, representing the lower and upper bounds of the variable as a pair of real numbers.

A narrowing process takes the interval table from its initial state to the converged state, at which all variables can no longer be narrowed. Using the narrowing functions provided in the interval arithmetic library, the interval of each variable is being narrowed with interval arithmetic operations. Special techniques described in the preceding sections are incorporated in the narrowing process to
enforce the efficiency and stability. A narrowing pass is a recursive process starting at the root node and ending at the end nodes, with all variable intervals being updated in real time to the variable table. At the end of a narrowing pass, all variable intervals are narrowed at least once according to the constraints encountered during narrowing. In most cases, especially for complex multi-variable problems, a narrowing pass is not strong enough to stabilize the variable intervals, i.e. the intervals can be further narrowed. Thus, additional narrowing passes are requested and carried out repeatedly until a stage is reached at which all variable intervals can no longer be narrowed. This is the solution stage and all variable intervals represent the solution to the problem (Figure 3-10 (b)).
Figure 3-12 Constraint expression UML
3.2.3 Solution Space Plotter

The solution space plotter is handled by PlotAreaPanel, which is tightly integrated with the main solver control module UInterface, as shown in the UML diagram of PlotAreaPanel (Figure 3-16). The plotter adopts the graphing procedures described in section 3.1.7.2, plotting in 2-D space the relationship of any pair of variables in the constraint sets.

The premise to generate a solution space plot is the availability of a successful solution set produced by UInterface. When plotting the variable pair \( V_a \) and \( V_b \), their solutions must be available to fill up the solution space; otherwise, the solution space will be blank. As previously introduced, QES2 solves numerical equations and in-equations based on a proving process; a sub-region is valid only if its parent region is valid. Therefore, if a region of solution space is invalid in the constraint system, its sub-regions are invalid as well. The adaptive plotting method used by QES2 subdivides the solution space into quadrants. When the validity of the entire solution space of \( V_a \) and \( V_b \) is not available, the quadrant regions are invalid, resulting in a blank solution space plot.

Equation 3-2 is used to illustrate the process of solution space plotting. First the equations or constraints are solved by UInterface, resulting in a solution space spanning \( x: [-1,3] \), \( y: [0,9] \). As shown in Figure 3-14, PlotAreaPanel divides the solution space into quadrants once the validity of the entire solution space of \( x \) and \( y \) is known. It generates constraint pairs corresponding to each
quadrant. Each pair of constraint sets is added to the existing solution set and stored in `UIInterface`, which then initiates a narrowing process to prove the availability of solutions with the added constraints. If a quadrant is proven to contain no solution, it is plotted as blank; otherwise, the quadrant is further divided into smaller quadrants until the requested resolution is reached. The solution space in Figure 3-14 stops at the resolution of 8×8, and shows that only quadrants containing solutions are further subdivided. Figure 3-15 shows the actual solution space plots of QES2 with increasing resolution. Being able to produce solution space plots up to any resolution, QES2 conveys the graphical output to users with the finest possible detail since the maximum resolution of a graph is physically limited by the output media, such as monitor screens and paper prints.

Equation 3-2

\[ y = x^2, x = [-1,3] \]
Figure 3-15 Solution space plots of Equation 3-2 with increasing resolutions
Figure 3-16 Solution space plotter UML
3.2.4 Result Sets Manager

QES2 has the ability to store, retrieve and plot multiple result sets in the same 2-D solution space. This capability is useful for comparison of different result sets, such as the multi-code metal fatigue calculation described in the next section, serving the goal of quickly depicting complex relationships in engineering calculations.

In plotting solution spaces, the solutions are in the form of boxes, which are rectangular shaped regions with four vertices defining the boundaries of the valid results. For example, there are 15 such boxes in Figure 3-14. Many of these boxes compose a complete solution space plot, or solution file set when stored in a file. Multiple solution file sets are processed and plotted when a solution space comparison is requested. Boxes, result file sets, and multiple result file sets are represented in QES2 by the member classes Solution, SolutionFileSet and SolutionFileSets, respectively. Figure 3-19 to Figure 3-21 illustrate the UML diagrams of these classes showing their relationships and contents.

As shown in Figure 3-17, each solution, or box, is composed of a 2-D location given by a pair of integers \((i, j)\). The size and position of a solution box is decided by the classes PlotBoxSize, Range and SolutionFileSet. While plotting multiple solution sets, the compliance between multiple SolutionFileSet classes is computed for compatible PlotBoxSize and Range. One of the key
functions of the QES2 solution space plotter is to illustrate the intersection between multiple solution spaces. This is done by computing the interferences between SolutionFileSets. There are three states of interference: independent, interfered and intersected. The state of each solution box in a multiple solution plot is computed by the locations of the four vertices relative to other solution boxes, as shown in Figure 3-18.

![Figure 3-18 Interference of solution sets](image)

![Figure 3-19 Solution UML](image)
Figure 3-20 SolutionFileSet UML

Figure 3-21 Solution file sets UML
4 Fatigue Design and Engineering Applications with QES2

4.1 Engineering Applications Using QES2

The software framework QES2 integrates the techniques of constraint satisfaction processing, interval arithmetic and adaptive graphing, as described in the preceding chapters. The outcome of this integration is the ability of QES2 to handle engineering problems that have complex relationships and uncertainties, and also to disclose graphically the correlations among all design variables. QES2 calls for the following requirements when modeling an engineering problem:

- The problem must be described, explicitly or implicitly, by equations that define the relationships between the design variables. The components of the equation system, or the equation system as a whole, can be linear or nonlinear, determinate or indeterminate.
- All design variables can be described numerically, in the form of a single real number, or as an interval between two real numbers.

By solving the given problem with numerical constraint reasoning (a constructive proving process), QES2 produces sound solutions, where the highest achievable numerical accuracy is limited only by the hosting software and hardware platforms. Upon completion of a successful analysis, QES2 presents the results in the following forms:

- The design variable results are expressed as intervals between two real numbers. If an interval of a design variable is too narrow to be physically meaningful, the value of the variable may be considered to be a single value; otherwise, the variable is valid inside of the interval.
- A 2-D graph with specified resolutions can be plotted between two design variables. Provided that the variables are valid as intervals, instead of as a single real number, the plot will be an area, or solution space, instead of a line. The 2-D graph further improves the
representation of the output by including discontinuous solution spaces. The solution space plot is as sound as the numerical interval output, since it is produced by the same proving process.

- Multiple sets of solution spaces can be plotted on the same graph, showing the relationships between a pair of variables under different sets of constraints.

During the solving process, QES2 may fail to derive a solution due to numerical reasoning difficulties, but this is rarely encountered. In such cases, adding narrower constraints, which are usually readily available when modeling engineering problems, may help to eliminate the reasoning difficulties.

Most engineering analysis problems are described in the form of numerical variables and equations, and therefore can be solved using QES2 regardless of the complexities in the problem formulation. All engineering problems are physically meaningful and can provide additional constraints to overcome reasoning difficulties, if ever encountered. When numerical output of variables is not sufficiently intuitive for cognition, the plots of solution spaces may help to understand the problems on a conceptual level. The ability of the design space plot to disclose discontinuities in the solution space may help avoid potential design snares when dealing with complex design rules. Multiple solution space plots provide the means to directly compare results from different constraint sets, which makes this technique well suited to design scenarios where multiple design codes are needed for an additional margin of safety.

4.1.1 QES2 Procedures

An engineering problem is analyzed in QES2 using the following procedure:
1. Describe the problem with constraints given as formulas, equations and inequalities. There is no limit on the number of constraints. Refer to Chapter 6 for syntax on primary mathematic functions such as powers, exponentials and sine functions.

2. Provide as many additional constraints as possible to the key design variables. For example, if the fatigue life \( n_{Cycle} \) of a structural detail is to be calculated, a physically meaningful constraint such as \( n_{Cycle} = [0, 10^{16}] \) may be used. Such additional constraints may not be necessary for carrying out a successful analysis, but it will accelerate the computation process, and eliminate any reasoning difficulties if they exist.

3. Initiate the solving process. Additional solving iterations may be requested for additional numerical precision.

4. Select pairs of key variables from the design problem, and plot the solution space of the pairs. It is recommended to plot with low resolutions such as 32×32 to check the variable dependency first, then increase the plot resolution to the desired level. The solution space may be saved for future retrieval or comparison.

