Computer Methods for Design Automation

by

Christian Bliek

Burgerlijk Elektro-Werktuigbouwkundig Ingenieur
Katholieke Universiteit Leuven
Belgium, June 1986

Submitted to the Department of Ocean Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1992

Author .... [illegible]
Department of Ocean Engineering
July 6, 1992

Certified by .... [illegible]
Professor Chryssostomos Chryssostomidis
Thesis Supervisor

Accepted by ....
~~A. Douglas~~ Carmichael
Chairman, Departmental Committee on Graduate Students
Department of Ocean Engineering

Computer Methods for Design Automation
by
Christian Bliek

Submitted to the Department of Ocean Engineering
on July 6, 1992, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Design automation requires reliable methods for solving the equations describing the performance of the engineering system. While progress has been made to provide good algorithms for polynomial systems, we propose new techniques for the solution of general nonlinear algebraic systems. Moreover, the interval arithmetic approach we have chosen also guarantees numerical reliability.

In this thesis we present a number of new algorithms that improve both the quality and the efficiency of existing interval arithmetic techniques for enclosing the solution of nonlinear algebraic equations. More specifically, we propose an exact algorithm for the solution of midpoint preconditioned linear interval equations. We extend existing techniques for automatic compilation of fast partial derivatives to include interval slopes and have conceived a number of graph algorithms to improve their efficiency for general computational graphs. Furthermore, we have devised variable precision techniques to automatically control the required precision based on interval width. Finally, we have unified a number of enclosure languages using denotational semantics.

Since design computations can be performed with conservatively bounded models instead of point models, this approach also allows us to develop a framework in which the hierarchical design process can take place and to address the consistency problem associated with incremental refinements. In addition, conservative enclosures are particularly useful when computations are made in a distributed fashion, at different speeds and when communication delays are unpredictable.

We believe that since numerical set computations for engineering design fills a void between traditional numerical analysis and discrete mathematics, it promises to be an exciting new area of research.

Thesis Supervisor: Chryssostomos Chryssostomidis
Title: Professor of Ocean Engineering

à Doudou

Acknowledgments

Many people contributed to the work presented in this thesis.

My advisor, Professor Chryssostomidis gave me the opportunity to come to MIT and supported me throughout the years. He allowed me to explore new areas of research and has helped to broaden my background considerably.

Members of thesis committee, Nicholas Patrikalakis and Warren Seering gave valuable feedback and shared their enthusiasm.

Mike Drooker, my office mates and fellow students have provided for years of stimulating discussions, encouragement and friendship.

I thank my parents, who provided me with a stimulating environment to grow and learn, and my brother for showing me the way. Thanks to my friends, near and far, for standing by me.

Most of all, I would like to thank my wife Françoise for her patience and endurance, she made it all worthwhile.

This work was supported, in part, by the Office of Naval Research under the contracts N00014-87-K-0462, N00014-89-J-1187 and N00014-91-J-1014.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.1.1 Design Automation

The design of large-scale complex systems usually requires the participation of a large number of people. It typically proceeds as follows. A handful of people, e.g. the project manager, chief engineer and a few experts lay down initial concepts and dimensions. They tend to rely on back of the envelope type calculations or sometimes on personal computers and spread sheets for working through this very first design stage. This initial design solution is refined by the design engineers. Drafting people finally detail and enter the entire model into the computer. Once the necessary detail is available, some preliminary analysis is done. It is only very late in the design cycle that CAD software is used and extensive analysis, such as finite element analysis, is used to check the performance of the design.

1.1.2 Shortcomings

When designing complex systems, usually about 70% of the cost is committed during the early design stages. This means that the most important decisions are made at the very beginning of the design cycle, when there is very little precise information concerning the unfolding of the remaining part of the design process. The people which have to make these decisions essentially rely on experience and intuition. Once these important decisions are made, it becomes very costly to reevaluate them, since substantial changes in the overall system would have to be made. To avoid major redesign efforts, corrections and fixes are made all along the design process, resulting in workable but usually suboptimal solutions. Ironically most of the computer resources are used during the later design stages. They are essentially used to create CAD models of the designed structure and to perform complicated numerical analysis.

1.2 Thesis

Although some aspects of the design process have been automated in an incremental fashion, we believe that further progress in this area requires us to rethink some of the basic premises.

To take advantage of the flexibility and the speed of available computer power requires integration of the entire design process in the machine. On the one hand, this can provide more and better information to support decision making so that it can be more effective. On the other hand when unanticipated problems or costs surface at a later design stage substantial changes can be made without incurring the large extra costs.

We believe that the key issues in conceiving such a system are reliability and robustness. Indeed, while designers often rely on numerical computations, they will always check the results to ensure their validity. Furthermore, current implementations do not allow for late coming information and their performance degrades drastically in the face of inconsistencies. In this thesis we therefore study fully reliable computational methods that allow for late coming information and for the detection of inconsistencies. Finally, it should be noted that while the current technology allows for very fast computations, the algorithmic time complexity remains an important issue.

We are convinced that this type of system will not only reduce the number of people involved in the design cycle, but that it will speed up the design process and improve the overall quality of the designed product.

1.3 Overview

In chapter 2 we discuss the problem of design in more depth and propose a general framework for design automation. Some basic aspects of interval arithmetic, which allows for full reliability of numerical algorithms, are reviewed in chapter 3. In this chapter we also describe a novel approach for specification of the denotational semantics of enclosure programming languages. In chapter 4 we describe a new algorithm for the solution of linear interval systems. This algorithm is a significant improvement of traditional linear interval solvers. A number of new results relating to computational graphs are proposed in chapter 5. In particular, we discuss the problem of optimal computation of partial derivatives and interval slopes, and propose a method for computing upperbounds on the dimensionality of the solution of computational graphs. The following chapter, chapter 6, contains a number of new methods that allow for automatic control of the required precision of interval computations, including the use of variable precision. In chapter 7 we put these techniques in perspective and discuss their importance to the design problem. This includes the computation of solution sets, the hierarchical aspect of design and its relation to the consistency problem. We conclude this chapter with the presentation of an approach for reliable and robust concurrent computations. Finally, in chapter 8, we sum up and discuss a number of further research topics in this new area of research.

For the reader interested only in the contributions in the field of interval arithmetic, we recommend to quickly go over chapter 3 and read chapters 4, 5 and 6. If one wants to focus only on the design aspects, we would suggest to read chapters 2 and 7.

Chapter 2

Design

2.1 Introduction

We believe that the general problem of design is of a combinatorial nature. As a result, except for a few of special cases, one can only approach this problem in a heuristic fashion. It is probably for this reason that design is often not considered an exact science.

Combinatorial explosion is typically avoided by incrementally adding problem specifications as the design process proceeds. It is therefore important to carefully distinguish the givens from the unknowns. In our view, problem specification can only be done by the designer. In addition, we will assume that models associated with the design space are given. The task of capturing requirements and modeling information can however considerably be facilitated by high level tools. Furthermore, we believe that the selection and solution of the appropriate models can to a large extent be automated. Our ultimate goal can therefore only be to devise tools that allow us to represent the available information effectively and to automatically select the set of design models that satisfy the imposed requirements, not to replace the designer.

We believe that further automation of the design process requires a careful study of the interaction of the representation problem and the tractability of automatic model selection methods. In this thesis we study a number of key issues related to automatic solution techniques for design.

In section 2.2 we define our approach of the design problem more precisely. In section 2.3 we propose a framework for design and identify the issues that need to be addressed. We then contrast our approach with existing ones and review some of the previous work in the area of design automation in section 2.4. Finally, in section 2.5 we use the Design Society metaphor to put this framework into perspective.

2.2 Towards a Theory of Design

In this section we present our interpretation of design. It also provides an abstract setting for the framework proposed in the next section.

2.2.1 Functional Requirements and Hierarchy

Design can be seen as the process of mapping *functional requirements* into a description of a realization of those requirements. By definition, functional requirements are functions or properties that the design needs to provide. This mapping process generally involves search to find the particular realizations that provide the required functions. A realization is unambiguously described by *design variables.*

The design process is hierarchical. First, a general solution is found by imposing the most important functional requirements. Then, more refined solutions can be obtained by imposing additional requirements. This mapping process from functional requirements to design variables continues level by level until sufficient detail is reached. The resulting model relates the functional requirements to the realization of those requirements and is called a *design model.* Although one ofter refers to this hierarchical approach as a design principle, it really only is a heuristic to control the complexity of the search space. Indeed, the designer attempts to focus the search of the solution space by identifying the dominant characteristics of the design model first.

Both design variables and functional requirements contain information concerning the design model. The fundamental difference between them is in the way they are used. Functional requirements correspond to those pieces of information that describe the problem specification, while the design variables are chosen in order to satisfy them. In a different design situation some of the functional requirements can become design variables while some design variables can be qualified as functional requirements. Note also that by definition the designer cannot require that design variables behave in a certain way, otherwise he would need to express this as additional functional requirements.

Our definition of design doesn't necessarily exclude manufacturing. Indeed, some of the design variables could in fact be associated with manufacturing processes and the functional requirements can in principle be mapped directly into process variables. This is however not how it is done traditionally. Instead, the mapping between functional requirements and design variables, i.e. the design process, is augmented with a second mapping from design variables to process variables, i.e. the manufacturing process. Fortunately, this extension is of the same nature as the design process itself, and can be supported by the same framework.

2.2.2 Design and Modeling

As described above, design is a requirement driven top-down selection process. However, the incremental specification of the functional requirements can only be meaningful in an appropriate modeling context. The design process therefore requires access to modeling information.

Models are valid representations of possible realizations and are described by *model variables.* Typical engineering models are beams, engines, resistors, while height, width, power, resistance are model variables. Models cannot only be created by definition of variables, equations and computations, but also by composition or enumeration of more specific ones. Furthermore they can inherit information of more general models. Design is the selection of sets of realizations in the solution space and therefore requires model selection operators. This includes the specification of parameters, objective functions, model types or names of previous designs.

The representation of models and their organization should be conceived so as to facilitate the selection process. To this end models should contain as much information as possible concerning the feasible realizations while avoiding combinatorial enumeration. Furthermore, design models can be refined by incorporating new information about the design solutions in an incremental fashion.

Establishing the relations between model variables, involves modeling knowledge, i.e. understanding of how a realization provides certain functions. Models can be created by modelers, can come in the form of predefined packages and even as manufacturers catalogs. The formulation of new functional requirements, based on more general design models, on the other hand, requires design knowledge, i.e. an insight as to what functions the design needs to provide to serve its purpose. Both modeling and design, usually rely on problem decomposition to break up bigger problems into smaller ones. Indeed, design requires understanding of how important requirements can be decomposed, while modeling is concerned with defining the right pieces of the model and putting them together.

The design process can also be viewed as model instantiation. In the case of parametric design, this process is straightforward as only parametric computations of prespecified models are required. Furthermore, in this case design knowledge can often be encoded, so that new functional requirements are automatically defined based on information provided by the design model. This implies that for parametric design, the design process can almost entirely be automated. In most cases however, the design process can be computationally intensive, since due to combinatorial complexity it usually is not possible to precompute all candidate solutions at modeling time.

As further scientific progress is made, one can improve existing models and incorporate new models. The automatic generation of entirely new models from novel combinations of functional requirements, which is at the heart of the engineering discipline, does however remains an open challenge.

2.2.3 Design and Set Theory

In a set theoretical sense, all possible realizations represent a set. The size of this solution set is incrementally reduced as the design process proceeds. Sets of solutions are represented by models and functional requirements select desired models from the current solution set.

The set theoretical interpretation of design allows for a very simple definition of hierarchy. Indeed, a model is a *refinement* of another if it is a subset of that model. Note that by this definition the introduction of additional modeling variables automatically constitutes a refinement. It should be emphasised that, as dictated by the subset relation, model hierarchies are only partially ordered. Finally, as with any hierarchy, one can implicitly incorporate an inheritance mechanism and only represent additional information.

Any operator that cuts down the solution space can represent a functional requirement. The boolean And operator for example, represents an intersection with another model that has certain desired properties. Optimization can also be viewed as a selection process retaining only the solutions from the current design model that are optimal w.r.t. a design objective. Observe that an optimum solution can be a set, it does not necessarily have to be a singleton. Note also that different optimization requirements can be compatible as long as one optimum doesn't preclude the other. This trivially holds for uncoupled optimization problems for example, with disjunct optimization domains.

Algebraic equations can also be considered as selection operators for they require a certain relation to hold between different quantities. They can each be viewed as an intersection with all possible solutions satisfying the equation, i.e. the hyperspace defined by the equation. The same consideration holds for equations defining solution strips. These equations, such as for example algebraic equations with interval coefficients, are called interval equations.

When no solutions remain, one obtains the empty set. This means that the models currently available do not contain a realization that can satisfy all functional requirements. Either new models have to be created that provide the desired functionality or the requirements have to be relaxed.

Different methods can be used to represent solution sets. Non-manifold boundary modeling techniques for example, can explicitly represent general point sets, whether continuous or discrete. The complexity of the representation can however be reduced if one considers special cases, such as manifold boundary models or convex point sets. In this thesis we will focus on interval representations, which are in fact a special case of convex point sets and represent the set of values a variable can take.

Discrete point sets can simply be represented by enumeration. This can include types, corresponding to a classification of models based on specific attributes, as well as ordered or unordered lists, corresponding to a set of models.

2.3 A Framework for Design

In the sections below, we describe the key elements that need to be integrated in a framework for design and point to the issues that will be resolved in later chapters.

2.3.1 Bounded Models

As indicated in the previous section, we are concerned with sets of solutions. We therefore propose to develop appropriate representations and computational methods to work with *bounded models*. In addition, to provide for fully reliable computations for design automation, these techniques need to control both approximation errors, such as discretization errors, and rounding errors. This approach should be contrasted with the traditional estimation models, where computations are performed with a representative point solution. Not only do approximate estimation models provide no information concerning the size of the solution sets, the accuracy of the corresponding design computations cannot be automatically verified.

We will say that an approximate bounded model is *conservative* if it encloses the solution set (see figure 2.1). As a consequence, we can no longer simply classify the elements of the domain based on whether or not they are solutions. The overestimation introduced by the conservative nature of the computations requires us to consider three types of sets. The exterior set contains no solutions, whereas all elements of the interior set are solutions. The boundary set contains elements with unknown classification. This is illustrated in figure 2.2. In this work, we will focus on conservative models which approximate the solution set by including both the interior and the boundary set. Note that while approximate

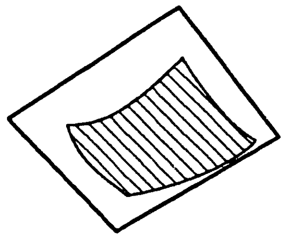


Figure 2.1: Approximation by Conservative Bounded Model

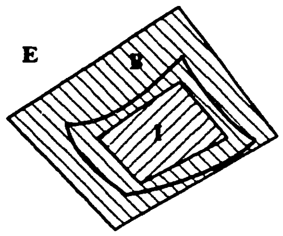


Figure 2.2: Definition of Interior (I), Boundary (B) and Exterior (E) Sets

bounded models enclose all possible realizations that satisfy the functional requirements, not all elements will be solutions. To efficiently prune the search space however, whether with numerical solution algorithms or branch and bound type techniques, it is desired that the overestimation remain small. Indeed, tighter bounds lead to faster elimination of unsatisfactory or suboptimal solutions. On the other hand, sometimes overestimation is introduced to reduce the complexity of the solution algorithms. Conservative enclosures of discrete point sets are a case in point.

As the size of the solution set is reduced, more general models of the hierarchy can be refined allowing for the computation of tighter bounds, so that more accurate models can be used in the selection process. It is important that no additional solutions are introduced in this process. Indeed, if the refined solution set is a subset of the original one, all statements concerning the enclosing solution set hold for the refined set as well. In that case, we say that a refinement is *consistent.* Since we are working with conservative models, we need to distinguish between semantic and procedural consistency. *Semantic consistency* refers to the consistency of the exact solution set, i.e. without overestimation. Consistency of approximate models on the other hand is called *procedural consistency.* Both concepts are illustrated in figure 2.3. Note that the latter definition can also be used for obtaining better approximations when the exact solution remains unchanged.

While semantic consistency preserves the meaning of models, procedural consistency ensures that the associated decision process remains valid. As illustrated in figure 2.3, the two concepts are not identical and cannot freely be interchanged. In particular, if a solution technique is conservative but not procedurally consistent, a refinement can be procedurally consistent only if it is enclosed by all approximations used in the solution process.

As indicated earlier, optimization problems can be associated with functional requirements. Similar techniques can therefore be used to conservatively bound the corresponding solution sets and the same definitions of consistency can be used. Observe that if the value

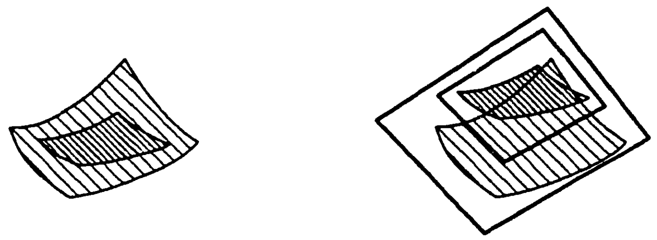


Figure 2.3: Semantic and Procedural Consistency

of the objective function is to be used in the computation, refined definitions should be enclosed by the original one. One of the main advantages of the model refinement technique is that the hierarchical aspects of the design process can be maintained. This is of crucial importance for controlling the complexity of the solution process.

It should be noted that model improvement is fundamentally different from model refinement. While the latter refers to incorporation of information during the design process, the former relates to the definition of new models that supersede older ones. In fact, improved models can be discovered and learned during the design process, so that future performance will be increased.

While bounded models can be represented in different ways, we have chosen to use interval arithmetic for a number of reasons. It provides a natural setting for computations with bounded sets and allows for automatic control of rounding errors. In addition, a large number of techniques have been developed for interval computations. The most important reason however is that the complexity of the algorithms remains polynomial.

We have decided to focus on nonlinear algebraic problems in particular, as the corresponding techniques can be extended to cover a large number of design situations. A brief review of interval arithmetic techniques related to the solution of nonlinear algebraic equations is presented in chapter 3. General nonlinear interval arithmetic solvers conservatively approximate nonlinear systems by linear ones and use linear interval solvers on the resulting system of equations. We have made contributions in both areas. In chapter 5 new algorithms are presented for more efficient computation of nonlinear approximations, while in chapter 4 we describe new linear interval solution algorithms that produce better enclosures then the existing ones.

We would like to indicate that for frequently used models or functions tight bounds can be precomputed. One can for example use techniques based on turning and border points (see figure 2.4) similar to the ones proposed by [36, 138]. In addition, efficient computation strategies can be precompiled. In our opinion this library-like approach, which explicitly incorporates modeling knowledge, will prove very valuable in practice.

Computations with sets of models that include ranges of operations can also be performed in the context of interval arithmetic. In this case, one imposes requirements on the size of the interior set. This type of computation therefore necessitates the use of conservative interior bounds. While very little work has been done on this subject, we believe that most interval techniques can simply extended to handle this case. A preliminary discussion of the inner solution of linear interval equations can already be found in [46], while the applicability of bounding interval endpoints is mentioned in [260].

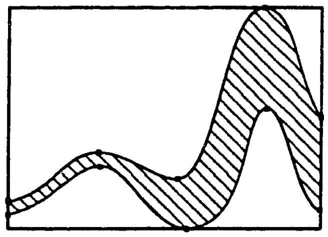


Figure 2.4: Precomputation of Interval Bounds

Appropriate computational models can automatically be selected based on the size of the solution set. Indeed, the width of an interval is usually a good indicator of the required accuracy. In chapter 6 we present a number of novel techniques for variable accuracy interval computations that are based on this idea. These techniques can also be extended to include interval range computations. In this case, the required accuracy should however be controlled by the size of the boundary set.

Finally, the reader is referred to chapter 7 for a discussion of consistency in interval arithmetic computations and practical examples.

2.3.2 Design Graphs

Graphs are a powerful, flexible and efficient way of representing structured information and are in our opinion particularly well suited for representing the dependencies prominent in engineering models. The two most important types of modeling dependencies are the relation between model variables and the hierarchical dependency of different design models. Models are typically associated with nodes, arcs with dependencies. For simplicity the models representing variables will be referred to as variable nodes.

The selection of modeling graphs based on the functional requirements can be done by activation algorithms. For computational graphs associated with equations or functions for example, these algorithms will automatically determine a partial order required for their solution [218]. In this thesis, we focus on reliable solution techniques for strongly coupled components of nonlinear computational graphs.

In general, design graphs can represent distributed computational systems. Moreover nodes can be associated with a variety of different models, as long as they satisfy a number of conditions associated with the nonlinear solution algorithms. For the basic nonlinear interval solvers, described in chapter 3, it is typically required that models provide a conservative linear approximation. More sophisticated methods, such as the ones introduced in chapter 6, can specify the required accuracy and request that the model provide nonlinear solution bounds. It should be emphasised that except for the above interface conditions, the actual computation of this information is internal to the model, in fact it can even be performed by a dedicated processor. In addition, this approach allows for the use of model libraries, which can include frequently used engineering models such as beams, resistors, etc., and manufacturers catalogs, providing product specific information.

While some models are valid throughout the entire domain, others are valid only over a limited domain. In our framework, this information can simply be represented by inheritance of the corresponding bounded model. Furthermore, activation algorithms can automatically select valid models based on the current solution domain.

The hierarchical structure of the design process is solely determined by the functional requirements. The corresponding partial order can therefore only be associated with design models and is not predetermined. Indeed, the inheritance mechanisms used in section 2.3.4 are semantic ones. A hierarchic level is a set of design models that can determine the corresponding value of the design variables, given the value of the functional requirements. The selection of more specific design models typically requires the introduction of new variable nodes. In this case a refinement of the more general models can for example be obtained by node condensation (see chapter 5), so that the complexity of the resulting model remains unchanged. For composed models it sometimes is necessary to introduce variable nodes describing more then one submodel, hereby leading to a partially overlapping decomposition [242]. This situation is illustrated in figure 2.5. Rectangles represent models (e.g. equations), circles variables nodes. The original decomposition, required for model

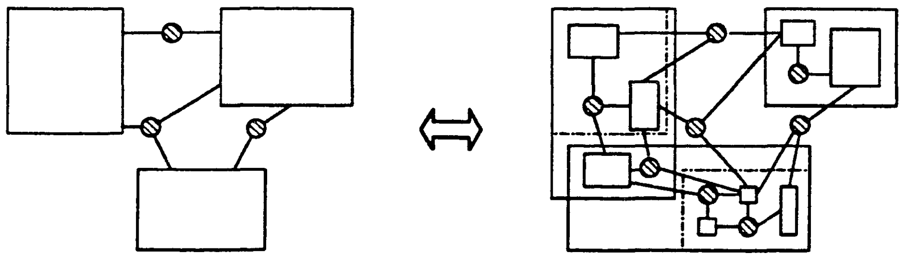


Figure 2.5: Partially Overlapping Decompositions

refinement, can however be reestablished by condensing these particular nodes.

Graph representations also encompass a large number of engineering discretizations. In fact, numerical solution packages use graph algorithms to exploit the sparsity of the associated matrices. We believe that it is therefore beneficial to directly represent the discretization information with graphs and use efficient node condensation techniques for their solution. In our framework, mesh refinement corresponds with automatic definition of functional requirements based on the governing equations, while finite element condensation techniques represent model refinements. Note also that the mesh refinement process can be controlled by the variable accuracy techniques.

Graphs that are not active and their associated semantic networks represent different views of a design model and provide the designer with information deduced from the current design solution (cfr. slices in [242]). Indeed, although the designer is by definition only interested in the functional requirements, he needs to be well informed about the design model so that he can impose appropriate additional functional requirements. While the information emerging from the different views is important, it is not directly controlling the design process, unless new functional requirements are added to do so. Since the nature of the solution usually is not know a priory, specific information requests can however only be made interactively.

2.3.3 Concurrency in Design

Problem subdivision, which is essential in engineering design forms the basis for specialization. In addition, it allows for the concurrent solution of the corresponding subproblems, where *concurrency* refers to parallelism with unbounded resources. We therefore believe that systems for design automation can be organized as a network of simple minded agents, encoding knowledge in a modular way and allowing for concurrency. Furthermore, parallelism is needed to overcome the fundamental speed limitations of current computer technology.

It should be emphasized that concurrent architectures and algorithms are conceptually very different as compared to their sequential counterparts. In fact, it is generally acknowledged that difficult problems have parallel decompositions that yield easier subtasks than serial decompositions. Techniques for automatic discovery of parallel subtasks on the other hand, have enjoyed only limited success due to the associated complexity issues. It is therefore important to incorporate concurrency information so that it is directly available for organizing parallel computation.

Unfortunately, distributed concurrent systems require mechanisms to coordinate local computations and ensure global consistency. Indeed, while local behavior can be assumed to be consistent and sequential (cfr. microtheories in [95]), there is no global knowledge nor time in a concurrent system [96]. In fact, organizations of concurrent decision making agents can show unstable or even chaotic behavior [105].

Computation with bounded models, as proposed in this framework and illustrated in chapter 7, significantly reduces the consistency problem in concurrent systems. Indeed, as decisions are based on solution sets, they also are valid for any subset. Consistency as defined here, therefore allows for asynchronous communications, as well as late arriving and changing information. As compared to the traditional point wise notion of consistency, the coordination mechanisms required by our approach do not constrain the performance of the system as much. Furthermore, global behavior can conveniently be controlled by the use of conservative enclosures.

However, as indicated earlier, inconsistencies can arise in model refinement. In addition, empty solution sets can prompt the designer for redefinitions or parameter changes. Fortunately, these consistency issues are of a topological nature and can be resolved algorithmically. We refer the reader to [144] for a general treatment of strict consistency contracts in concurrent languages. A practical implementation of these contracts requires that conventions about different communication mechanisms be made. The corresponding protocols can be viewed as a generalization of the continuation idea. Commands correspond to a single message, while evaluation protocols consist of both a request and a reply part. Protocols for general distributed control mechanisms, include a specification of the message set and their partial order.

Concurrent computations come to an end when local agents detect no further change and no additional decisions can be made. At this state of quiescence called *design equilibrium*, consistency contracts ensure that the design model reflects current functional requirements. A discussion of the termination problem for asynchronous distributed iterative algorithms can be found in [19]. Note that except for the partial order of the design models selected by the functional requirements, the sequence in which operations are performed is immaterial to the actual solution. External changes can therefore take place at any time and models can dynamically forward queries as soon as model refinements have been made.

Automated design systems require mechanisms to control the cost of computation, including communication and processor time, and both real and virtual memory usage. These more complicated contracts can make explicit tradeoffs between speed, optimality and modeling bounds hereby allowing the designer to impose additional functional requirements before equilibrium is reached.

2.3.4 A Language for Design

To be effective, a design system should be able to capture design as well as modeling information in a clear and unambiguous manner. Furthermore, communication should be done at an appropriate level. In our opinion both requirements entail the creation of a high level language for design that allows us to incorporate as much information as possible concerning the designers knowledge and intent. Indeed, the availability of this information can significantly reduce unnecessary complexity of solution algorithms and is therefore essential for further automation.

A language for design should support both design and modeling constructs. We have already indicated that modeling operators are sometimes used during the design process while modeling can involve selection. Although it is often artificial to distinguish both processes, modeling usually is associated with the expression of definitions, while the application of these definitions and the associate computational process corresponds with design.