5. Steps 1-4 may be repeated with different sets of constraints on the key design variables. Then the solution spaces from different constraint sets may be plotted together to check intersections and interferences.
4.1.2 QES2 Techniques

![Sample solution space plots](image)

**Figure 4-1 Sample solution spaces, fully populated solution set and partially populated**

For most engineering calculations, applying QES2 and obtaining solutions is straightforward. The following is a list of potential difficulties, and techniques that can be used to overcome them:

- **Failure to narrow.** QES2 computes the final numerical results by narrowing all variables from their initial ranges, possibly infinity, to ranges that satisfy all the specified constraints. In some situations described in Section 3.1.5, some variables may fail to be narrowed. In such cases, the narrowing failure can be overcome by providing additional constraints to help initiate the narrowing process. Theoretically, this is not always possible; however, with most engineering problems additional constraints are usually available.

- **Availability of solution space plot.** When a set of solutions is derived by QES2, the solution space plotter will always generate a plot of solution areas. The matter of concern is thus not the availability, but the shape of solution space. A solution space plot will not be available only in situations where *infinity* is presented in a variable range. In such cases, adding a physically meaningful bound to the variable will enable the plotting of the solution space.
• **Variable dependency and interpretation with solution space plot.** The solution space plot between two variables, say $x$ and $y$, illustrates the relationship between them, while satisfying all of the direct and indirect constraints on all of the involved variables. If the solution space plot between $x$ and $y$ does not reveal any further details, then it is fully populated with solutions as shown in Figure 4-1 (a). With this type of solution space, the plotted pair of variables may be either independent of each other, or correlated with solutions fully populated through the intervals defined by the numerical results. Many solution space plots are partially populated as shown in Figure 4-1 (b), which reveals additional information on the relationship between $x$ and $y$ beyond the numerical results. With this type of solution space, only shaded regions satisfy all the constraints in the problem, and un-shaded regions represent design "potholes" (disallowed design regions) that are difficult to reveal with conventional analysis.

• **Fidelity of solution space plot.** In a solution space plot, shaded areas represent the locations where a solution exists. Since the plotting is a narrowing process, increasing the resolution will always result in a smaller shaded area, providing the shaded area can still be decreased. For engineering problems, it is often the un-shaded, or the invalid design areas that are of critical concern. In such cases it is therefore necessary to plot the solution space to a sufficient resolution that regions of disallowed design can be revealed to a satisfactory level of detail.

4.2 **General Engineering Applications of QES2**

Due to the highly flexible method of problem formulation, QES2 has many applications where complex calculations are involved. For example, engineering calculations often involve solving and understanding multi-variable equations. Linear equations can be easily solved and illustrated with many software programs. Complex nonlinear equations are difficult or impossible to solve, and even
more difficult to interpret. The following formulation is a three-variable equation set for solving the force/displacement relationship of a simple truss:

\textbf{Equation 4-1}
\[
\frac{29.5 \times 10^6}{600} \begin{bmatrix}
15 & 0 & 0 \\
0 & 22.68 & 5.76 \\
0 & 5.76 & 24.32
\end{bmatrix} \begin{bmatrix}
X_1 \\
X_2 \\
X_3
\end{bmatrix} = \begin{bmatrix}
20000 \\
0 \\
-25000
\end{bmatrix}, \text{ or } KX = P
\]

where:
- \( K \) = stiffness matrix
- \( X \) = displacement vector
- \( P \) = force vectors

The solution to this linear equation set is:
\[
\begin{bmatrix}
X_1 \\
X_2 \\
X_3
\end{bmatrix} = \begin{bmatrix}
27.12 \times 10^{-3} \\
5.65 \times 10^{-3} \\
-22.25 \times 10^{-3}
\end{bmatrix}
\]

When nonlinearity is introduced as in Equation 4-2, equation solving using QES2 can proceed as usual, while most other solvers will fail. QES2 allows interpretation of nonlinear solutions with parameter variations by plotting a solution space (Figure 4-2). The QES2 input and output for the examples in this section are shown in Chapter 9.

\textbf{Equation 4-2}
\[
29.5 \times 10^6 / 600 \times (A_{11}X_1 + A_{12} \sin(X_2) + A_{13}X_3) = P_1; \\
29.5 \times 10^6 / 600 \times (A_{21}X_1 + A_{22}X_2^2 + A_{23}X_3) = P_2; \\
29.5 \times 10^6 / 600 \times (A_{31}X_1 + A_{32}X_2^3 + A_{33}X_3) = P_3;
\]
Figure 4-2 QES2 solution space plot of a multi-variable, nonlinear equation set

Another example is a design problem constraining the three variables \( x, y \) and \( z \) as in Equation 4-3, with the following variable variations: \( x: [-2, 2] \), \( z: [-2, 2] \) and \( P: [-2, 3.5] \). Figure 4-3 depicts the relationship between variable \( x, y \) and \( z \), revealing potential potholes that are often neglected by other analytical methods.

Equation 4-3
\[
\begin{align*}
y^2 &= x^2 (x-1) + z^2 \\
x \cdot y \cdot z &= P
\end{align*}
\]
Figure 4-3 QES2 solution space plot of $x$ vs. $y$ for Equation 4-3

Figure 4-4 QES2 solution space plot of $y$ vs. $z$ for Equation 4-3
Fatigue design is a critical component of any structural design involving heavy cyclical loads or movable structural parts. Some examples of structures that experience cyclical loads are bridges, material handling structures, offshore platforms, entertainment ride systems, and astronomical telescope enclosures. The aim of fatigue design is to ensure that the structure has an adequate fatigue life. Calculated fatigue life also forms the basis for efficient inspection programs during fabrication and the operational life of the structure.

To ensure that the structure will fulfill its intended function, a fatigue assessment, supported where appropriate by a detailed fatigue analysis, must be carried out for each individual member that is subjected to fatigue loading. In structural engineering design and analysis, a fatigue assessment can be made by the following methods: S-N (stress-life cycle) analysis, and LEFM (linear elastic fracture mechanics) analysis.
4.3.1 S-N Curve Method

S-N analysis is the most commonly used method of fatigue assessment in engineering design codes. Using this method, the codes prescribe S-N curves to use for common structural details. Figure 4-6 illustrates schematic S-N curves for fatigue design, in which curves with lower stress ranges correspond to details more prone to fatigue failure. Different S-N curve designations account for potential fatigue prone defects, as well as various states of residual stress and stress concentration effects at structural details. Such effects often greatly amplify local stress levels, creating what are known as “hot spot stresses”.

![Typical S-N curves for fatigue design](image)

Figure 4-6 Typical S-N curves for fatigue design

Current design codes in structural engineering, such as CAN/CSA-S16.1 [Ref. 10], AISC [Ref. 3], Eurocode 3 [Ref. 21], base their fatigue assessments on the S-N curve method. This method is widely used for its simplicity in calculation, clarity in concept, and for the fact that it is concerned with fatigue life before crack initiation. Structural engineering has very high safety demands, and therefore the service life of a structure is required to be crack free. The S-N method is well suited to this requirement provided appropriate S-N data are available.
The S-N method is accurate when designing a particular detail subject to a specific loading where corresponding S-N test data is available. This data is obtained by subjecting many specimens of the same configuration to load testing. For common structural details, such as a steel bridge girder, S-N data are usually available, and if not it is usually worthwhile to obtain testing. However, many structures are unique, such as those used for material handling, entertainment ride systems, and astronomical telescopes. This brings up the major disadvantage of the S-N method in its applicability to structural engineering: S-N data are often not available for the specified structural detail and loading. Projects in structural engineering are often heavily constrained financially and many structural details and loading situations are often unique; therefore, obtaining S-N data through experimental means for every critical detail is likely infeasible. In such a scenario, engineers have to take available S-N data from commonly used details and apply this to more complicated details, trying to make such adaptations conservative. Due to the profoundly complex nature of fatigue failures, it is difficult to ensure conservativeness in such data adaptations. This ambiguity in S-N data selections is a weakness of fatigue design, and calls for methods to handle uncertainties in the design process.

QES2 offers the capability of simultaneously considering multiple S-N curves and other design parameters with uncertainties. With this capability, proper S-N data can be rationally selected. To further increase the level of conservatism in S-N based fatigue analysis, QES2 can also be used to analyze and visualize the outcome of calculations based on multiple codes.