One of the most powerful mechanism for expressing information in high level languages is inheritance. Indeed, hierarchical semantic networks and the associated inheritance mechanism can implicitly associate existing information with new definitions. Semantic inheritance and its compact notation can be used for the definition of models as well as relations. While for models this implies the implicit definition of a large number of modeling variables and relations associated with the parent models:

```
(Define Name
  (Model {ParentModel*}
     (VariableBinding*)
    Body))
```

application of Lambda expressions can automatically create a large number of arcs relating its model arguments. The following simple resistor model:

```
(Define Resistor
  (Model {}
     ((current Real)
      (resistance Real)
      (voltage Real))
    (= (* current resistance) voltage)))
```

can be used in series by application of:

```
(Define Connect-in-Series
  (Lambda (Resistor1 Resistor2) {Resistor}
    (= current (current of Resistor1) (current of Resistor2))
    (= resistance (+ (resistance of Resistor1) (resistance of Resistor2)))
    (= voltage (+ (voltage of Resistor1) (voltage of Resistor2)))))
```

Variables defined by semantic inheritance can simply be accessed by name. Model variables associated with components, on the other hand require the use of accessor constructs.

In the above example the left hand variables could be referenced by semantic inheritance of the **Resistor** model, while the variables of the component resistors had to be accessed. In addition, name spaces can be associated with top level inheritance to reduce the complexity of the selection process, hereby providing design specific contexts. Types, which reflect general information associated with models, are themselves first class models. This implies not only that types can be inherited but also that they can form a hierarchy. The **current** variable of **Resistor** for example, inherits the model **Real**, representing the set of all real numbers. **Real** itself could inherit from the type **Complex**.

We want to reiterate that this inheritance mechanism is of a semantic nature. It facilitates the creation of models and the expression of functional requirements, but does not necessarily reflect the hierarchy associated with design models. In the example above one could for example express functional requirements relating to the components before the ones associated with the series resistor.

Boolean operations unify the variables of their arguments. If, for example, we define:

```
(Define Heating-Element
  (Model {Resistor}
    ((power Real))
   (= (* current voltage) power)))
```

the statement:

```
(And Heating-Element Series-Resistor)
```

will automatically identify the variables **current**, **resistance** and **voltage** of both models. This type of identification mechanism is important, for it reduces the number of trivial identities and therefore the complexity of the solution process. The boolean operator **Or** also allows us to enumerate models:

```
(Define resistance (Or 1kΩ 10kΩ))
```

The selection of the solution set by optimization can be expressed by:

(**Optimize** *Objective Model*)

resulting in the subset of *Model* that minimizes *Objective*.

A concise specification of the meaning of language expressions, requires the definition of a semantic domain. Whereas denotational semantics (see chapter 3) uses a λ-calculus denotation, we propose to use graph denotations for the semantic specification of concurrent languages. The semantics of a design language will therefore be expressed as a mapping between syntax and design graphs. As with traditional denotational semantics, this representation is independent of specific parallel computer architectures or network implementations. The topological computational structure will however be directly available to the compiler. In this regard, we would like to indicate that as design graphs will be more structured then computational graphs associated with general purpose languages, the allocation of computational networks on specific architecture can be done more efficiently.

The practical value of graph denotations is dependent on the combinatorial complexity of the associated solution algorithms. As a consequence basic graph denotational components for design will reflect the state of the art in design automation. As this thesis contains an number of contributions in the area of automatic solution techniques, it will allow for more powerful language constructs.

Observe that the design language does not concern itself with message passing, but rather with the construction of appropriate graphs and automatic creation of protocols. Practical implementations will however need to control the interaction of concurrent computational systems with the sequential designer, requiring integration of language editors, command shells and system managers. This approach would also allow for the incremental definition of inconsistency handlers that are invoked upon the detection of inconsistencies. These user defined handlers incorporate design knowledge by automatic respecification of functional requirements.

2.4 Previous work on Design Automation

2.4.1 Design Theory

In *Synthesis of Form*, one of the early works in the area of design theory and methodology, Alexander [8] defines the design problem as a match between context, i.e. functional requirements, and form, i.e. the subject of design. In the *analytical* part of the design process, requirements are analyzed for possible interactions and a minimum information exchange principle is used to attempt to decouple subsystems and devise a hierarchy of requirements. Then, in the *synthesis* part, constructive diagrams are created to incorporate both requirements and form aspects. By composition and fusion of these diagrams, called *hypotheses*, form is synthesized in a bottom up fashion. Throughout this process, binary decision variables are used to indicate fit or misfit between context and form, i.e. if the hypothesis are satisfied.

In our opinion, one can in general not define functional requirements independently of the physical solution. Indeed, a different approach is proposed by Mandel and Chryssostomidis. In [151], they describe a hierarchical method for designing complex systems using the principle of subdivision. As opposed to [8], functional requirements are specified in a top down fashion based on a high level description of the design solution. New design information is then incorporated through the hierarchical use of iterative techniques. In this thesis can be viewed as a continuation of this work, with a specific emphasis on the robustness and consistency problem arizing in the automation of the hierarchical solution process.

Suh, in *Principles of Design* [240], has attempted to provide a scientific basis for design by developing fundamental design axioms. The *independence* axiom dictates that one should maintain the independence of the functional requirements, hereby avoiding coupled design situations. This allows the designer to satisfy the functional requirements without having to make compromises. Finally, by the *information* axiom, of all the design solutions that satisfy the independence axiom, the one with the least information content is defined as the best one.

Although the guidelines offered by this approach are very valuable, we believe that for many practical problems it is usually not possible to maintain the independence of the functional requirements. Nor is it, in our opinion, practical to eliminate some of the functional requirements, or to come up with a different physical solution in order to obtain independence, as would be required by the axiomatic approach. The techniques proposed in this thesis therefore focus on coupled design problems, and can be viewed as a rational approach for making engineering compromises in coupled design situations.

It is generally acknowledged, that optimization is not used to its full potential in the design process. In [244], Taguchi indicates that design techniques based on optimization lead to better and more integrated products. He therefore developed a quality loss minimization framework for design and in order to obtain robust products, defines sensitivity to noise factors as the objective to be minimized. As pointed out by Suh [240], both approaches do not necessarily contradict each other. Indeed, minimization of the information content associated with the functional requirements also promotes simple and robust solutions. However, this imposed criterion often is not compatible with the wishes of the designer and in fact generally precludes a minimum cost design. While we agree that optimization is an integral part of the design process, it is in our opinion not appropriate to prescribe objective functions and hereby to define what good design is. We believe that this should be left to the designer to decide. Note however, that in our framework, it is still possible to define these particular objectives explicitly.

A general theory of design is presented by Yoshikawa in [272, 250]. Design is defined as a mapping from the attribute space to the function space, both of which are defined over the entity concept set. Using three basic axioms, the axiom of recognition, the axiom of correspondence and the axiom of operation, it follows that with ideal knowledge the specification of function directly leads to a design solution. With real knowledge however, it is possible to make design a convergent process, but nor the uniqueness, nor the existence of the solution is guaranteed. This abstract set theoretical view of design should be contrasted to the practical approach presented in this work. Whereas the general design theory focuses on the difference between real and ideal knowledge and pays very little attention to complexity issues, our goal is to devise tractable set algorithms that operate in real knowledge in order to find conservative design solutions.

In recent years researchers have becoming increasingly aware of the need for a sound theoretical basis for design automation. Since a general overview of research in this area is given in the papers by Finger and Dixon [60, 61], we will only review a number of recent contributions.

Painter, in [178], identifies energy as the common denominator in mechanical systems and proposes a unifying framework for design based on bond graph representations, which are independent of geometry. This representation, and the corresponding transformations have in fact recently been used in the area of artificial intelligence for further automation of the synthesis process [254]. An interesting extension of the energy based approach to include geometry is discussed in [220].

Kott and May [125] propose to classify design problems, either as decomposable or as monolithic and suggest that, depending on the type of problem, a different model of the design process should be used. They do however acknowledge that no design problem will be exclusively decomposable, or exclusively monolithic. In our opinion both types of design problems should be allowed in the same framework, where the solution of monolithic or coupled problems typically dictates interface conditions for the subsequent decomposition into decoupled problems.

Other researchers have attempted to cast the design process into explicit iterations between synthesis and analysis [102]. In fact, most domain specific design programs are based on this methodology. In this generate and test approach, analysis has obtained the status of performance checking while other techniques are used to propose design changes.

It is our opinion that one should solve the design equations directly so that the above iterations are kept implicit. In fact, the *design for analysis* [241] approach suggests that designers should only be allowed to use models which can be analyzed in an automated fashion. A number of systematic approaches of exploring the solution space are reviewed by [245]. In this context it should be noted that while our approach allows for discrete enumeration by using enclosures, we will not address the general combinatorial problem associated with discrete point sets.

2.4.2 Modeling

In one of the early control-mathematical theories on hierarchical systems [158], Mesarovic studies multilevel goal-seeking and optimizing systems. Analysis of the conditions under which this type of systems can be decomposed and coordinated in a hierarchical structure lead to the definition of the *interaction decoupling*, *interaction balance*, and *interaction prediction* coordination principles. It is interesting to note that since in our framework coordination is determined by conservative approximation of possible interactions, it is in fact a combination of all three coordination techniques. The more restrictive case of linear systems will not be discussed here. We refer the reader to the literature review presented chapter 5 or to [48] for a general overview.

A number of techniques have been suggested to identify measures of importance. Saaty [211] in particular, proposes to use pairwise comparisons in conjunction with a eigenvalue formulation to obtain weights which reflect the priorities of different parts of the hierarchy. These weights can subsequently be used to analyze the consistency of the comparisons and to guide conflict resolution. Unfortunately, this technique assumes that the pairwise preferences are given and can therefore not be used in an automated fashion.

Hierarchies based on inclusion of operators and metrics were already defined by Manheim in [152] to solve the highway location problem. However, the actual solution is found by a best first search combined with a decision process based on Bayesian probability theory.

Inheritance mechanisms have been used extensively in the field of artificial intelligence [59] as a representation for problem solving systems, see [247] for a recent overview. More recently a formal approach was proposed to represent product model information [54]. Although, this approach is not directly based on set inclusion, it forms in our opinion an important contribution towards extending automation beyond parametric design.

Sussman points out that most engineering systems can only be partioned in almost hierarchical decompositions. Furthermore, these decompositions are not unique. He therefore proposes, not only to consider different levels of detail, but also different views or *slices* [242]. In fact, multiple views of the same model can be used to solve difficult problems. Indeed, exchange of information between different slices allows for the solution of problems which appear hard or impossible in one slice using information derived in another. The solution techniques proposed here do not require redundant information to solve Sussman's hard problems. In our framework views are therefore only used to deduce information concerning the design model and to allow the designer to impose additional functional requirements. Design graphs can in fact be viewed as *metamodels* [249] to integrate the background theories behind the different views.

Graphs are a natural choice to represent the interdependencies in complex engineering design. In [56], it is proposed to use a quality loss criterion to control loss of design freedom in associated decision process. As we use bounded models instead of estimation models, our framework allows for maximum design freedom compatible with previous decisions. An architecture for integrating engineering idealizations in a unified framework is proposed by [124]. However, the use of approximate point models frequently leads to significant errors resulting in wrong decisions [243]. Furthermore, unlike the conservative error control used in this thesis, traditional error estimation provides no guarantee concerning the accuracy of the results, which ultimately need to be verified manually. In this context, it should be noted that the robust numerical methods proposed here considerably extend the applicability of truth maintenance systems [50].

Modeling and database techniques for creating, manipulating and analyzing models of the designed structure are used extensively and have drawn considerable attention of the research community. For recent contributions on the topological aspects of mixed dimensional modeling systems, we refer the reader to [261, 208, 83, 28]. An overview of the current state of the art in geometry processing can be found in [14] while some of the outstanding geometric issues in Computer Aided Design are presented in [37]. Geometry has also extensively been used in the area of automatic analysis, the medial axis techniques [82] in particular, have proven to be very promising. More recently, researchers have attempted to integrate current geometric capabilities in so called *geometric reasoning* systems [268, 197, 269]. While this research furthers our understanding of geometric modeling and analysis, it does, in our opinion, not provide an appropriate organizational framework needed for design automation.

2.4.3 Computation in Design

The computational aspects of the design process encompass a large number of different areas, including optimization, numerical analysis, computer science, and artificial intelligence. Rather then being exhaustive, we will therefore only review the literature relevant to our framework for design.

Only very few general design methods are based on global optimization techniques [244], whereas they have been used extensively in the area of VLSI design [121]. In our opinion, one of the reasons for the limited success of optimization techniques is that general design problems, as opposed to VLSI design, are usually very coupled [200]. Other shortcomings of mechanical design automation are cited in [66]. The area of optimization for design has however enjoyed increased attention in recent years.

For an overview of well known general optimization algorithms we refer the reader to [74], while the specific aspects of translating design problems into constrained optimization problems can be found in [198]. In addition, a number of powerful methods have been developed for shape optimization based on discretization techniques, see for example [256] or [187] for the specific case of elliptic systems. Unfortunately, little work has been done to incorporate optimization techniques in practical computer languages as is proposed in [147].

Papalambros and Wilde suggest that a large number of engineering optimization problems can be solved using *monotonicity analysis* [181]. They derive rigorous analytic formulas for a large number of design optimization problems and are able to show how it can justify engineering design practice. For problems which to not exhibit monotonic behavior, this approach can be combined with numerical analysis techniques [91]. Approximate optimization algorithms need however often to be combined with multistart methods to increase their reliability [109]. A graphic interpretation of problem of attraction basins for optimization is given [35]. While optimization problems are not directly addressed in this thesis, it should be noted that one can simply use the nonlinear solvers of chapter 3 for their solution and that the efficient techniques of chapter 5 can be extended to obtain second order derivatives.

Many designers argue that one really wants to optimize different criteria at the same time. Multiobjective optimization methods, however, make implicit assumptions on the relative importance of the various objectives. We believe that this should be left for the designer to decide under the form of functional requirements.

Combinatorial explosion in large scale design optimization problems can be handled effectively using hierarchical decomposition by relating subproblem objectives to the original optimization problem [114]. A number of numerical techniques for automatic decomposition techniques are described in [22]. Design however is an incremental process and care should be taken to adapt the objective function as more information becomes available. While some methods do indeed allow for incremental changes in the optimization process [42, 75], they do in our opinion not provide the reliability required for design automation. Instead we propose to rely on procedural consistency for incremental selection of the design model.