Advanced techniques such as finite element method (FEM) and linear elastic fracture mechanics (LEFM) may help in designing fatigue resistant details, and are described in following sections. Such advanced techniques require additional resources; as a result, FEM and LEFM methods are not
commonly used in fatigue design of structures. The effectiveness of FEM and LEFM methods is also reduced by uncertainties in the input variables.

4.3.2 Using Finite Element Method with S-N Curves

Many structural details subjected to fatigue loading in complex structures do not have prescribed S-N curves. In such cases, an engineer has to use good judgment in order to make a conservative selection of an approximate S-N curve. However, a sound judgment call in this case requires abundant experience since the design outcome is very sensitive to the S-N curve selection. All structural design codes adopt the S-N curve method, but the concrete definitions of S-N curves and details do vary, in some cases on a very large scale. To avert such instability in S-N curve selection for uncommon details, some codes, such as NORSOK [Ref. 53], prescribe procedures to more accurately identify the hotspot stress by using FEM.

It is difficult to calculate the notch stress at a weld due to significant scatter in local weld geometry and different types of imperfections. This scatter is normally more efficiently accounted for by use of an appropriate S-N curve. It should also be noted that the weld toe region has to be modeled with an appropriate radius to obtain reliable results for the notch stress.

If a corner detail with zero radius is modeled, the calculated stress will approach infinity as the element size is decreased to zero. The modeling of an appropriate radius requires a very fine element mesh, which greatly increases the size of the model. Moreover, the selection of the radius size to be used for analysis is critical to the results. Therefore, a simplified procedure is recommended to reduce the demand for large, finely meshed models used for calculation of stress concentration factors:

- The effect of stress concentration or the notch factor due to the weld itself should be included in the S-N curve to be used.
The stress concentration due to the geometry effect of the actual structural detail can be calculated using a FEM model, resulting in a geometric stress concentration factor. The model should have a fine mesh for extrapolation of stresses back to the weld toe in order to ensure a sufficiently accurate calculation of the stress concentration factor.

Many publications refer to this simplified procedure as the “hot spot method”. The main emphasis of this method is to accurately model stresses in regions that are not affected by the weld. In using the hot spot method, some common practices of FEM modeling should be observed:

- In the location where hot spot stress is calculated, there should be a gradual change of element density, or mesh size.
- The size of the model should be large enough that the calculated results are not significantly affected by assumptions made for boundary conditions and application of loads.

Hot spot FEM has uncertainties as well, mainly due to the fact that FEM stress concentration models are generally very sensitive to element type and mesh size. By refining the element mesh, the FEM stresses at discontinuities will approach infinity. It is therefore necessary to set a lower bound for element size and use an extrapolation procedure to the hot spot to have a uniform basis for comparison of results from different computer programs and users. Some design codes and guidelines prescribe methods for setting a lower bound to element size [Ref. 53]. To minimize the uncertainties in the FEM hot spot method, it is recommended to perform a verification of the FEM procedure on a known detail for which an S-N curve is available. An alternative is to model the fatigue detail with a very fine element mesh including the weld notch to determine a target for the simplified analysis. Due to the nature of the hot spot method, it cannot be used for fatigue cracks starting from the weld root of fillet or partial penetrated welds, where crack-like geometries already exist. In this case LEFM procedures should be used.
4.3.3 Using Linear Elastic Fracture Mechanics

Linear Elastic Fracture Mechanics (LEFM) is recommended for use in assessment of acceptable defects, evaluation of acceptance criteria for fabrication, and planning of in-service inspection. Most structural engineering designs have a high safety demand and do not accommodate the existence of cracks, thus LEFM method is rarely used. Defects due to welding and crack-like geometries, such as a notch-shaped connection detail, sometimes exist in structures. LEFM method may be used if the latter case is of critical concern.

The purpose of LEFM analysis is to calculate the crack size that might occur during the service life, and determine if this will exceed the crack size corresponding to unstable fatigue fracture. A LEFM fatigue analysis uses Paris' equation to predict crack propagation or fatigue life:

\[ \frac{da}{dN} = C \cdot (\Delta K)^m \]

where:

\[ \Delta K = K_{\text{max}} - K_{\text{min}}, \text{ range of stress intensity factor} \]

\[ K \text{ may be calculated as } K = g \cdot \sigma \sqrt{\pi} \cdot a \text{ where:} \]

\[ \sigma = \text{ Nominal stress in the member normal to the crack} \]

\[ g = \text{ Geometric factor depending on detail and crack geometry} \]

\[ N = \text{ Number of cycles to failure} \]

\[ a = \text{ Size of crack} \]

\[ C, m = \text{ Material parameters, prescribed by many design codes and handbooks, e.g. Ref. 8.} \]

The calculations should be performed such that the structural reliability found using LEFM will not be less than that found using S-N data. This can be achieved by performing the following procedure [Ref. 8]:

88
1. The crack growth parameter \( C \) is determined by using the mean parameter value plus two times the standard deviation.

2. Initial defects in the structure are carefully evaluated, taking into account the actual non-destructive evaluation (NDE) inspection method used to detect cracks during fabrication.

3. Conservative geometric functions are used.

4. Utilization factors similar to those used with S-N method are used.

Since crack initiation is not included in the LEFM approach, shorter fatigue life is normally derived from LEFM than from S-N data. In the case that there is no S-N data available for comparison to the LEFM analyses, it may be beneficial to verify the assumptions used in LEFM analyses by performing an additional comparison for a detail where S-N data is available.

The initial crack size to be used in the calculation should be carefully considered in each case, taking into account typical imperfections, weld defects, geometries, access and reliability of the inspection method. For surface cracks starting from transitions between weld and base material, a crack depth of 0.5-mm may be assumed if other documented information about crack depth is unavailable. It is normally assumed that compressive stresses do not contribute to crack propagation. However, for welded connections containing residual stresses, the entire stress range must be applied to the connection, while only the stress components normal to the propagation plane need to be considered for the fatigue calculation.

4.4 STEEL FATIGUE DESIGN WITH QES2

Fatigue design of steel structures in engineering projects involves many structural details, often with unique geometries and loading. Engineering practice in such projects applies state-of-the-art design codes and engineering knowledge to achieve a design that is resistant to fatigue failure. Since the design codes are often not precise enough to cover the actual cases under design, the reliability of the
design outcome is the responsibility of the design engineer. Calculations for fatigue design are complex in nature and involve many linear and nonlinear formulas and equations, which make intuitive and conservative judgments difficult. QES2 can be used with multiple design codes to optimize the design process. The following text describes the fatigue design of a typical structural detail to illustrate such a process.

4.4.1 Structural Details under Fatigue Loading

Entertainment ride systems consist of steel structures that often have complex shapes and loading conditions. The structural components are subjected to fatigue loading from moving vehicles. The design and construction of such systems require expertise in both mechanical and structural engineering. While most mechanical components are off-the-shelf parts that usually have high reliability against fatigue failures, the structural steel components are custom designed and manufactured. Fatigue problems may occur if they are not well designed. Figure 4-7 illustrates such a structural detail under fatigue loading. The loading and geometrical configurations are simplified for clarity in the presentation of the methodology.
The structural detail consists of two parts: a welded wide flange (WWF) girder and a link arm fillet welded to the web of the girder. The web of the girder is locally reinforced with two vertical stiffeners to help counter the bending and tensile forces from the link arm. The fatigue loading described in the following text is a combination of axial and transverse forces acting on the free end of the link arm. Such a structural detail is by no means a good design detail by itself; however, the actual configuration of the detail may be the combined outcome of geometrical and functional requirements of the whole structural system. All relevant design data are shown on Figure 4-7.