In general one can view nonlinear algebraic equations as constraints. This approach is suggested by Sussman [242] who uses constraints as fundamental building blocks and constraint propagation as basic model of computation. In addition, his *constraints* language allows for propagation in almost hierarchical systems. To overcome the limitations of the basic propagation algorithm, Serrano extends this work by incorporating numerical solution algorithms to solve strongly connected components and by including techniques for handling inequality constraints [218, 219]. Other hierarchical approaches related to constraint satisfaction have been proposed, see for example [257, 199]. As indicated earlier, one of the contributions of this thesis is to address the robustness problems associated with these numerical solution algorithms.

Some researchers view design as exploring constraints [79] and use constraints to represent design knowledge [80]. Indeed, in many ways models can be viewed as constraints. In this context, we would like to emphasize the importance of automatic differentiation methods. The additional information they derive is not only required for nonlinear solution techniques, but can also be used for the automatic incorporation of changes and to provide an indication as to the relative importance of the variables involved. Constraints can also be used as a computational model for combinatorial problems. A good account of constraint satisfaction and optimization methods for this type of problems can be found in [255]. Finally, it should be noted that particular solution techniques can be selected based on the type of constraints [31].

Constraint propagation techniques were recently extended to include intervals which can represent sets of artifacts [259]. Unfortunately, as the associated iterations terminate quite rapidly [212], these techniques do not allow for the computation of very tight bounds. More importantly, interval propagation algorithms are limited to directed acyclic graphs and can therefore not handle strongly connected components. Since these methods can be viewed as interval evaluation, one can in fact use the more sophisticated enclosure methods described in chapter 3. Finally, the labeling of intervals, used to better describe the constraints associated with the set, can, as we have indicated earlier, significantly be simplified.

Fuzzy set calculus [267] extends interval propagation by associating preferences to individual solutions. Unfortunately, this technique cannot handle coupled systems, moreover the computation of the associated preferences is of combinatorial complexity. In our view, designer preferences concerning particular solutions should explicitly be expressed as functional requirements. While the probabilistic approach [93, 225] is much more powerful and probabilistic analysis is needed to model stochastic uncertainty, it is in our opinion inappropriate for modeling the imprecision uncertainty associated with the selection of alternatives in design.

A very appealing idea is to use continuously decreasing Lyapunov [148] functions for stabilizing nonlinear design systems, including large organizations of intelligent agents [237]. Unfortunately, we were not able to find any literature on applications of this method in the area of design and therefore refer the reader to [17], who uses the dynamic systems metaphor to provide a general framework for design.

Freeman [64] was one of the first to address the problem of design automation from the artificial intelligence perspective. He uses automatic reasoning techniques to connect structures providing certain functions with the ones requiring them. Models therefore directly represent design knowledge, hereby allowing for automatic specification of functional requirements. The knowledge representation and reasoning techniques used in expert systems are in fact very similar, see for example [120]. As indicated earlier, explicit association of design knowledge with models and the corresponding inferences are however bound to be highly problem specific. A more powerful approach was presented by Brown [32], who extends the expert systems technology by providing specific contexts for design knowledge and uses a hierarchical control structure to coordinate a number of simple design experts. Observe that the techniques of chapter 5 can be used to automatically provide these experts with required derivative information. In fact, some of the design expert reasoning techniques are a simplification of approximate nonlinear solution operators.

In addition, researchers in the field of qualitative reasoning are attempting to apply some of their findings to the field of design [62]. In our framework qualitative knowledge is represented by heuristic models, which typically do not have very tight bounds. It was indicated earlier that the use of these models is not compromising the validity of the results as long as more refined models are procedurally consistent.

Frameworks for expert system cooperation [76, 73] and environments to support team based design [262, 235] often use so called *blackboard architectures* [57]. This very popular centralized representation mechanism has in fact been proposed as a fundamental tool for design automation [49]. Any form of centralized control or representation does however form a computational bottleneck in a distributed concurrent problem solving architecture [145]. Indeed, the growing area of distributed problem solving really focuses on coordination of participating agents to achieve global coherence. Early distributed problem solving systems [123] are based on concurrent *Planner*-like architectures with fine grain parallelism. More elaborate meta-level control schemes to achieve global coherence have been suggested by [39] and [145] who introduces *functionally accurate* cooperative distributed systems to deal error-recovery and robustness. A general paradigm for task-sharing based on contract nets and negotiation is proposed by [231, 45]. For more recent work we refer to [52] who introduces a special issue on this subject.

This type distributed problem solving architectures is however relatively coarse grained and typically organized around nodes of a physical network [146]. Moreover, the functionality of the nodes can be compared to human expertise. Since, we are only moderately successful in mimicking human intelligence, let alone in a social context [33], we have proposed to organize systems for design automation as networks of simple computer agents. Furthermore, we believe that to be practical, problem solving architectures should leave implementation aspects to hardware dependent compilers.

A different approach for distributed intelligent systems is based on Pètri Nets [185]. While experimental results comparing hierarchical and other decision making architectures using this computational model can be found in [113], further results concerning organization and coordination of distributed decision makers are presented in [182]. We like to point to [209] in particular who discusses the design of simulation languages with multiple modularities based on this approach. In fact, some protocols required for distributed design systems based have already been established based on Petri Nets [58].

2.5 The Design Society

Design is a social process [33] and from CEO to people in the workshop, a large number of people participate in the design process [21]. We therefore propose to use the *Design Society* metaphor to view systems for design automation as large organizations of simple agents that cooperate coherently to solve design problems. The word society is borrowed from [159] who suggests that human intelligence can best be explained in terms of a society of simple agents. As opposed to human agents these simple agents do not understand how other agents work, they only know how to exploit them. The focus is therefore on integration of different types of knowledge to create a system that appears to be smart.

A design society is made up of large number of different computer agents that solve simple routine subproblems which were previously solved by individual engineers. These agents are organized in a hierarchical manner and are working concurrently to achieve the same goal. Indeed, in *The Architecture of Complexity* [228, 229] Simon points out that among possible systems of a given size and complexity, hierarchical systems are the most likely to appear through evolutionary processes and require much less information transmission among their parts than do other types of systems. The short run behavior of subproblem agencies is therefore approximately independent of other agencies, whereas their long run behavior depends on them only in an aggregate way.

Design agents are influenced by their superiors, colleagues and by their subordinates. The respective influences are determined by the designers requirements and depending on the particular design philosophy, a design society will show more resemblance to a centralized bureaucracy then to a democracy. Furthermore, by charging for processor time, communication cost and memory requirements, explicit compromises can be made between performance and limited resources.

The model selection process based on designers requirements corresponds to activation of Minsky's K-lines and can for example be implemented using the bucket brigade algorithm [75] or activation spreading mechanism [150]. Model refinements and the corresponding level shifts operate very much in the same fashion as the *Subsumption Architecture* [29].

Throughout the design process design agents cooperate with each other by sharing results concerning bounded models. We therefore propose that the design model will be organized around agents of the design society and will be represented in a distributed fashion. Indeed, since they are the originators of information they also are the best keepers of it. The meaning of a design model is therefore defined through the agencies making up the model, for only they can answer queries and analyze dependencies. Furthermore depending on which agency is queried, different views of the same design can be obtained. Indeed, in our opinion, only a small fraction of knowledge is static, and intelligence really has to be associated with processes [30]. We therefore postulate that intelligent design behavior can emerge from interaction simple minded agents and agencies in a natural fashion [100].

We have decided to use the term behavior because of the basic adaptation capabilities of agents in the design society, for automatic solution techniques and incremental specification make design a dynamic rather then a static process. Furthermore, we believe that only a hybrid symbolical–numerical approach [251] can overcome the robustness problems associated with rule based systems [43, 44]. We therefore expect that inconsistency and indeterminacy will cause the performance of the design society to deteriorate gracefully, instead of causing total failure.

The design society also can be used as a metaphor for devising collaborative work platforms. Indeed, a large number of people can interact with local computer agents who are in communication with other agencies over a large network [180]. This approach therefore allows for the use of different languages and compilers as long as the interface to other agencies remains the same. Furthermore, the authority to make design changes can be granted only to a specific agency that can reject individual requests for changes based on consistency considerations. Specialists can be provided different views on the design model by drawing information from appropriate agencies. We could also make a distinction between designers and modelers. Designers decide the requirements of the product, while modelers input new technology and know–how. In this context, partly automated modeling techniques can be allowed, where part of the communication is handled by computer agents, while other messages are forwarded to engineers.

Although it is highly desirable to pick the *better* alternative, it is usually not possible to find the *best* one. Indeed, the hierarchical aspects of our framework are of a heuristic nature and bounded models therefore enclose only *satisfycing* solutions [230]. Finally, in *Administrative Behavior* [227], Simon studies rational decision making in organizations and suggests that two rational agents provided with the same information, can rationally only reach the same decision. He concludes that behavior is therefore not determined by rationality but rather by the limits of rationality. In this thesis we therefore set out to study computational methods for reasoning with information bounds.

Chapter 3

Interval Arithmetic

3.1 Motivation

We are concerned with the automation of the design of complex structures. In this context interval arithmetic is an excellent computational tool for two reasons. First, full automation requires the 100% reliability provided by interval arithmetic, and second the inclusion monotonicity properties are extremely useful for solving the consistency problems associated with the hierarchical design methodology. In addition, it allows for concurrency in a natural fashion, as performance degrades gracefully with excessive interval widths.

We believe that robust solution methods for nonlinear algebraic equations are essential in design automation, and in the present chapter we are therefore investigating this problem.

Although some very reliable techniques have recently been developed for the solution of polynomial equations [138, 184], these methods are currently limited to 2–dimensional problems and do not apply in the case of general nonlinear equations. Classical numerical algorithms are applicable [41, 252], but they do not guarantee that all the solutions will be found, nor is there any guarantee of their accuracy. Moreover, user action is sometimes required in case of singularities or inappropriate starting points.

Instead of using point approximations to the solution, feasible domains can be described by intervals. In the literature, computations involving intervals have been have been referred to as *Interval Arithmetic*. Interval computations, when performed conservatively, are 100% reliable. Numerical solution methods based on these techniques can therefore lead to further automation.

3.2 Basics

Interval arithmetic can be traced back to Young's arithmetic for calculation with sets of numbers, developed in 1931 [273], and later to Dwyer, who considered the special case of closed intervals [53]. The first practical applications, however, date back to Moore in 1959 [160]. And since the mid 1960 interval arithmetic has been an area of active research [86, 162]. There exists a large literature on interval techniques, it covers subjects such as inclusions of the ranges of functions [193], linear interval equations [6], inclusion of zeros complex rational polynomials [186], and optimization [67, 194]. More recently various interval arithmetic packages became available for general use [140, 165].

In this section we give definitions and explain the basic principles of interval arithmetic. We have attempted to use simple notation and terminology consistent with the literature [7, 164, 171, 193, 194].

3.2.1 Definitions

An *interval* $X \in \mathbf{I}$ is a set of numbers $x \in \mathbf{R}$ defined by:

$$X = \{x \mid X_l \leq x \leq X_u\} \tag{3.1}$$

I is the set of intervals and **R** is the set of real numbers. Capital letters denote interval quantities, whereas lowercase letters correspond to numbers. Subscripts are used to indicate entities related to an interval. X_l stands for the *lower bound* of X and X_u for the *upper bound.* When we explicitly refer to the bounds of an interval we will use the following notation:

$$X = [X_l\,,\,X_u]$$

The traditional definition of the *midpoint* of an interval is:

$$X_c = \frac{X_l + X_u}{2}$$

although, for implementation purposes it sometimes is defined as the median of the floating point enumerations between X_l and X_u. When an interval is centered around its midpoint, we have:

$$\Delta X = X - X_c$$

which is a symmetric interval. The *interior* of an interval corresponds with:

$$i(X) = \Box\{x \mid X_l < x < X_u\}$$

$\Box$ indicates the convex hull operation to ensure that the result can be represented with intervals. Finally, *width* is defined as interval norm:

$$w(X) = X_u - X_l$$

and *distance* as interval metric [162, 169]:

$$d(X\,,\,Y) = \max\{|Y_l - X_l|, |Y_u - X_u|\}$$

Note that we use the following convention when referring to absolute values of intervals [168]:

$$\begin{aligned} \langle X \rangle &= \min\{|x| \mid x \in X\} \\ |X| &= \max\{|x| \mid x \in X\} \end{aligned}$$

3.2.2 Operators

Interval arithmetic operations are defined by applying the corresponding algebraic operators to the elements of the interval operands, hereby generating a new interval:

$$X \circ Y = \Box\{x \circ y \mid x \in X, y \in Y\} \tag{3.2}$$

For the usual algebraic operators $\circ \in \{+, -, \cdot, /\}$. Based on this definition we can derive the following explicit expressions:

$$\begin{aligned}
[X_l\,, X_u] + [Y_l\,, Y_u] &= [X_l + Y_l\,, X_u + Y_u] \\
[X_l\,, X_u] - [Y_l\,, Y_u] &= [X_l - Y_u\,, X_u - Y_l] \\
[X_l\,, X_u] \cdot [Y_l\,, Y_u] &= [\min(X_lY_l, X_lY_u, X_uY_l, X_uY_u)\,, \max(X_lY_l, X_lY_u, X_uY_l, X_uY_u)] \\
[X_l\,, X_u]/[Y_l\,, Y_u] &= [X_l\,, X_u] \cdot [1/Y_u\,, 1/Y_l]
\end{aligned} \tag{3.3}$$

As is the case with the multiplication of numbers, we will usually omit the $\cdot$ notation.

The definition of set operations on intervals is straightforward, interval intersection and union, for example are defined as follows:

$$\begin{aligned}
X \cap Y &= \{z \mid z \in X \textit{ and } z \in Y\} \\
X \cup Y &= \{z \mid z \in X \textit{ or } z \in Y\}
\end{aligned}$$

The result of the set operations are not necessarily intervals, an intersection can be the empty set ϕ.