4.4.2 Conventional Method of Fatigue Design

Two design codes are used in the following fatigue calculations: CAN/CSA-S16.1 [Ref. 10] and NORSOK [Ref. 53]. Both of these codes are based mainly on the S-N method, and are selected for illustration since they each represent a different application of the S-N method. The fatigue calculation prescribed by CAN/CSA-S16.1 is similar to most of the steel design codes in North America, such as AISC [Ref. 3] and AASHTO [Ref. 1] for which relatively few detail categories are defined and various complex effects, such as mean stress effect and thickness effect, are not covered. Such codes are relatively simple and easy to use for common cases such as details of steel bridges, but may become ambiguous for complex detail configurations. The NORSOK code is similar to other European codes such as Eurocode 3 [Ref. 21], DIN [Ref. 19], and FEM [Ref. 22], and prescribes the detail configurations into more categories than North American codes, and attempts to incorporate effects of mean stress and thickness. Contradictory to simple calculations such as sectional yielding, fatigue calculations using different codes often produce results that are quite different. For a particular application, it is difficult to identify which code is superior to others. All design codes are backed by sound experimental data, but potentially in different settings. By using multiple design codes for fatigue design, loss of reliability due to ambiguity in design variable determinations can be reduced.
Both CSA-S16.1 and NORSOK prescribe the fatigue life in following form:

**Equation 4-5**

\[
\log N = A - m \cdot \log(B\Delta\sigma)
\]

where:

- \( N \) = Fatigue life in number of cycles to failure
- \( A, B, m \) = Parameters pertaining to material, detail categories and other effects
- \( \Delta\sigma \) = Specified stress range

CSA-S16.1 and NORSOK divide structural details and loading situations into a number of categories and assign different sets of parameters \( A, B \) and \( m \). NORSOK has more detailed categories, and also accounts for other effects in calculations of the parameters. For example, when determining stress range \( \Delta\sigma \), NORSOK takes into account the effect of geometric stress concentration, or hot spot stress.

The detailed calculations are included in Appendix C, with the final results in Table 4-1.

**Table 4-1 Comparative fatigue calculation using CSA-S16.1 and NORSOK**

<table>
<thead>
<tr>
<th>Design Variables</th>
<th>CSA-S16.1</th>
<th>NORSOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detail category selection</td>
<td>El (Worst category) for HSS fillet welded to base plate. Bevel groove weld can be used at this detail for greater fatigue resistance, but not directly specified by S16.1. E is the worst category for groove weld by S16.1</td>
<td>W1 (Butt welded HSS sections with intermediate plate), or W2 accounts for potentially higher stress concentration from W1, or F3 (butt welded HSS sections). Use the worst case W2</td>
</tr>
<tr>
<td>A</td>
<td>( \log(y) = \log(1.28 \times 10^{11}) ) ( = 11.107 )</td>
<td>11.107</td>
</tr>
<tr>
<td>B</td>
<td>1.0</td>
<td>Stress concentration factor, 1.80 (F3)~2.25 (W2). Use 2.25</td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td>Thickness reduction factor 0.81</td>
</tr>
<tr>
<td>N</td>
<td>2,344,044</td>
<td>772,816</td>
</tr>
<tr>
<td>Fatigue utilization (actual load cycles/allowable load cycles)</td>
<td>0.969</td>
<td>5.967</td>
</tr>
</tbody>
</table>
### Design Variables

<table>
<thead>
<tr>
<th>Design Variables</th>
<th>CSA-S16.1</th>
<th>NORSOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Safe</td>
<td>Fails</td>
</tr>
</tbody>
</table>

Although the actual calculations involved are uncomplicated with both codes, selecting the appropriate categories may not be a trivial decision. No single category given in either code can precisely describe the detail given in the current example. The detail category is selected based on how closely it matches those in the codes in terms of weld type, loading and geometric configurations. As a result, the final fatigue utilizations calculated by each of the two codes are quite different. Uncertainties and ambiguities in the design code process may greatly affect the design outcome. For example, if category W1 is used instead of category W2 in NORSOK code, the final fatigue utilization changes from 5.967 to 2.90.

Using the worst-case category from CSA-S16.1, the calculations state that the detail has sufficient fatigue resistance. NORSOK describes a similar, but not identical case to the studied detail, and also accounts for additional relevant factors such as plate thickness. The calculations per NORSOK rule that the detail has insufficient fatigue resistance. Intuition may have initially concluded that CSA-S16.1 would result in a more conservative outcome since the worst case and bulk factors were used, but comparison with the NORSOK results shows the opposite. Similar situations are frequently encountered in fatigue design. By using conventional methods in fatigue design, it would be concluded that the studied detail might be insufficient in fatigue resistance due to the negative verdict given by the NORSOK code; at this point, no means are available to quantify the insufficiency.

#### 4.4.3 Hotspot Stress Calculation Using FEM

To obtain higher reliability in design outcomes, NORSOK prescribes FEM procedures to compute the hotspot stresses in the studied detail, while in the procedures of the preceding section, the hotspot
effect is included in the bulk factors used in the calculations. Such bulk factors may not sufficiently describe the studied problem.

Figure 4-8 FEM modeling for hotspot stress

Figure 4-9 FEM stress plot of link arm near weld (von Mises)
A FEM model was set up in ANSYS [Ref. 5] using 8-node elastic shell elements. The welds are not modeled since only the effect of geometric stress concentration is needed for the calculation. For the studied location, the hot spot stress is calculated from the Gaussian integration points as per NORSOK procedures, using an element size of 11mm near the welds. The resulting factor for hotspot stress is 8.20, compared to the bulk factor of 2.20 assumed in the preceding section. With the revised hotspot stress factor, the utilization factor from NORSOK calculations is now greater than 100, showing the unreliability in dealing with complex details in code based fatigue design.

It should be noted that FEM analysis for hotspot stress is usually not practical in most structural engineering projects due to resource constraints, even though it does increase the reliability of the calculation results.

4.4.4 Using QES2 in Fatigue Calculations

By using QES2 to carry out fatigue calculations, the following improvements can be made to help understand the studied detail:

1. Uncertainties of various parameters can be incorporated.

2. Solution spaces of fatigue utilization will be available for comparison of different design codes.

When using CSA-S16.1, the studied structural detail may be approximated as detail E1 or E, depending on the quality of the weld. This variation of detail categorization can be represented by using a corresponding range of fatigue life constants from $12.8 \times 10^{10}$ to $36.1 \times 10^{10}$, resulting in a fatigue utilization range of $[0.34, 0.97]$ calculated using QES2 (see section 8.2 for calculations). In the QES2 scripts, $\texttt{utilf}$ is the final design utilization, calculated by actual load cycles divided by
allowed load cycles. $F_{y\text{max}}$ is the maximum vertical force applied at the free end of the link arm. The variable $\log a$ considers the uncertainties in determining a proper detail category.

When using NORSOK, uncertainties exist with not only the detail classification, but also with the stress concentration factor. NORSOK’s definition of detail categories puts the studied case between F3 and W2, with the corresponding stress concentration factor varying from 1.80 to 2.25. Such variations result in a fatigue utilization range of $[1.11, 4.19]$. The variations of design variables and results are summarized in Table 4-2.

Table 4-2 Revised parameters in fatigue calculations considering uncertainties

<table>
<thead>
<tr>
<th>Design Variables</th>
<th>CSA-S16.1</th>
<th>NORSOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detail category selection</td>
<td>E–E1</td>
<td>F3–W2</td>
</tr>
<tr>
<td>A</td>
<td>$\log(\gamma) = \log(1.28 \times 10^{11}) = 11.107$ for E, and $\log(\gamma) = \log(3.61 \times 10^{11}) = 11.5575$ for E.</td>
<td>11.261–11.546</td>
</tr>
<tr>
<td>B</td>
<td>1.0</td>
<td>1.80–2.25</td>
</tr>
<tr>
<td>M</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>2,344,044–6,610,938</td>
<td>542,774–2,043,379</td>
</tr>
<tr>
<td>Fatigue utilization</td>
<td>0.34–0.97</td>
<td>1.11–4.19</td>
</tr>
<tr>
<td>Design</td>
<td>Safe*</td>
<td>Fails*</td>
</tr>
</tbody>
</table>

* A design is considered safe if actual number of cycles < allowed number of cycles by codes; otherwise fail.
Figure 4-10 Fatigue design using QES2, solution space plot NORSOK

(See Appendix C for variable definitions and calculations)
To help understand the effect of such variations, the solution space using the NORSOK method is plotted in Figure 4-10 with the primary force $F_y$ varying from 0 to 5 kN. The plot shows that the studied detail is by no means sufficient unless $F_y$ is approaching zero. Using QES2's Multi-plot, the solution spaces of CSA-S16.1 and NORSOK can be combined for comparison (Figure 4-11). The combined solution space shows that for the studied structural detail the outcome of the two codes intersect partially, while NORSOK is dominantly conservative with CSA-S16.1 effectively populating the lower (less conservative) bound of the solution space. Aided by calculations and plots from QES2, the following conclusions can be drawn:

(See Appendix C for variable definitions and calculations)

Figure 4-11 Fatigue design using QES2, multiple solution space plot per NORSOK and CSA-S16.1
1. The studied structural detail cannot be exactly described in the design codes. Using conservative design assumptions, the two codes result in contrasting utilization factors.

2. By incorporating uncertainties in the code calculations, the first conclusion still holds true with NORSOK giving the verdict of fatigue failure.

3. Multiple code design by plotting solution space of fatigue utilization reveals that the two design codes do correlate, insofar as the code parameters in CSA-S16.1 result in a lower bound to the NORSOK results.

4. Considering all the unknown and uncertain factors, the structural detail may be prone to fatigue failure during its service life. Proper design modification is recommended.

Although a design engineer may draw the same conclusion using plain calculations with the two codes, incorporations of uncertainties and plots of the solution space allow such a judgment to be made with additional soundness.
5 CONCLUSIONS

The research described in this thesis results in a computer software framework known as QES2. This work applies the latest developments in qualitative constraint reasoning to practical engineering problems. Using these techniques, QES2 is free of the constraints that limit conventional approaches, with the capability of solving complex engineering problems regardless of their formulations, explicitness or linearity.

Conceptual design is often considered the most challenging part of engineering. It involves simplifying a complex system of design processes, and at the same time making assumptions for design variables that introduce uncertainties. Traditionally, engineers largely rely on their knowledge and experience to rationalize these assumptions as much as possible. However, the quality of these assumptions degrades with increasing problem complexity. Probabilistic methods require the availability of a large amount of existing data, and therefore are not always useful. Expert systems can be used to sustain engineering knowledge and fuzzy logic systems can accommodate uncertainties. However, neither of them is capable of direct and reliable numerical analysis. Neural networks can deal with uncertainties and numerical analysis, but require sufficient training before effective employment. Therefore, existing engineering tools are not well suited to problems with design complexities and uncertainties, where abundant experience or design data are insufficient.

Fatigue design is an excellent example of an engineering problem with the aforementioned characteristics. The complex nature of metal fatigue results in uncertainties in the design process. Most existing fatigue design methods adopt simplified formulations, sometimes sacrificing accuracy and reliability of design outcomes. Advanced techniques such as finite element analysis and linear elastic fracture mechanics may increase the design reliability through comprehensive analyses of the
structural details under consideration; however, such methods are often impractical for structural engineers due to time and financial constraints. Furthermore, these types of analyses rely heavily on assumptions of residual stresses, loading variations and welding defects. With simplified formulations, the case study presented in this thesis shows that the design outcomes are sensitive to the assumptions of the parameter values used. A tool capable of rationally incorporating uncertainties and complexities, such as QES2, could increase the soundness of the fatigue design process. The techniques integrated in QES2 remove the traditional concerns associated with complex formulations, thus making it possible to simultaneously use multiple design codes and accommodate uncertainty in all design parameters. The capacity to depict complex relationships in solution spaces also increases the understanding of fatigue design problems. Although the relaxed requirements on problem formulation in QES2 accommodate calculations during all stages of design, QES2 is especially useful during the conceptual design stage due to its capability of delivering important design information that considers uncertainties.

Qualitative reasoning techniques are effective in solving problems that can be expressed with constraints. Constraints are able to account for uncertainty in engineering design by using numerical intervals for input parameters. Being a proving process, qualitative reasoning also possesses the following characteristics that are beneficial for engineering applications:

1. *Faithfulness to the mathematical model*. In conventional numerical computations, floating-point numbers in the computer substitute the true real numbers in a strictly mathematical sense. It is well documented that this can potentially result in large numerical errors. Qualitative reasoning regards numerical computations as computer-generated proofs that true real numbers are contained within a certain pair of computer floating-point numbers. This avoids any computational divergence caused by computer errors.
2. *Soundness of results.* The reasoning operations result in intervals that contain all values that are logically possible from arithmetic operations. Consequently, the existing computer hardware can in effect prove nonexistence of solutions.

3. *No restriction on forms of formulations.* Conventional numerical methods usually restrict the problem formulations to certain forms, such as linear equations or polynomials. Qualitative reasoning approaches problems by breaking down all formulations into simple logical relationships, thus in effect ignores the actual forms of problem formulations.

4. *Accommodation of uncertainties.* Qualitative analysis operates on pairs of real numbers representing intervals containing all possible values. The interval expression is used in input as well as output. Engineers are able to take into account uncertainties by specifying interval bounds for the design parameters.

In the past, research on qualitative reasoning has focused on binary and discrete problems, and algorithms have been optimized for generality of application and strict mathematical validity. Consequently, existing qualitative reasoning tools are inefficient at handling a large amount of numerical computation, which is essential for most engineering analysis.

To retrofit qualitative reasoning techniques for application to numerical engineering problems, this thesis approaches the reasoning techniques in a way that is analogous to the approximation of finite element modeling used in structural analysis. With proper modeling, the finite element method is able to approximate the true solution with increased accuracy as element size decreases. The reasoning process in QES2 generates solution spaces of design variables through adaptive plotting. This allows a logical, rather than analytical, treatment of continuous solution space, thereby avoiding the speed limitations of reasoning with singularities and other analytical anomalies. The accuracy of the result space plot is increased not through analytical means, but instead by increasing plot resolutions to the required level of fineness. The accuracy in the resulting solution space plot is
limited by the resolution of the presentation media, such as the computer monitor, and the accuracy is
guaranteed by the nature of logical reasoning. Neglecting binary and discrete problems further
reduces the reasoning overhead in QES2. For algorithm optimization, incorporating innovative
global and median consistency enforcements also increases the reasoning efficiency and reliability of
numerical variables. The resulting retrofit leads to an implementation of qualitative reasoning
techniques that is capable of performing everyday engineering design computations, while keeping
the desired characteristics of logical soundness and handling of uncertainties and complexities.

Further improvements on applying qualitative reasoning to engineering designs may concentrate on
two areas. Firstly, implementations of new techniques will potentially reduce the time complexity of
the reasoning algorithms. Great advances have been made in the field of numerical constraint
reasoning in recent years, and it is expected that more capable techniques will become available.
Secondly, relating variables with reliability functions and plotting probabilistic contours into solution
space can further improve representation of uncertainties and complexities, especially in the early
stages of engineering design. Through such integration, the fidelity of solution space plots is
improved with the increasing availability of detailed design data. This integration will require great
effort to upgrade the current reasoning algorithms, and will bring the methodology in considering
engineering uncertainties and complexity to a new level.
6 Appendix A – QES2 User Manual

6.1 Introduction

The reasoning engine of QES2 is capable of analyzing sets of numerical constraints with uncertain value definitions. The outcome of the analysis is presented as narrowed domains of input variables, as well as 2-D plots of related variables illustrating complex interfacing regions. This engine can be used to analyze engineering problems involving complex formulation and logic. Metal fatigue evaluation of complex structures with multiple design codes would be one such application.

6.2 Background

Many engineering design practices have not taken advantage of advanced theories and modern analysis procedures due to complexity of analysis as well as the large quantity of vaguely defined parameters used in actual designs. QES2 enables the use of complex analysis formulations to tackle practical problems with uncertainties, and present the design outcome in two-dimensional design space. Appropriate engineering assumptions and judgments in carrying out these procedures, often the most difficult part for practicing engineers, can be partially produced by using qualitative reasoning to define the trends and ranges, interval constraint analysis to derive the controlling parameters, and design space plotting to generate practical results. QES2 depicts a framework for bringing complex academic theories and analysis to practical engineering designs with help from information techniques.

Using the qualitative techniques implemented in QES2, all variables involved in an engineering design can be expressed in ranges (or intervals) in order to consider uncertainty, and any type of explicit or implicit design equations can be applied regardless of their complexity or whether that the equation system is complete or not. Instead of performing a typical multiple variation analysis to
describe “what-if” scenarios, the qualitative techniques produce results for all variables. This is presented in a design space, thus visualizing the engineering knowledge (and covering sets of conventional “what-if” scenarios with one analysis pass). Figure 6-1 depicts a typical information flow from design variables to solution space.