The transition to machine interval arithmetic is made by conservative rounding of the bounds, so that $X \subseteq X_M$ and by ensuring that all machine operations satisfy the inclusion principle:

$$\left.\begin{array}{l} x \in X \\ y \in Y \end{array}\right\} \Rightarrow x \circ y \in (X \circ Y)_M$$

Practical aspects of implementing interval arithmetic on a computer are discussed in [141].

3.2.3 Properties

Most properties of algebraic operators carry over to their interval definition, such as commutativity:

$$\begin{aligned}
X + Y &= Y + X \\
XY &= YX
\end{aligned}$$

and associativity:

$$\begin{aligned}
X + (Y + Z) &= (X + Y) + Z \\
X(YZ) &= (XY)Z
\end{aligned}$$

Distributivity however, is not satisfied in general, but we do have the following *subdistributivity* property:

$$X(Y+Z) \subseteq XY + XZ$$

An interval function $F(X)$ is said to be ***inclusion monotone*** if:

$$X \subseteq Y \Rightarrow F(X) \subseteq F(Y) \tag{3.4}$$

For an interval operator $\circ$ it corresponds to:

$$\left.\begin{array}{l} X' \subseteq X \\ Y' \subseteq Y \end{array}\right\} \Rightarrow X' \circ Y' \subseteq X \circ Y$$

One easily verifies that the explicit expressions (3.3) for $\circ \in \{+, -, \cdot, /\}$ satisfy the above relation.

3.3 Enclosures for the Range of a Function

The challenge of interval arithmetic is to find bounds that are as tight as possible. This problem really is fundamental in that any pointwise dependency of variables is lost when interval representations are used.

Methods to compute an inclusion of the range of functions and their derivatives, can be found in the literature [162, 193]. The simplest method is to use interval evaluation, replacing all variables by their intervals and arithmetic operations by the equivalent interval operations. This can be used for functions as well as for their derivatives after explicit differentiation. Better inclusions are obtained with centered forms. They are often used in conjunction with the more efficient recursive differentiation or Taylor expansion techniques. Recently, Neumaier defined the concept of inclusion algebra [171] to unify different recursive interval techniques. We have chosen to formally define these recursive systems using *Denotational Semantics* [214, 239]. This approach allows us to define a more complete language, including procedures and higher level constructs.

3.3.1 Definitions

The *range* of a function is defined as:

$$f(X) = \square\{f(x) \mid x \in X\}$$

Note that we use a different notation for the range as for an interval enclosure function of that range:

$$f(X) \subseteq F(X)$$

Example 3.1

As illustrated in figure 3.1, $f_1(x) = 1 - 5x + x^3/3$ has range $f_1(X) = [1 - 10\sqrt{5}/3\,, -5]$ over $X = [2\,, 3]$

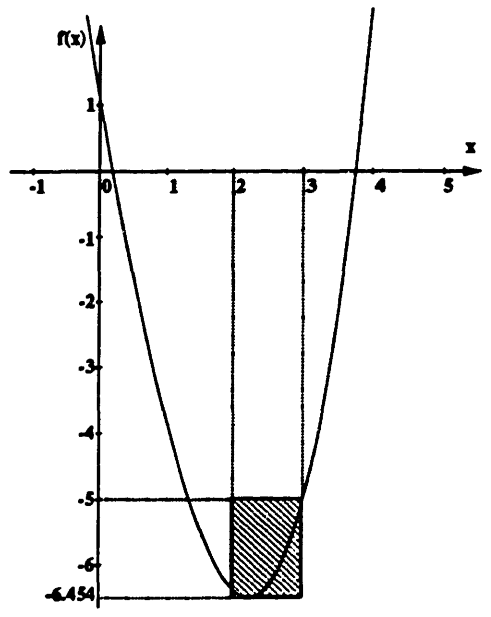


Figure 3.1: Range of $1 - 5x + x^3/3$ over $X = [2\,,3]$

An interval function $F : \mathbf{I} \rightarrow \mathbf{I}$ is ***Lipschitz continuous*** over X if:

$$\forall X', X'' \subseteq X\,,\ \exists L \in \mathbf{R}\ :\ d(F(X')\,, F(X'')) \leq Ld(X'\,, X'') \tag{3.5}$$

the real number L is called the *Lipschitz constant*. An interval enclosure function over X is of order $\alpha > 0$ when:

$$\forall X' \subseteq X\,,\ \exists A \in \mathbf{R}\ :\ d(f(X')\,, F(X')) \leq Aw(X')^{\alpha}$$

The definitions for multivariable functions are very similar. The scalar quantities simply need to be replaced by vector interval quantities, see section 3.4 below.

Example 3.2

The range of $f_2(x, y) = e^{1/(x^2+y^2)}$ over $X = [1\,, 2]$ and $Y = [2\,, 4]$ is $f_2(X, Y) = [1.051\,, 1.222]$

3.3.2 Interval Evaluation

Enclosures of arithmetic expressions can be simply be computed by interval evaluation [162], i.e. by replacing all quantities by their interval equivalents.

Example 3.3

The interval evaluation of $f_1(x)$ of example 3.1 is computed by:

$$\begin{aligned} E(f_1(X), X) &= 1 - 5X + 1/3XXX \\ &= [-11.334\,, 0] \end{aligned} \tag{3.6}$$

over $X = [2, 3]$. Note the conservative rounding of the lower bound -34/3 to -11.334.

To define this interval evaluator with denotational semantics, we need the following domains:

$$v \in \textit{Expressible-Value} = \{ \textit{Interval} + \textit{Procedure} \}$$

$$u \in \textit{Value-Environment} = \textit{Identifier} \mapsto \textit{Expressible-Value}$$

the valuation functions are:

$$\mathcal{C} : \textit{Constant} \mapsto \textit{Expressible-Value}$$

$$\mathcal{E} : \textit{Expression} \mapsto \textit{Value-Environment} \mapsto \textit{Expressible-Value}$$

$\mathcal{C}$ maps constants into expressible values in the obvious way and $\mathcal{E}$ is defined as follows:

$$\mathcal{E}[\![C]\!] \mapsto \lambda \dot{u}.\mathcal{C}[\![C]\!]$$

$$\mathcal{E}[\![I]\!] \mapsto \lambda u.u[\![I]\!]$$

$$\mathcal{E}[\![\texttt{(+ } E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.(\mathcal{E}[\![E_1]\!]u + \mathcal{E}[\![E_2]\!]u)$$

$$\mathcal{E}[\![\texttt{(- } E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.(\mathcal{E}[\![E_1]\!]u - \mathcal{E}[\![E_2]\!]u)$$

$$\mathcal{E}[\![\texttt{(* } E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.(\mathcal{E}[\![E_1]\!]u \cdot \mathcal{E}[\![E_2]\!]u)$$

$$\mathcal{E}[\![\texttt{(/ } E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.(\mathcal{E}[\![E_1]\!]u / \mathcal{E}[\![E_2]\!]u)$$

$$\mathcal{E}[\![\texttt{(lambda (}I\texttt{) } E\texttt{)}]\!] \mapsto \lambda u.\lambda v.\mathcal{E}[\![E]\!](\textit{bind}\, v[\![I]\!]u)$$

$$\mathcal{E}[\![\texttt{(}E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.\mathcal{E}[\![E_1]\!]u(\mathcal{E}[\![E_2]\!]u)$$

The auxiliary function used to bind an identifier in the environment is:

$$\textit{bind} = \lambda v.\lambda I.\lambda u.\lambda I'.(I = I') \mapsto v [\!] uI'$$

Example 3.4

For expression (3.6) of example 3.3 the interval evaluator would be used as follows:

$$\mathcal{E}[\![\texttt{(+ (- 1 (* 5 x)) (* 1/3 (* x (* x x))))}]\!](\textit{bind}[\![\texttt{x}]\!][2, 3]u_0) = [-11.334, 0]$$

where u_0 stands for the initial environment.

Note that this type of recursive evaluation can be done in many different orders and the question arises as to the optimality of one evaluation strategy w.r.t. another, for example:

$$X(1 - X) \subseteq X - XX$$

because of subdistributivity. Theoretically, the best computable enclosure is the intersection of all possible equivalent computations. This issue is similar to ordering floating point operations. Various strategies for obtaining optimal accuracy are discussed in [122, 193].

Based on these definition we can also introduce curried arguments:

$$\mathcal{E}[\![\texttt{(lambda (}I\ I*\texttt{)}\ E\texttt{)}]\!] \mapsto \mathcal{E}[\![\texttt{(lambda (}I\texttt{)(lambda (}I*\texttt{)}\ E\texttt{))}]\!]$$

$$\mathcal{E}[\![\texttt{(}E_0\ E_1\ E*\texttt{)}]\!] \mapsto \mathcal{E}[\![\texttt{((}E_0\ E_1\texttt{)}\ E*\texttt{)}]\!]$$

and define the `let`-form as a syntactic sugar:

$$\mathcal{E}[\![\texttt{(let ((I }E\texttt{)}\ B*\texttt{)}\ E_0\texttt{)}]\!] \mapsto \mathcal{E}[\![\texttt{((lambda (}I\texttt{) (let (}B*\texttt{)}\ E_0\texttt{))}\ E\texttt{)}]\!]$$

This evaluator can be complemented with natural interval extensions of elementary functions:

$$\mathcal{E_F} : \mathit{Function} \mapsto \mathit{Procedure}$$

The exponential function for example, could be included by the following definitions:

$$\mathcal{E}[\![\texttt{(exp }E\texttt{)}]\!] \mapsto \lambda u.\mathcal{E_F}[\![\texttt{exp}]\!](\mathcal{E}[\![E]\!]u)$$

$$\mathcal{E_F}[\![\texttt{exp}]\!] \mapsto \lambda v.[\mathrm{e}^{v_l}, \mathrm{e}^{v_u}]$$

Let us illustrate these extensions with an example.

Example 3.5

The computation of $f_2(x, y)$ of example 3.2 is expressed as follows:

```
𝓔⟦ (let ((square (lambda (v) (* v v))))
      (exp (/ 1 (+ (square x) (square y))))) ⟧
```

$$(\mathit{bind}[\![\mathrm{x}]\!][1, 2](\mathit{bind}[\![\mathrm{y}]\!][2, 4]u_0)) = [1.051, 1.222]$$

which is the exact value of the range.

If the extensions are Lipschitz continuous, it can be shown that extended interval evaluation is also Lipschitz continuous [160, 164]. The enclosures produced by this interval evaluation are of order 1 and therefore of lesser quality. Nevertheless, they are essential for more sophisticated methods. Also, when w(X) is large, interval evaluation often produces more effective bounds than centered forms. In fact, their quadratic convergence implies that the width of the enclosure can grow quadratically.

3.3.3 Centered Forms

The centered form of a function $f(x)$ with center c is an expression of the type:

$$C(f, X, c) = f(c) + G(X - c)(X - c)$$

and was originally introduced by Moore in [162]. This form is of interest because it is of order 2 when G is Lipschitz continuous [137]. Centered forms of rational polynomials were studied by Ratschek and Rokne in [193]. The general *Taylor form* [162, 191, 192] is defined by:

$$T_k(f, X, c) = f(c) + \sum_{j=1}^{k-1} \frac{f^{(j)}(c)}{j!}(X - c)^j + \frac{F^{(k)}(X)}{k!}(X - c)^k$$

The special case $k = 1$ is called the *Mean Value form*:

$$T_1(f, X, c) = f(c) + F'(X)(X - c)$$

If $F'(X)$ is Lipschitz continuous this form has quadratic convergence.

Example 3.6

When applied to $f_1(x)$ as defined in example 3.1:

$$T_1(f_1, [2, 3], 5/2) = [-8.292, -4.291]$$

The value of the interval derivative $F'(X)$ was obtained by interval evaluation of $f'(x)$ over X. A geometric interpretation of the enclosure produced by the mean value form is given in figure 3.2. The shaded area represents the feasible domain for the given $F'(X)$.

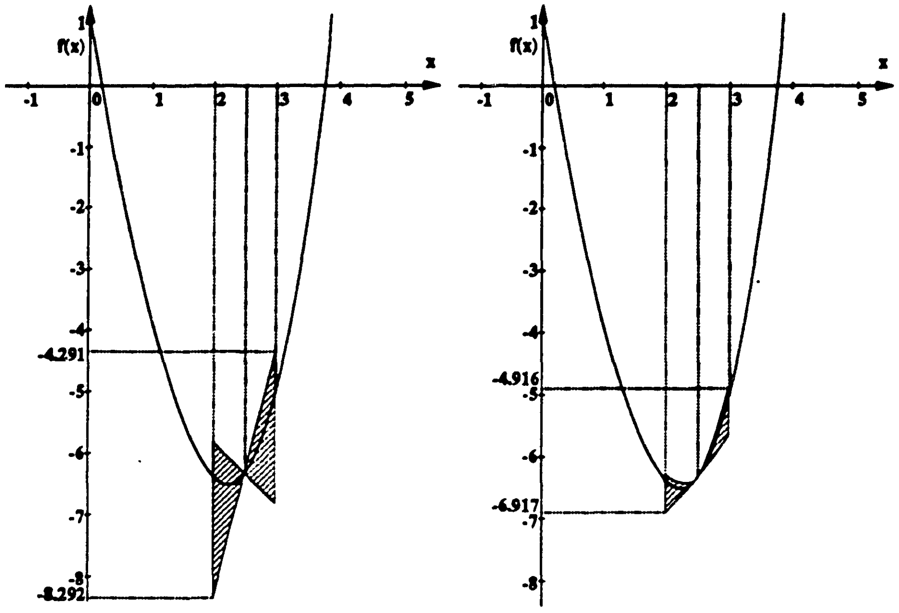


Figure 3.2: Geometric interpretation of the centered forms for the function f_1 of example 3.1

Another very useful Taylor form is the second order one:

$$T_2(f, X, c) = f(c) + f'(c)(X - c) + \frac{F''(X)}{2}(X - c)^2$$

for it can be shown that the boundedness of $F''(X)$ is sufficient to ensure second order convergence. Note that the computation of $(X - c)^n$ can be done exactly by using an interval extension of the power function.

Example 3.7

The second order Taylor form improves on the bound of example 3.6, indeed:

$$\begin{aligned} T_2(f_1, [2, 3], 5/2) &= [-6.917, -4.916] \\ &\subseteq T_1(f_1, [2, 3], 5/2) \end{aligned}$$

This is illustrated in figure 3.2. again $F''(X)$ is computed by interval evaluation of $f''(x)$ over X.

The convergence of the higher order Taylor forms remains of order 2. The inclusion they provide however, is usually increasingly better. If, for example, the higher order derivatives are enclosed by the mean value form, we have:

$$T_k(f, X, c) \subseteq T_{k-1}(f, X, c)$$

Example 3.8

If we use:

$$\begin{aligned} E(f_1'', [2\,,3]) &= [4\,,6] \\ T_1(f_1', [2\,,3], 5/2) &= [-1.75\,, 4.25] \end{aligned}$$

in the computation of the following enclosures:

$$\begin{aligned} T_1(f_1, [2\,,3], 5/2) &= [-8.417\,, -4.166] \\ T_2(f_1, [2\,,3], 5/2) &= [-6.917\,, -4.916] \end{aligned}$$

we have:

$$T_2(f_1, [2\,,3], 5/2) \subseteq T_1(f_1, [2\,,3], 5/2)$$

Again, these definitions can simply by extended to multivariate functions by using interval vectors defined in section 3.4.

3.3.4 Interval Differentiation

Very often, as is the case in the mean value form, it is required to evaluate the range of derivatives. The process of computing the derivative and at the same time evaluating the range of this derivative time is called interval differentiation. A good account of recursive interval differentiation is given by Rall [190]. To formally define a recursive interval differentiator, we introduce some additional semantic domains:

$d \in \textit{Expressible-Differential}$

$g \in \textit{Differential-Environment} = \textit{Identifier} \mapsto \textit{Expressible-Differential}$

the valuation function is:

$\mathcal{D} : \textit{Expression} \mapsto \textit{Value-Environment} \mapsto \textit{Differential-Environment} \mapsto \textit{Expressible-Differential}$

$\mathcal{D}[\![C]\!] \mapsto \lambda u.\lambda g.zero$

$\mathcal{D}[\![I]\!] \mapsto \lambda u.\lambda g.g[\![I]\!]$

$\mathcal{D}[\![\texttt{(+}\ E_1\ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda g.(\mathcal{D}[\![E_1]\!]ug + \mathcal{D}[\![E_2]\!]ug)$

$\mathcal{D}[\![\texttt{(-}\ E_1\ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda g.(\mathcal{D}[\![E_1]\!]ug - \mathcal{D}[\![E_2]\!]ug)$

$$\mathcal{D}[\![\texttt{(* } E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.\lambda g.(\mathcal{D}[\![E_1]\!]ug \cdot \mathcal{E}[\![E_2]\!]u + \mathcal{E}[\![E_1]\!]u \cdot \mathcal{D}[\![E_2]\!]ug)$$

$$\mathcal{D}[\![\texttt{(/ } E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.\lambda g.(\mathcal{D}[\![E_1]\!]ug - \mathcal{E}[\![\texttt{(/ } E_1 \; E_2\texttt{)}]\!]u \cdot \mathcal{D}[\![E_2]\!]ug)/\mathcal{E}[\![E_2]\!]u$$

$$\mathcal{D}[\![\texttt{(lambda (}I\texttt{) }E\texttt{)}]\!] \mapsto \lambda u.\lambda g.\lambda v.\lambda d.\mathcal{D}[\![E]\!](\mathit{bind}\, v[\![I]\!]u)(\mathit{bind}\, d[\![I]\!]g)$$

$$\mathcal{D}[\![\texttt{(}E_1 \; E_2\texttt{)}]\!] \mapsto \lambda u.\lambda g.\mathcal{D}[\![E_1]\!]ug(\mathcal{E}[\![E_2]\!]u)(\mathcal{D}[\![E_2]\!]ug)$$

Example 3.9

As in example 3.4, the first interval derivative is simply obtained by:

$$\mathcal{D}[\![\texttt{(+ (- 1 (* 5 x)) (* 1/3 (* x (* x x))))}]\!](\mathit{bind}[\![\texttt{x}]\!][2\,,3]u_0)(\mathit{bind}[\![\texttt{x}]\!][1]d_0) = [-1\,,4]$$

u_0 is again the initial value environment and d_0 is the initial derivative environment. As we noted earlier, interval evaluation sometimes produces better enclosures than more sophisticated methods. Indeed, in this example recursive differentiation is better than the mean value form computed in example 3.8:

$$\begin{aligned} D(f_1,[2\,,3]) &= E(f_1',[2\,,3]) \\ &\subseteq T_1(f_1',[2\,,3],5/2) \end{aligned}$$

Introduction of elementary functions proceeds as with the interval evaluator:

$$\mathcal{D}[\![\texttt{(exp }E\texttt{)}]\!] \mapsto \lambda u.\lambda g.\mathcal{D}_{\mathcal{F}}[\![\texttt{exp}]\!](\mathcal{E}[\![E]\!]u)(\mathcal{D}[\![E]\!]ug)$$

$$\mathcal{D}_{\mathcal{F}}[\![\texttt{exp}]\!] \mapsto \lambda v.\lambda d.[\mathrm{e}^{v_l}\,,\mathrm{e}^{v_u}]\cdot d$$

Example 3.10

If we go back to example 3.5 we now have:

$\mathcal{D}[\![$ `(let ((square (lambda (v) (* v v))))`
`(exp (/ 1 (+ (square x) (square y)))))` $]\!]$

$$(\mathit{bind}[\![\texttt{x}]\!][1\,,2](\mathit{bind}[\![\texttt{y}]\!][2\,,4]u_0))(\mathit{bind}[\![\texttt{x}]\!][1](\mathit{bind}[\![\texttt{y}]\!][0]d_0)) = [-0.196\,,-0.005]$$

$\mathcal{D}[\![$ `(let ((square (lambda (v) (* v v))))`
`(exp (/ 1 (+ (square x) (square y)))))` $]\!]$

$$(\mathit{bind}[\![\texttt{x}]\!][1\,,2](\mathit{bind}[\![\texttt{y}]\!][2\,,4]u_0))(\mathit{bind}[\![\texttt{x}]\!][0](\mathit{bind}[\![\texttt{y}]\!][1]d_0)) = [-0.391\,,-0.011]$$

which can be used in the multivariate mean value form:

$$T_1(f_2,([1\,,2]\,[2\,,4])^T,([3/2]\,[3])^T) = [0.604\,,1.582]$$

3.3.5 Implementation

We illustrate the above semantic concepts with *Scheme* code of a similar evaluator. For efficiency both interval evaluation and differentiation are performed at the same time:

```
(define general-eval (expression environment)
  (cond ((number? expression)
         (number-eval expression))
        ((variable? expression)
         (variable-eval expression environment))
        ((primitive? expression)
         (primitive-eval expression environment))
        ((lambda? expression)
         (lambda-eval expression environment))
        (t
         (application-eval expression environment))))

(define number-eval (expression)
  (let ((value-interval (point-interval expression))
        (derivative-interval (point-interval 0)))
    (make-tuple value-interval derivative-interval)))

(define variable-eval (expression environment)
  (lookup expression environment))

(define primitive-eval (expression environment)
   (let* ((operator (primitive-operator expression))
          (operand1 (primitive-operand1 expression))
          (operand2 (primitive-operand2 expression))
          (tuple1 (general-eval operand1 environment))
          (tuple2 (general-eval operand2 environment)))
          (value1 (tuple-value tuple1))
          (value2 (tuple-value tuple2))
          (derivative1 (tuple-derivative tuple1))
          (derivative2 (tuple-derivative tuple2)))
     (cond ((+? operator) (+tuple value1 value2 derivative1 derivative2)
           ((-? operator) (-tuple value1 value2 derivative1 derivative2))
           ((*? operator) (*tuple value1 value2 derivative1 derivative2))
           ((/? operator) (/tuple value1 value2 derivative1 derivative2)))))

(define lambda-eval (expression environment)
  (let ((identifier (lambda-identifier expression))
        (body (lambda-body expression)))
    (lambda (tuple)
      (general-eval body (bind identifier tuple environment)))))

(define application-eval (expression environment)
  (let ((operator (application-operator expression))
        (operand (application-operand expression)))
    ((general-eval operator environment)
     (general-eval operand environment))))
```

The primitive operations are defined as follows:

```
(define +tuple (value1 value2 derivative1 derivative2)
  (make-tuple (i+ value1 value2) (i+ derivative1 derivative2)))

(define -tuple (value1 value2 derivative1 derivative2)
  (make-tuple (i- value1 value2) (i- derivative1 derivative2)))
```

```
(define *tuple (value1 value2 derivative1 derivative2)
  (make-tuple (i* value1 value2)
              (i+ (i* derivative1 value2) (i* value1 derivative2))))

(define /tuple (value1 value2 derivative1 derivative2)
  (let ((value1/value2 (i/ value1 value2)))
    (make-tuple value1/value2
                (i/ (i- derivative1 (i* value1/value2 derivative2)) value2))))
```

Where `i+`, `i-`, `i*` and `i/` are the arithmetic interval operators. To be complete we include a simple implementation of the auxiliary functions:

```
    (define bind (identifier tuple environment)
      (lambda (other-identifier)
        (if (equal other-identifier identifier) tuple (environment identifier))))

(define lookup (identifier environment)
  (environment identifier))
```

3.3.6 Higher Order Taylor Forms

In the same spirit as the interval differentiator, higher order Taylor coefficients, needed for the computation of enclosures with higher order, can be computed recursively [162, 164, 191]. Using denotational semantics we can formally define this computation as follows:

$$x, y \in \textit{Variable-Set}$$

$$t \in \textit{Taylor-Environment} = \textit{Identifier} \mapsto \textit{Variable-Set} \mapsto \textit{Expressible-Taylor-Coefficient}$$

$$f \in \textit{Coefficient-Environment} = \textit{Variable-Set} \mapsto \textit{Expressible-Taylor-Coefficient}$$

with valuation function:

$$\mathcal{T} : \textit{Expression} \mapsto \textit{Taylor-Environment} \mapsto \textit{Variable-Set} \mapsto \textit{Expressible-Taylor-Coefficient}$$

$$\mathcal{T}[\![C]\!] \mapsto \lambda t.\lambda x.(x = \phi) \mapsto \mathcal{C}[\![C]\!][\,]\textit{zero}$$

$$\mathcal{T}[\![I]\!] \mapsto \lambda t.\lambda x.t[\![I]\!]x$$

$$\mathcal{T}[\![\texttt{(+ } E_1\ E_2\texttt{)}]\!] \mapsto \lambda t.\lambda x.(\mathcal{T}[\![E_1]\!]tx + \mathcal{T}[\![E_2]\!]tx)$$

$$\mathcal{T}[\![\texttt{(- } E_1\ E_2\texttt{)}]\!] \mapsto \lambda t.\lambda x.(\mathcal{T}[\![E_1]\!]tx - \mathcal{T}[\![E_2]\!]tx)$$

$$\mathcal{T}[\![\texttt{(* } E_1\ E_2\texttt{)}]\!] \mapsto \lambda t.\lambda x.\textstyle\sum_{i=0}^{n} \mathcal{T}[\![E_1]\!]t(x)_i \cdot \mathcal{T}[\![E_2]\!]t(x\backslash(x)_i)$$

$$\mathcal{T}[\![\texttt{(/ } E_1\ E_2\texttt{)}]\!] \mapsto \lambda t.\lambda x.(\mathcal{T}[\![E_1]\!]tx - \textstyle\sum_{i=1}^{n} \mathcal{T}[\![E_2]\!]t(x)_i \cdot \mathcal{T}[\![\texttt{(/ } E_1\ E_2\texttt{)}]\!]t(x\backslash(x)_i))/\mathcal{T}[\![E_2]\!]t\phi$$

$$\mathcal{T}[\![\texttt{(lambda (}I\texttt{) }E\texttt{)}]\!] \mapsto \lambda t.\lambda x.\lambda f.\lambda y.\mathcal{T}[\![E]\!](\textit{bind}\, f[\![I]\!]t)y$$

$$\mathcal{T}[\![\texttt{(}E_1\ E_2\texttt{)}]\!] \mapsto \lambda t.\lambda x.(\mathcal{T}[\![E_1]\!]tx)(\mathcal{T}[\![E_2]\!]t)x$$

In the above summations, all subsets $(x)_i$ of x are enumerated starting with the empty set for $\sum_{i=0}^{n}$ and with singletons in the case of $\sum_{i=1}^{n}$.

Example 3.11

To obtain the interval Taylor coefficient used in example 3.7 we simply evaluate:

$$\mathcal{T}\llbracket\texttt{(+ (- 1 (* 5 x)) (* 1/3 (* x (* x x))))}\rrbracket(\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{\phi}[2\,,3](\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{[\mathtt{x}]}[1]t_0))\{\llbracket\mathtt{x}\rrbracket\llbracket\mathtt{x}\rrbracket\} = [2\,,3]$$

with initial environment t_0.

An enclosure for the Taylor coefficients of the exponential function, for example, would be:

$$\mathcal{T}\llbracket\texttt{(exp } E\texttt{)}\rrbracket \mapsto \lambda t.\lambda x.\mathcal{T}_{\mathcal{F}}\llbracket\mathtt{exp}\rrbracket(\mathcal{T}\llbracket E\rrbracket t)x$$

$$\mathcal{T}_{\mathcal{F}}\llbracket\mathtt{exp}\rrbracket \mapsto \lambda f.\lambda y.(y = \phi) \mapsto [\mathrm{e}^{(f\phi)_l}\,,\mathrm{e}^{(f\phi)_u}] \,[]\, \textstyle\sum_{i_1=1}^{n_1} i_1/n_1\, f(y)_{i_1} \cdot \mathcal{T}_{\mathcal{F}}\llbracket\mathtt{exp}\rrbracket f(y\backslash(y)_{i_1})$$

The index i_1 refers to the first element of y. The following example illustrates the use of the above extension.