QES2 focuses on optimizing and integrating a group of qualitative techniques. The software consists of a number of components as shown in Figure 3-8. The center of QES2 is a numeric constraint solver, which carries out reasoning over input constraints. Design modules such as metal fatigue design represent the user interface for fatigue design. The design module may include multiple design codes so that the design outcome can cover multi-code acceptance, and also visualize the difference between codes, which is a current request of the industry.

At the start of each analysis run, all design equations and constraints are broken down into a tree-like structure. This is achieved by using a parser named JavaCUP and an appropriate parser table. The numeric constraint solver is built from a set of constraint-reasoning algorithms. Various modifications have been made to adapt and optimize the reasoning process for purely float-point numeric calculations. Commonly available conventional solvers target mostly binary and integer operations for their constraint reasoning. QES2 is able to reason through large numbers of engineering variables.
and constraints. During one analysis pass, the solver operates through the tree-like constraints, and keeps narrowing over the valid domain of each design variable until convergence is reached. The numeric capability of the solver is largely attributed to an arithmetic interval library. A group of functions for engineering calculations, such as step function and build-in fatigue functions are developed and used together with a generic interval arithmetic library named ia_math. The solution space plotting module uses adaptive plotting techniques to visualize the analysis result in solution spaces. The plotting resolution and analysis time are linearly proportional to each other.

The other aspect of qualitative reasoning, the one that makes it very different from traditional numerical techniques, is that its computations represent proofs, and these are always proofs of the nonexistence of solutions. As proofs, they carry a degree of logical force generally absent from traditional floating-point numerical computing, which is, by comparison, concerned only with heuristics. To take full advantage of this proof aspect of the technique, it is sometimes necessary to formulate problems negatively, so that a failure indicates a successful proof. Thus in QES2 system described here, one asks questions of the form: if components all lie within their specified tolerance intervals, can a system parameter lie outside its specification? A negative answer then indicates that a successful proof of compliance has been constructed, thus achieving a formal verification for all systems characterized by the model and initial intervals. If, however, a contradiction is not found, then the final intervals indicate conditions in which the specifications might not be met, and thus provide a direct indication of where the design may be marginal.

6.3 Syntax

The input to the engine consists of a set of constraints, and each constraint consists of a combination of expressions.

A constraint is given by one of the following relations, where A and B are expressions:
The syntax of an expression is described in Table 6-1.

### Table 6-1 Syntax of expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14159265</td>
<td>Real numbers</td>
</tr>
<tr>
<td>( \text{Infty}, -\text{infty} )</td>
<td>Infinity (IEEE 754)</td>
</tr>
<tr>
<td>Variable</td>
<td>A string of letters and/or digits, must start with a letter</td>
</tr>
<tr>
<td>([A, B])</td>
<td>The interval notation. Equivalent to a variable (X) with the additional constraints that (A \leq X \leq B).</td>
</tr>
<tr>
<td>+ - * /</td>
<td>Arithmetic operators, e.g. (A*B)</td>
</tr>
<tr>
<td>^</td>
<td>Integer power operator, e.g. (A^n) where (n) must be an integer</td>
</tr>
<tr>
<td>**</td>
<td>Real power operator, e.g. (A**B)</td>
</tr>
<tr>
<td>exp(A), log(A)</td>
<td>Exponential and the logarithm function</td>
</tr>
<tr>
<td>(\sin(A), \cos(A), \tan(A), \sinh(A), \cosh(A), \tanh(A))</td>
<td>Trigonometric functions and their inverse</td>
</tr>
<tr>
<td>MP(x), LP(x), RP(x)</td>
<td>Midpoint function MP(x) and left and right endpoint functions (LP(x), RP(x)).</td>
</tr>
<tr>
<td>(\text{iif}(E, A, B))</td>
<td>Conditional function. Returns (A) if (E) evaluated to TRUE, otherwise (B). Requires Boolean operation.</td>
</tr>
<tr>
<td>max(A, B), min(A, B)</td>
<td>Special conditional function. Returns the maximum/minimum value, requires Boolean operation</td>
</tr>
</tbody>
</table>

### 6.4 User Interface

A basic user interface is developed for illustrating the features of the reasoning engine, and provides a simple platform for engineering applications.
The user interface and the reasoning engine are built upon Java technology, and therefore can be run on any platform supporting Sun’s Java. To run the interface, Java Runtime Engine 1.3 or later must be installed. The latest Java Runtime Engine can be downloaded from:

http://java.sun.com/j2se/

Having Java Runtime Engine installed, the user can use the interface in two ways:

1. From a web page as a web Java applet. Using this method, copying and pasting to other Windows applications will be disabled due the Java security restriction.

2. From command prompt or shell. The user interface will appear after typing the following command:

   `Java -jar QES2.jar`

   This method allows copying and pasting to other Windows application, due to the fact that the Java application is running as a local application.

6.5 SAMPLE ANALYSES

6.5.1 Example 1

Example 1 describes solving a pair of linear equations with two variables. Screen shots of this example are shown in Figure 6-2. The parameters in the equations contain uncertainties. The equations to solve are:

\[
\begin{align*}
  a_{11} \cdot x_1 + a_{12} \cdot x_2 &= B_1 \\
  a_{21} \cdot x_1 + a_{22} \cdot x_2 &= B_2
\end{align*}
\]

The following constraints apply to the above equations:
To analyze the problem with QES2, use the following procedures:

1. Enter the equations along with constraints in Constraint box.
2. Click button Solve to solve; click button Solve/Solve More for better precision.
3. The numerical results containing valid variable domains are now shown in box Results.
4. Click tab Plot on the right hand side of the Window to show the plotting page.
5. Choose two variables to plot from the drop down boxes, specify the plotting resolution with the sliding bar, and then click button Plot/Re-Plot. The relationship between the two chosen variables is now shown graphically.

![Figure 6-2 Example analysis 1](image)

6.5.2 Example 2

Example 2 illustrates the plotting of an implicit relationship of the variables in equation (Figure 6-3):
\[ x^2 + y^2 = \sin(x^2 + a \cdot y), \text{ where } a \in [1.1, 2.2] \quad x < 3 \]
<table>
<thead>
<tr>
<th>Entities</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Classes</td>
<td>Classes whose attributes (fields and properties) and methods are inherited by another class. Also called superclass, parent class, or base class.</td>
<td><img src="image" alt="TextEditClass" /></td>
</tr>
<tr>
<td>Classes</td>
<td>Structures that define objects. A class definition defines fields and methods.</td>
<td><img src="image" alt="AWTEvent" /></td>
</tr>
<tr>
<td>Abstract Classes</td>
<td>Classes that are superclasses of another class but that can't be instantiated.</td>
<td><img src="image" alt="LayoutManager" /></td>
</tr>
<tr>
<td>Extending Classes</td>
<td>Classes that extend (inherit from) the superclass. Also called subclass or child class.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Implementing Classes</td>
<td>Classes that implement the central interface.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Extended Interfaces</td>
<td>Parent interfaces that are inherited by a subinterface.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Interfaces</td>
<td>Groups of constants and method declarations that define the form of a class but do not provide any implementation of the methods. Interfaces allow you to specify what a class must do but not how it gets done.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Implemented Interfaces</td>
<td>Interfaces that are implemented by the central class.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Dependencies/Reverse Dependencies</td>
<td>Using relationships in which a change to the used object may affect the using object.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Associations/Reverse Associations</td>
<td>Specialized dependencies where a reference to another class is stored.</td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Packages</td>
<td>Collections of related classes.</td>
<td><img src="image" alt="java.applet" /></td>
</tr>
</tbody>
</table>
8 APPENDIX C – SAMPLE FATIGUE CALCULATIONS

8.1 CONVENTIONAL METHODS USING PLAIN CALCULATIONS
FATIGUE ANALYSIS NORSOK, DETAIL D2-

TITLE

FATIGUE DESIGN PER NORSOK - LINK ARM BASE METAL

FORCES

Applied specified forces

Axial

Fxmax = 8 [kN]
Fxmin = 0 [kN]
Fymin = -5 [kN]
Fymax = 1 [kN]
Fzmax = -1 [kN]