Example 3.12

Referring again to the expression of example 3.5, the 2nd order Taylor coefficients are:

$$\mathcal{T}\llbracket\begin{array}{l}\texttt{(let ((square (lambda (v) (* v v))))}\\ \quad\texttt{(exp (/ 1 (+ (square x) (square y)))))}\end{array}\rrbracket$$
$$(\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{\phi}[1\,,2](\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{[\mathtt{x}]}[1] (\mathit{bind}\,\llbracket\mathtt{y}\rrbracket_{\phi}[2\,,4](\mathit{bind}\,\llbracket\mathtt{y}\rrbracket_{[\mathtt{y}]}[1]t_0))))\{\llbracket\mathtt{x}\rrbracket\llbracket\mathtt{x}\rrbracket\} = [-0.047\,,0.160]$$

$$\mathcal{T}\llbracket\begin{array}{l}\texttt{(let ((square (lambda (v) (* v v))))}\\ \quad\texttt{(exp (/ 1 (+ (square x) (square y)))))}\end{array}\rrbracket$$
$$(\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{\phi}[1\,,2](\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{[\mathtt{x}]}[1] (\mathit{bind}\,\llbracket\mathtt{y}\rrbracket_{\phi}[2\,,4](\mathit{bind}\,\llbracket\mathtt{y}\rrbracket_{[\mathtt{y}]}[1]t_0))))\{\llbracket\mathtt{x}\rrbracket\llbracket\mathtt{y}\rrbracket\} = [0.002\,,0.688]$$

$$\mathcal{T}\llbracket\begin{array}{l}\texttt{(let ((square (lambda (v) (* v v))))}\\ \quad\texttt{(exp (/ 1 (+ (square x) (square y)))))}\end{array}\rrbracket$$
$$(\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{\phi}[1\,,2](\mathit{bind}\,\llbracket\mathtt{x}\rrbracket_{[\mathtt{x}]}[1] (\mathit{bind}\,\llbracket\mathtt{y}\rrbracket_{\phi}[2\,,4](\mathit{bind}\,\llbracket\mathtt{y}\rrbracket_{[\mathtt{y}]}[1]t_0))))\{\llbracket\mathtt{y}\rrbracket\llbracket\mathtt{y}\rrbracket\} = [-0.040\,,0.676]$$

which gives us the following second order enclosure:

$$T_2(f_2,([1\,,2]\,[2\,,4])^T,([3/2]\,[3])^T) = [0.633\,,2.218]$$

which in this case is of lesser quality then the mean value bound.

3.3.7 Interval Slopes

Krawczyk [127] found that if *interval slopes* $S(f,X,c)$, defined by:

$$f(x) - f(c) \in S(f,X,c)(x-c)$$

instead of derivatives are used to develop the centered form:

$$R(f,X,c) = f(c) + S(f,X,c)(X-c)$$

better enclosures are produced than with the mean value form. As is the case for derivatives. interval slopes can be computed recursively. It can be shown [171] that for the slopes computed by the slope evaluator described below:

$$w(S(f,X,c)) = \frac{w(D(f,X))}{2} + \mathcal{O}(w(X)^2) \qquad (3.7)$$

In addition, the slopes $S(f,X,c)$ are Lipschitz continuous so that *Krawczyk's centered form* is also of order 2.

For Krawczyk's interval slopes evaluator, we define the following semantic domains:

$p \in$ *Expressible-Value*

$t \in$ *Expressible-Slope*

$c \in$ *Center-Environment* = *Identifier* $\mapsto$ *Expressible-Value*

$s \in$ *Slope-Environment* = *Identifier* $\mapsto$ *Expressible-Slope*

The valuation functions is:

$\mathcal{S}$: *Expression* $\mapsto$ *Value-Environment* $\mapsto$ *Center-Environment* $\mapsto$ *Slope-Environment* $\mapsto$ *Expressible-Slope*

$\mathcal{S}[\![C]\!] \mapsto \lambda u.\lambda c.\lambda s.zero$

$\mathcal{S}[\![I]\!] \mapsto \lambda u.\lambda c.\lambda s.s[\![I]\!]$

$\mathcal{S}[\![\texttt{(+ } E_1 \ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.(\mathcal{S}[\![E_1]\!]ucs + \mathcal{S}[\![E_2]\!]ucs)$

$\mathcal{S}[\![\texttt{(- } E_1 \ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.(\mathcal{S}[\![E_1]\!]ucs - \mathcal{S}[\![E_2]\!]ucs)$

$\mathcal{S}[\![\texttt{(* } E_1 \ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.(\mathcal{E}[\![E_1]\!]u \cdot \mathcal{S}[\![E_2]\!]ucs + \mathcal{S}[\![E_1]\!]ucs \cdot \mathcal{E}[\![E_2]\!]c)$

$\mathcal{S}[\![\texttt{(/ } E_1 \ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.(\mathcal{S}[\![E_1]\!]ucs - \mathcal{E}[\![\texttt{(/ } E_1 \ E_2\texttt{)}]\!]c \cdot \mathcal{S}[\![E_2]\!]ucs)/\mathcal{E}[\![E_2]\!]u$

$\mathcal{S}[\![\texttt{(lambda (}I\texttt{) }E\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.\lambda v.\lambda p.\lambda t.\mathcal{S}[\![E]\!](bind\, v[\![I]\!]u)(bind\, p[\![I]\!]c)(bind\, t[\![I]\!]s)$

$\mathcal{S}[\![\texttt{(}E_1 \ E_2\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.\mathcal{S}[\![E_1]\!]ucs\,(\mathcal{E}[\![E_2]\!]u)(\mathcal{E}[\![E_2]\!]c)(\mathcal{S}[\![E_2]\!]ucs)$

Example 3.13

We apply this recursive evaluator to expression (3.6) of example 3.3:

$\mathcal{S}[\![\texttt{(+ (- 1 (* 5 x)) (* 1/3 (* x (* x x))))}]\!]$
$(bind\,[\![\texttt{x}]\!][2\,,3]u_0)(bind\,[\![\texttt{x}]\!][5/2]c_0)(bind\,[\![\texttt{x}]\!][1]s_0) = [0.083\,,2.584]$

u_0, c_0 and s_0 are the initial value, center and slope environments. With this interval slope Krawczyk's centered form gives us the following enclosure:

$$R(f_1,[2\,,3],5/2) = [-7.584\,,-5]$$

which is better than $T_1(f_1,[2\,,3],5/2)$ of example 3.6.

In this case the exponentiation function would be defined as follows:

$$\mathcal{S}[\![\texttt{(exp } E\texttt{)}]\!] \mapsto \lambda u.\lambda c.\lambda s.\mathcal{S}_{\mathcal{F}}[\![\texttt{exp}]\!](\mathcal{E}[\![E]\!]u)(\mathcal{E}[\![E]\!]c)(\mathcal{S}[\![E]\!]ucs)$$

$$\mathcal{S}_{\mathcal{F}}[\![\texttt{exp}]\!] \mapsto \lambda v.\lambda p.\lambda t.[\mathrm{e}^{v_l}, \mathrm{e}^{v_u}] \cdot t$$

Example 3.14

The slope for the expression of example 3.10 would be:

$\mathcal{S}[\![$ `(let ((square (lambda (v) (* v v))))`
`(exp (/ 1 (+ (square x) (square y)))))` $]\!]$

$$(bind\,[\![\mathrm{x}]\!][1\,,2](bind\,[\![\mathrm{y}]\!][2\,,4]u_0))$$
$$(bind\,[\![\mathrm{x}]\!][3/2](bind\,[\![\mathrm{y}]\!][3]c_0))$$
$$(bind\,[\![\mathrm{x}]\!][1](bind\,[\![\mathrm{y}]\!][0]s_0)) = [-0.076\,, -0.011]$$

$\mathcal{S}[\![$ `(let ((square (lambda (v) (* v v))))`
`(exp (/ 1 (+ (square x) (square y)))))` $]\!]$

$$(bind\,[\![\mathrm{x}]\!][1\,,2](bind\,[\![\mathrm{y}]\!][2\,,4]u_0))$$
$$(bind\,[\![\mathrm{x}]\!][3/2](bind\,[\![\mathrm{y}]\!][3]c_0))$$
$$(bind\,[\![\mathrm{x}]\!][0](bind\,[\![\mathrm{y}]\!][1]s_0)) = [-0.152\,, -0.023]$$

which is better then the enclosure produced by the differentiator. The Krawczyk centered form accordingly improves the bound of the mean-value form given in example 3.10:

$$R(f_2, ([1\,,2]\,[2\,,4])^T, ([3/2]\,[3])^T) = [0.902\,, 1.283]$$

3.4 Interval Matrices

3.4.1 Definition

A multidimensional interval array simply is a multidimensional array with interval components. For example, an interval vector $X \in \mathbf{I}^n$ is a set of vectors $x \in \mathbf{R}^n$ with:

$$X = \{x \mid x_i \in X_i, i = 1, \ldots, n\}$$

It represents an N-dimensional box aligned with the coordinate axes. Similarly, an interval matrix $A \in \mathbf{I}^{m\times n}$ is a set of real matrices $a \in \mathbf{R}^{m\times n}$ defined as:

$$A = \{a \mid a_{ij} \in A_{ij}, i = 1, \ldots, m\,,\; j = 1, \ldots, n\}$$

Many quantities related to intervals can simply be extended to the multidimensional case by componentwise application of the corresponding interval definition. For example, if we use the componentwise notation $A = (A_{ij})$ for $A \in \mathbf{I}^{m\times n}$, we have:

$$\begin{aligned}(A_{ij})_l &= (A_{ij_l})\\ (A_{ij})_u &= (A_{ij_u})\\ (A_{ij})_c &= (A_{ij_c})\end{aligned}$$

The definitions of width and distance do however need to be adapted to deal with multiple dimensions:

$$\begin{aligned} w((A_{ij})) &= \|(w(A_{ij}))\|_\infty \\ d((A_{1ij}),(A_{2ij})) &= \|(d(A_{1ij},A_{2ij}))\|_\infty \end{aligned}$$

We use the row sum norm of a matrix defined as:

$$\|A\|_\infty = \max_{i=1,\dots,m}\{\sum_{j=1}^{n}|A_{ij}|\}$$

3.4.2 Operators

Operations on multidimensional arrays are defined the same way as in definition (3.2). For example if $A \in \mathbf{I}^{m\times n}$ and $B \in \mathbf{I}^{n\times p}$ we define:

$$A \circ B = \square\{a \circ b \mid a \in A, b \in B\} \tag{3.8}$$

with $\circ \in \{+,-,\cdot\}$. Explicit expressions for these operators are:

$$\begin{aligned} (A_{ij})+(B_{ij}) &= (A_{ij}+B_{ij}) \\ (A_{ij})-(B_{ij}) &= (A_{ij}-B_{ij}) \\ (A_{ij})\cdot(B_{ij}) &= (\textstyle\sum_{k=1}^{n} A_{ik}B_{kj}) \end{aligned}$$

Note that for the interval representation of a matrix multiplication, as defined in equation (3.8), in general we do not have strict equality, but rather:

$$\{a\cdot b \mid a \in A, b \in B\} \subseteq A\cdot B \tag{3.9}$$

3.4.3 Properties

The commutativity of $+$ and $\cdot$ is maintained when applied to higher dimensional vectors, as is subdistributivity. But although $+$ is associative, $\cdot$ in general is not. However, in case $w(P)=0$, i.e. P is a point matrix, we have:

$$\begin{aligned} (PA)B &\subseteq P(AB) \\ A(BP) &\subseteq (AB)P \end{aligned}$$

Finally, one can show that the higher dimensional operators as defined in equation (3.8) also are inclusion monotone.

3.4.4 Inverse

The inverse of a regular square interval matrix is defined as:

$$A^{-1} = \square\{a^{-1} \mid a \in A\}$$

An interval matrix is *regular* if it contains no singular point matrices. Verifying the regularity of interval matrices is, in general, very difficult. Some regularity tests can be found in [171]. For certain classes of matrices the problem becomes more tractable. In particular, the verification of regularity of $A_c^{-1}A$, i.e. strong regularity, can be done by checking for strict diagonal dominance. The latter holds if:

$$\|A_c^{-1}A - I\|_\infty < 1 \tag{3.10}$$

Note that $(A_c^{-1}A)_c = I$.

3.5 Linear Interval Equations

3.5.1 Definition

The solution $X_\sigma \in \mathbf{I}^n$ of $AX = B$ with $A \in \mathbf{I}^{n\times n}$ and $B \in \mathbf{I}^n$ is defined as:

$$X_\sigma = \square\{x \ : \ ax = b \mid a \in A, b \in B\} \tag{3.11}$$

When A is regular this corresponds to:

$$X_\sigma = \square\{a^{-1}b \mid a \in A, b \in B\} \subseteq A^{-1}B$$

It is a remarkable fact that no strict equality is obtained in the above expression. If in addition the solution is restricted to a given interval X, we use the following notation:

$$\Sigma(A, B, X) = X_\sigma \cap X$$

The solution of interval equations is inclusion monotone, in the sense that:

$$\left.\begin{array}{l} A' \subseteq A \\ B' \subseteq B \end{array}\right\} \Rightarrow X'_\sigma \subseteq X_\sigma$$

For a square system of N interval equations, there can be up to $\mathcal{O}(2^{2N})$ linear point systems bounding the exact solution. So, the difficulty is to find tight bounds for the solution of this equation while avoiding combinatorial explosion. This problem is relatively well understood, and a thorough treatment of possible solution methods can be found in [171].