Bending

Mymax = Fymax*lenA/1000 = 2.5 [kN-m]
Mymin = Fymin*lenA/1000 = -2.5 [kN-m]
Mzmax = Fzmax*lenA/1000 = 0.5 [kN-m]
Mzmin = Fzmin*lenA/1000 = -0.5 [kN-m]

number of cycles, (cycles/run)*(runs/hour)*(hours/day)*(days/year)*(years)

nCycle = 4*5*16*355*20 = 2,272,000

GEOMETRIES

Link arm

nameA = HSS127x127x11
lenA = 500 [mm]
areaA = 4840 [mm²]
lzA = 1.05E+07 [mm⁴]
dyA = 127 [mm]
dzA = 127 [mm]
twA = 11 [mm]

Cross beam

nameB = WWF400b200
lenB = 4000 [mm]
dyB = 200 [mm]
dzB = 400 [mm]
twB = 11 [mm]
tB = 20 [mm]

areaB = tB*(dyB+dzB-2*tB) = 11960 [mm²]

STRESSES

Axial

strXmax = Fxmax*1000/areaA = 1.7 [MPa]
strXmin = Fxmin*1000/areaA = 0.0 [MPa]

Bending

strByMax = Mymax*10*6/lyA*(dzA/2) = 15.1 [MPa]
strByMin = Mymin*10*6/lyA*(dzA/2) = -15.1 [MPa]
strBzMax = Mzmax*10*6/lzA*(dyA/2) = 3.0 [MPa]
strBzMin = Mzmin*10*6/lzA*(dyA/2) = -3.0 [MPa]

Normal stress at HSS section corner

strXcorMax = strXmax+strByMax+strBzMax = 19.8 [MPa]
strXcorMin = strXmin+strByMin+strBzMin = -18.1 [MPa]
strXcorRng = strXcorMax-strXcorMin = 37.9 [MPa]

CODE CHECK, FATIGUE RESISTANCE CSA S.16

Failure location A: link arm HSS section corner near weld toe

Code parameter, category F3-W3

code ref thickness tref = 25.00 [mm]
stress conc. factor SCF = 2.00
S-N inverse slope m = 3.00
logA value loga = 11.261
k = 0.25

Adjusted stress range adStrRng = strXcorRng*(twA/tref)*k*SCF = 61.80
nCode = 10*(loga-m*log(adStrRng)) = 772,816 [cycle]

fatigue util. utilf = nCycle/nCode = 2.940

ASSUMPTIONS

Crack first initiated near the weld toe in link arm section

Weld is sufficiently designed against maximum static and fatigue loads.
FATIGUE DESIGN PER CSA S16 - LINK ARM BASE METAL

FORCES

Applied specified forces

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axial</td>
<td>Fxmax</td>
<td>8[kN]</td>
</tr>
<tr>
<td></td>
<td>Fxmin</td>
<td>0[kN]</td>
</tr>
<tr>
<td></td>
<td>Fymax</td>
<td>5[kN]</td>
</tr>
<tr>
<td></td>
<td>Fymin</td>
<td>-5[kN]</td>
</tr>
<tr>
<td></td>
<td>Fzmax</td>
<td>1[kN]</td>
</tr>
<tr>
<td></td>
<td>Fzmin</td>
<td>-1[kN]</td>
</tr>
</tbody>
</table>

Bending

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mymax</td>
<td>2.5[kN-m]</td>
</tr>
<tr>
<td></td>
<td>Mymin</td>
<td>-2.5[kN-m]</td>
</tr>
<tr>
<td></td>
<td>Mzmax</td>
<td>0.5[kN-m]</td>
</tr>
<tr>
<td></td>
<td>Mzmin</td>
<td>-0.5[kN-m]</td>
</tr>
</tbody>
</table>

number of cycles: (cycles/run)x(runs/hour)x(hours/day)x(days/year)x(years)

nCycle = 4*5*16*355*20 = 2,272,000

GEOMETRIES

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nameA</td>
<td>HSS127x127x11</td>
</tr>
<tr>
<td></td>
<td>lenA</td>
<td>500[mm]</td>
</tr>
<tr>
<td></td>
<td>areaA</td>
<td>4840[mm^2]</td>
</tr>
<tr>
<td></td>
<td>IyA</td>
<td>1.05E+07[mm^4]</td>
</tr>
<tr>
<td></td>
<td>IzA</td>
<td>1.05E+07[mm^4]</td>
</tr>
<tr>
<td></td>
<td>dyA</td>
<td>127[mm]</td>
</tr>
<tr>
<td></td>
<td>dzA</td>
<td>127[mm]</td>
</tr>
</tbody>
</table>

Cross beam

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nameB</td>
<td>WWF400b200</td>
</tr>
<tr>
<td></td>
<td>lenB</td>
<td>4000[mm]</td>
</tr>
<tr>
<td></td>
<td>dyB</td>
<td>200[mm]</td>
</tr>
<tr>
<td></td>
<td>dzB</td>
<td>400[mm]</td>
</tr>
<tr>
<td></td>
<td>twB</td>
<td>11[mm]</td>
</tr>
<tr>
<td></td>
<td>tfB</td>
<td>20[mm]</td>
</tr>
</tbody>
</table>

areaB = tfB*dyB+dzB*twB = 11960[mm^2]

STRESSES

Axial

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>strXmax</td>
<td>1.7[MPa]</td>
</tr>
<tr>
<td></td>
<td>strXmin</td>
<td>0.0[MPa]</td>
</tr>
</tbody>
</table>

Bending

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>strByMax</td>
<td>15.1[MPa]</td>
</tr>
<tr>
<td></td>
<td>strByMin</td>
<td>-15.1[MPa]</td>
</tr>
<tr>
<td></td>
<td>strBzMax</td>
<td>3.0[MPa]</td>
</tr>
<tr>
<td></td>
<td>strBzMin</td>
<td>-3.0[MPa]</td>
</tr>
</tbody>
</table>

Normal stress at HSS section corner

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>strXcorMax</td>
<td>19.8[MPa]</td>
</tr>
<tr>
<td></td>
<td>strXcorMin</td>
<td>-18.1[MPa]</td>
</tr>
<tr>
<td></td>
<td>strXcorRng</td>
<td>37.9[MPa]</td>
</tr>
</tbody>
</table>

CODE CHECK, FATIGUE RESISTANCE CSA S.16

Failure location A: link arm HSS section corner near weld toe

Fatigue life constant, category E1

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gamma</td>
<td>1.28E+11</td>
</tr>
</tbody>
</table>

Code allowed cycles

nCode = gamma*(strXcorRng)^3 = 2,344,044 [cycle]

fatigue util. = nCycle/nCode = 0.969

ASSUMPTIONS

Crack first initiated near the weld toe in link arm section

Weld is sufficiently designed against maximum static and fatigue loads.
8.2 USING QES2 – INPUT AND OUTPUT

QES Input – NORSOK

/* NORSOK Detail D2 */

Fxmax = [8.8] ;
Fxmin = 0.00000 ;
Fymax = [0.5] ;
Fymin = [-5.0] ;
Fzmax = 1.00000 ;
Fzmin = -1.00000 ;

Mymax = Fymax*lenA/1000 ;
Mymin = Fymin*lenA/1000 ;
Mzmax = Fzmax*lenA/1000 ;
Mzmin = Fzmin*lenA/1000 ;

nCycle = 4*5*16*355*20 ;

lenA = 500.00000 ;
areaA = 4840.00000 ;
IyA = 10500000.00000 ;
IzA = 10500000.00000 ;
dyA = 127.00000 ;
dzA = 127.00000 ;
twA = 11 ;

strXmax = Fxmax*1000/areaA ;
strXmin = Fxmin*1000/areaA ;
strByMax = Mymax*10^6/IyA*(dzA/2) ;
strByMin = Mymin*10^6/IyA*(dzA/2) ;
strBzMax = Mzmax*10^6/IzA*(dyA/2) ;
strBzMin = Mzmin*10^6/IzA*(dyA/2) ;

strXcorMax = strXmax+strByMax+strBzMax ;
strXcorMin = strXmin+strByMin+strBzMin ;
strXcorRng = strXcorMax-strXcorMin ;

tref = 25 ;
SCF = [1.8,2.0] ;
m = 3 ;
k = 0.25 ;

adStrRng = strXcorRng*({twA/tref})**k)*SCF ;
nCode = 10**((loga-m*log(adStrRng)/log(10)) ;
utilf = nCycle/nCode ;

/* ------- final constraints --------- */
utilf < 10;

/* plots: utilf Fymax */
/* plot: utilf strXcorRng */
/* also adjust Fymax range and check influ */

QES2 Output – NORSOK

Fxmax = [8.0, 8.0]
Fxmin = [0.0, 0.0]
Fymax = [0.0, 5.0]
Fymin = [-5.0, 0.0]
Fzmax = [1.0, 1.0]
Fzmin = [-1.0, -1.0]
Mymax = [0.0, 2.5000000000000001]
lenA = [500.0, 500.0]
Mymin = [-2.500000000000001, 0.0]
Mzmax = [0.4999999999999999, 0.5000000000000002]
Mzmin = [-0.4999999999999999, -0.4999999999999999]
nCycle = [2271999.9999999986, 2272000.0000000014]
areaA = [4840.0, 4840.0]
IyA = [1.05E7, 1.05E7]
IzA = [1.05E7, 1.05E7]
dyA = [127.0, 127.0]
dzA = [127.0, 127.0]
twA = [11.0, 11.0]
strXmax = [1.6528925619834707, 1.6528925619834716]
strXmin = [0.0, 0.0]
strByMax = [0.0, 15.119047619047707]
strByMin = [-15.119047619047707, 0.0]
strBzMax = [3.0238095238095144, 3.023809523809542]
strBzMin = [-3.023809523809542, -3.0238095238095144]
strXcorMax = [4.676702085792984, 19.79574970484073]
strXcorMin = [-18.142857142857256, -3.0238095238095144]
strXcorRng = [7.700511609602497, 37.93860684769799]
tref = [25.0, 25.0]
SCF = [1.8, 2.0]
m = [3.0, 3.0]
loga = [11.107, 11.398]
k = [0.25, 0.25]
adStrRng = [11.288994311058705, 61.79801761325429]
nCode = [542096.2070237357, 1.7379378333065295588]
utilf = [0.013072964733109862, 4.191137976179424]
QES2 Input – CSA-S16.1

/* CSA-S16 Detail D2 */

Fxmax = [8,8] ;
Fxmin = 0.00000 ;
Fymin = [-5,5] ;
Fymin = 1.00000 ;
Fzmin = -1.00000 ;

Mymax = Fymax*lenA/1000 ;
Mymin = Fymin*lenA/1000 ;
Mzmax = Fzmax*lenA/1000 ;
Mzmin = Fzmin*lenA/1000 ;

nCycle = 4*5*16*355*20 ;

lenA = 500.00000 ;
areaA = 4840.00000 ;
IyA = 10500000.00000 ;
IzA = 10500000.00000 ;
dyA = 127.00000 ;
dzA = 127.00000 ;

strXmax = Fxmax*1000/areaA ;
strXmin = Fxmin*1000/areaA ;
strByMax = Mymax*10^6/IyA*(dzA/2) ;
strByMin = Mymin*10^6/IyA*(dzA/2) ;
strBzMax = Mzmax*10^6/IzA*(dyA/2) ;
strBzMin = Mzmin*10^6/IzA*(dyA/2) ;

strXcorMax = strXmax+strByMax+strBzMax ;
strXcorMin = strXmin+strByMin+strBzMin ;
strXcorRng = strXcorMax-strXcorMin ;

gamma > (12.8)*10^10; gamma < (36.1)*10^10;
nCode = gamma/(strXcorRng)^3 ;
utilf = nCycle/nCode ;

/*/ ------ final constraints -------- */
utilf < 10;

/* plots: utilf Fymax */
/* plot: utilf strXcorRng */
/* also adjust Fymax range and check influ */
QES2 output – CSA-S16.1

Fxmax = [8.0, 8.0]
Fxmin = [0.0, 0.0]
Fymax = [5.0, 5.0]
Fymin = [-5.0, 5.0]
Fzmax = [1.0, 1.0]
Fzmin = [-1.0, -1.0]
Mymax = [2.499999999999999, 2.500000000000001]
Mxmin = [-2.500000000000001, 2.500000000000001]
Mzmax = [0.4999999999999999, 0.5000000000000002]
Mzmin = [-0.5000000000000002, -0.4999999999999999]
nCycle = [2271999.9999999986, 2272000.0000000014]
areaA = [4840.0, 4840.0]
IyA = [1.05E7, 1.05E7]
IzA = [1.05E7, 1.05E7]
dyA = [127.0, 127.0]
dzA = [127.0, 127.0]
strXmax = [1.6528925619834707, 1.6528925619834716]
strXmin = [0.0, 0.0]
strByMax = [15.119047619047707, 15.119047619047707]
strByMin = [-15.119047619047707, 15.119047619047707]
strBzMax = [3.0238095238095144, 3.0238095238095144]
strBzMin = [-3.0238095238095144, -3.0238095238095144]
strXcorMax = [19.795749704840055, 19.795749704840073]
strXcorMin = [-18.142857142857256, 12.095238095238196]
strXcorRng = [37.938606847697999, 37.938606847697999]
gamma = [1.279999999999999E11, 3.6100000000000415E11]
nCode = [2344044.394217609, 7.905848032950819E8]
utilf = [0.0028738219992725886, 0.9692649190453348]
9 Appendix D – Sample QES2 Scripts for Solving Equation Sets

QES2 Input:

\[
29.5 \times 10^6 / 600 \times (A11 \times X1 + A12 \times \sin(X2) + A13 \times X3) = P1;
29.5 \times 10^6 / 600 \times (A21 \times X1 + A22 \times (X2^2) + A23 \times X3) = P2;
29.5 \times 10^6 / 600 \times (A31 \times X1 + A32 \times X2^3 + A33 \times X3) = P3;
\]

\[
P1 = [20000, 22000];
P2 = [-100, 1500];
P3 = -25000;
\]

\[
A11 = [15.000, 18];
A12 = 0.000;
A13 = 0.000;
A21 = 0.000;
A22 = [22.680, 35];
A23 = [5.760, 6];
A31 = 0.000;
A32 = [5.760, 7];
A33 = [24.320, 27];
X2 = [-10, 1];
\]

QES2 Output:

\[
A11 = [15.0, 18.0]
X1 = [0.022598870056497043, 0.029830508474576363]
A12 = [0.0, 0.0]
X2 = [-0.08318735041578275, 0.08318735041578275]
A13 = [0.0, 0.0]
X3 = [-0.021073365554857396, -0.018683144527712577]
P1 = [20000.0, 22000.0]
A21 = [0.0, 0.0]
A22 = [22.68, 35.0]
A23 = [5.76, 6.0]
P2 = [-100.0, 1500.0]
A31 = [0.0, 0.0]
A32 = [5.76, 7.0]
A33 = [24.32, 27.0]
P3 = [-25000.0, -25000.0]
\]
10 Abbreviation

CLP p.18
CSP p.7
FEM p.86
QR p.4
RAD p.62
UML p.62
11 REFERENCES


Ref. 2 G. Alefeld. On the convergence of some interval arithmetic modifications of Newton's methods. In Journal of numerical analysis, 21(2) SIAM 1984


Ref. 5 ANSYS Inc. ANSYS User Manuals, Ver. 7.0. ANSYS Inc. 2002


Ref. 15 G.B. Danzig. Linear Programming and Extensions. Princeton University Press 1965


Ref. 19 DIN. *DIN 15018: Cranes*. Deutsches Institut für Normung (German Institute for Standardization) Nov. 1984


Ref. 21 European Committee for Standardization (CEN). Part 1.9 Fatigue strength of steel structures (pre-standard), *Eurocode 3: Design of Steel Structures*. European Committee for Standardization (Brussels) 2001

Ref. 22 F.E.M. *Rules for the design of Hoisting Appliances 3rd ed.*, Federation Europennne De La Manutention, Section I, 1993


Ref. 33 H. Hong. RISC-CLP(Real): logic programming with nonlinear constraints over reals. In Constraint logic programming – selected research. The MIT Press 1993


Ref. 35 E. Hyvonen. Constraint reasoning based on interval arithmetic. In Proceedings of the 11th International Joint Conf. on AI, p 1193-1198, Detroit, USA. 1989


Ref. 43 D.G. Luenberger. *Linear and nonlinear programming*. Addison-Wesley 1984


Ref. 53 NORSOK. NORSOK Standard N-004: Design of Steel Structures, Rev.1. NORSOK Standards Organization (Oslo) 1998