Example 3.15

To illustrate the complexity of interval equations we borrow the following simple example from [86]:

$$\begin{array}{rcl} [2\,,3]X_1 + [0\,,1]X_2 & = & [0\,,120] \\ [1\,,2]X_1 + [2\,,3]X_2 & = & [60\,,240] \end{array} \tag{3.12}$$

The exact solution is represented by the shaded area in figure 3.3, the corresponding interval solution is:

$$X_\sigma = [-120\,,90] \times [-60\,,240]$$

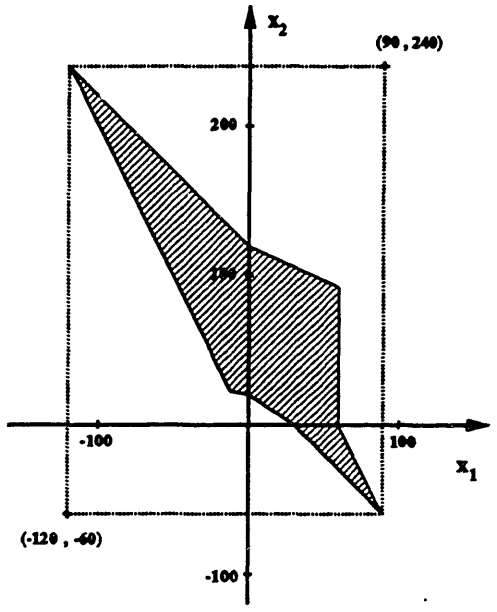


Figure 3.3: Exact solution of interval equations (3.12)

Solution methods can be simplified by preconditioning the system with the midpoint inverse [90, 130, 157, 168] and use the solution X_p of $(A_c^{-1}A)X = A_c^{-1}B$ instead, with $X_\sigma \subseteq X_p$. The overestimation occurring by this simplification is quantified by:

$$w(X_\sigma) \leq w(X_p) \leq \frac{1+\beta}{1-\beta} w(X_\sigma)$$

for $\|A_c^{-1}A - I\|_\infty \leq \beta < 1$ [171]. Unfortunately this implies that considerable overestimation is possible when $\beta \approx 1$.

3.5.2 Krawczyk's method

Krawczyk's operator [126] is defined by:

$$\mathcal{K}(A, B, X) = (B - (A - I)X) \cap X \tag{3.13}$$

It has the property:

$$X_\sigma \subseteq X \Rightarrow X_\sigma \subseteq \mathcal{K}(A, B, X)$$

For simplicity we will assume here that $X_\sigma \subseteq X$. The enclosures produced by Krawczyk's operator on simple initial enclosures have the quadratic approximation property if A is regular, i.e. if both $w(A)$ and $w(B)$ are of $\mathcal{O}(\epsilon)$ then $d(\mathcal{K}(A, B, X), X_\sigma)$ is of order $\mathcal{O}(\epsilon^2)$.

When A is strongly regular we can apply Krawczyk's operator on the following crude bound:

$$X_\sigma \subseteq X_0 = \langle A \rangle^{-1} |B| [-1, 1] \tag{3.14}$$

to obtain Krawczyk's direct inverse solution:

$$X_{\kappa d} = B + \langle A\rangle^{-1}|A - I||B|[-1\,,1] \tag{3.15}$$

$\langle A\rangle$ is the comparison matrix [177], with:

$$\begin{aligned} \langle A\rangle_{ii} &= \langle A_{ii}\rangle \\ \langle A\rangle_{ij} &= -|A_{ij}| \quad \textit{for } i \neq j \end{aligned}$$

So that Krawczyk's direct inverse solution requires only the inversion of one point matrix. Krawczyk's iteration is obtained by repeated application of the operator:

$$X_{\kappa}^{(l+1)} = \mathcal{K}(A, B, X_{\kappa}^{(l)})$$

and in the limit one obtains:

$$X_{\kappa} = \lim_{l\to\infty} X_{\kappa}^{(l)} \tag{3.16}$$

When:

$$\|A - I\|_{\infty} \leq \beta < 1$$

the quality of the above bounds can be quantified by:

$$w(X_{\sigma}) \leq w(X_{\kappa}) \leq w(X_{\kappa d}) \leq \frac{1+\beta}{1-\beta} w(X_{\sigma})$$

for every initial enclosure $X_{\kappa}^{(0)}$ of iteration (3.16) [171]. This indicates that both solutions are very good for small β. Furthermore if $X_{\kappa d} \subseteq X_{\kappa}^{(0)}$ it can be shown that $X_{\kappa} = X_{\kappa d}$. Although Krawczyk's iteration can yield better results than the direct inverse solution, the latter is usually preferred since it is more efficient.

Example 3.16

For the linear interval equations (3.12) of example 3.15, after preconditioning with the midpoint inverse, we find:

$$X_{\kappa d} = \begin{pmatrix} [-140.455\,, 167.728] \\ [-163.637\,, 267.273] \end{pmatrix}$$

and indeed:

$$X_{\kappa} = \begin{pmatrix} [-140.455\,, 167.728] \\ [-163.637\,, 267.273] \end{pmatrix}$$

This solution corresponds to the following system of equations:

$$\begin{aligned} 2.5X_1 + 0.5X_2 &= [-217.5\,, 337.5] \\ 1.5X_1 + 2.5X_2 &= [-157.5\,, 457.5] \end{aligned}$$

and is illustrated in figure 3.4. For this example $\beta = \|A_c^{-1}A - I\|_{\infty} = 0.364$ so that the overestimation bound $(1+\beta)/(1-\beta) = 6.334$ is rather conservative as compared to $w(X_{\kappa})/w(X_{\sigma}) = 1.468$.

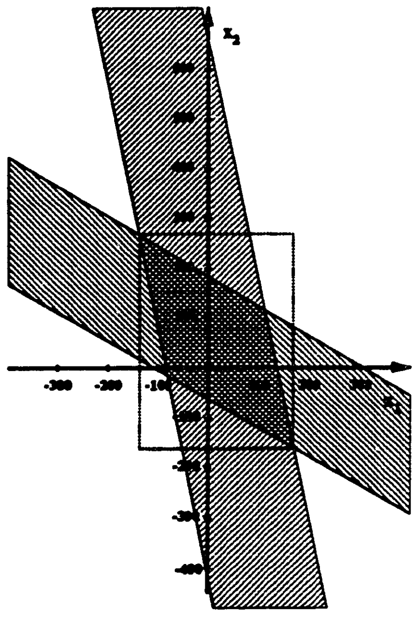


Figure 3.4: Krawczyk's solution X_κ of equations (3.12)

3.5.3 Gauss-Seidel Iteration

The Gauss-Seidel method can be extended to interval arithmetic [201]. For A and $B \in \mathbf{I}$ we first define the one dimensional solution operator over domain $X \in \mathbf{I}$:

$$\gamma(A,B,X) = \Box\{x \in X \ : \ ax = b \mid a \in A, b \in B\}$$

The corresponding explicit formula is [89]:

$$\gamma(A,B,X) = \begin{cases} B/A \cap X & \textit{if } 0 \notin A \\ \Box\{x \mid x \in X \textit{ and } (x \leq B_l/A_l \textit{ or } x \geq B_l/A_u)\} & \textit{if } B > 0 \in A \\ \Box\{x \mid x \in X \textit{ and } (x \leq B_u/A_u \textit{ or } x \geq B_u/A_l)\} & \textit{if } B < 0 \in A \\ X & \textit{if } 0 \in A, B \end{cases}$$

The multidimensional solution operator for $A \in \mathbf{I}^{n \times n}$ and $B \in \mathbf{I}^n$ can now be defined as:

$$\gamma_i(A,B,X) = \gamma(A_{ii}, B_i - \sum_{k<i} A_{ik}Y_k - \sum_{k>i} A_{ik}X_k, X_i) \tag{3.17}$$

Note that even when A is singular this operator can possibly improve the estimate. In addition, it can be proven that the Gauss-Seidel iterates *always* are better than Krawczyk's iterates [171]:

$$\gamma(A,B,X) \subseteq \kappa(A,B,X) \tag{3.18}$$

The Gauss-Seidel iteration consequently also has the quadratic approximation property. More importantly, this statement is true for a larger class of Krawczyk like iterations [171]. In particular, it shows that, as opposed to non interval equations, no improvement can be obtained by the use of so called overrelaxation techniques.

The limit of the Gauss-Seidel iteration:

$$X_\gamma^{(l+1)} = \gamma(A, B, X_\gamma^{(l)})$$

is denoted as:

$$X_\gamma = \lim_{l \to \infty} X_\gamma^{(l)}$$

When A is strongly regular X_γ is of similar quality as X_κ [171] but in view of (3.18) the Gauss-Seidel method is always preferred.

Example 3.17

The Gauss-Seidel solution X_γ of the same preconditioned system of example 3.16 is:

$$X_\gamma = \begin{pmatrix} [-130.228, 167.728] \\ [-104.416, 267.273] \end{pmatrix}$$

Clearly we have $X_\gamma \subseteq X_\kappa$.

The speed of convergence can be improved if the forward sweep (3.17) is followed by a backward sweep [2]. This is a fairly inexpensive improvement as some quantities do not need to be recomputed.

Finally, when $X_\gamma^{(l)} \subset i(X_\gamma^{(0)})$ and assuming that midpoint inverse preconditioning is used, it can be shown that A is strongly regular. Furthermore, Neumaier in [171] found that for these matrices, X_γ can be obtained in one step. As was the case for Krawczyk's direct solution (3.15), application of the Gauss-Seidel operator on initial bound (3.14) yields following explicit result:

$$X_{\gamma d_i} = \frac{b_i + (\langle A_{ii} \rangle X_{0i} - |B_i|)[-1, 1]}{A_{ii}}$$

The iterative solution process can therefore be replaced by this direct solution as soon as strong regularity is detected. On the other hand, the strong regularity result also implies that if the matrix is not strongly regular and $\kappa(A, B, X) \neq \phi$, at least one of the bounds will not be improved.

3.5.4 Gauss Elimination

Systems of linear interval equations can also be solved by Gauss elimination. Indeed, the Gauss elimination solution $e(a, b)$ can be applied on definition (3.11):

$$X_\sigma = \square\{e(a, b) \mid a \in A, b \in B\}$$

If we simply carry out the elimination step:

$$\begin{aligned} L_{ij} &= A_{ij}^{(j-1)} / A_{jj}^{(j-1)} \\ A_{ik}^{(j)} &= A_{ik}^{(j-1)} - L_{ij} A_{jk}^{(j-1)} \\ B_i^{(j)} &= B_i^{(j-1)} - L_{ij} B_j^{(j-1)} \end{aligned} \tag{3.19}$$

for i *and* $k > j$ and the backsubstitution step:

$$X_{\epsilon j} = \frac{B_j^{(j-1)} - \sum_{k>j} A_{jk}^{(j-1)} X_{\epsilon k}}{A_{jj}^{(j-1)}} \cap X_j \tag{3.20}$$

in interval arithmetic, by inclusion monotonicity we have:

$$X_\sigma \subseteq X_\epsilon$$

As shown above, if the solution is restricted to a given domain X, component wise intersection can be performed in the backsubstitution step. The corresponding solution operator is denoted by:

$$X_\epsilon = \mathcal{E}(A, B, X)$$

The Gaussian elimination algorithm can only be used when $0 \notin A_{jj}^{(j-1)}$, furthermore it can sometimes give unacceptably bad results [201]. However, when midpoint inverse preconditioning is used and A is strongly regular, it can be shown that Gauss elimination always can be carried out [3]. In addition, we have the following important result [171]:

$$\begin{aligned} X_\epsilon &\subseteq X_\gamma \\ |X_\epsilon| &= |X_\gamma| \end{aligned}$$

This means that under the above conditions, one endpoint of each of the components of X_ϵ agrees with X_γ, whereas the other endpoint may be sharper.

Example 3.18

If we apply the Gauss elimination algorithm to equations of example 3.16, we obtain:

$$X_\epsilon = \begin{pmatrix} [-130.228, 167.728] \\ [-60.000, 267.273] \end{pmatrix}$$

which is an improvement over X_γ.

It is therefore always recommended to use midpoint inverse preconditioning in conjunction with Gauss elimination, in which case Gauss elimination is the preferred solution. Unfortunately it can only be guaranteed that the algorithm can be carried out when the matrix is strongly regular.

3.5.5 Other Methods

In [6] Alefeld focuses on the 'Single Step Method with Intersection after Every Component' (SIC). Gauss-Seidel iterates are very similar to SIC iterates, but in addition $X_i^{(l+1)}$ is solved for in every step. They will therefore always produce sharper results. Note that the symmetric version of SIC finds its counterpart in the application of Gauss-Seidel in forward and in backward sweeps.

An interval version of the Gauss-Jordan method was proposed by Hansen in [85, 90]. This method is however less efficient then Gauss elimination and the generalization of (3.18) by [171] proves that it never produces bounds that are sharper than Gauss-Seidel.

A different approach was adopted by Hansen in [86]. The proposed method uses the signs of $\partial x_r/\partial a_{ij}$ and $\partial x_r/\partial b_i$ to determine which interval bounds of A_{ij} and B_i correspond with the bounds of X_σ. This method is however computationally expensive and relies on other solution methods when the signs of the partial derivatives can not be determined.

More recently, methods have been proposed to find the exact solution X_σ. In [203, 204], Rohn shows that if $\beta = \|A_c^{-1}A - I\|_\infty \ll 1$ and A is inverse stable, i.e. $0 \not\in (A^{-1})_{jj}$, this can be done in $\mathcal{O}(N^3)$ operations. When $\beta < 1$ becomes larger, the complexity increases to $\mathcal{O}(N^4)$. Finally, when A is not strongly regular and $\beta \geq 1$ the worst case complexity is often exponential in N.

3.6 Solution of Nonlinear Equations

Most of the solution techniques for solving nonlinear algebraic equations are obtained by extending classical numerical algorithms. A good overview of these techniques can be found in [266].

3.6.1 Newton Interval Method

One of the most effective techniques is to extend Newton's method with intervals [5, 7]. Figure 3.5 illustrates the 1–dimensional case. The classical Newton method is locally ap-

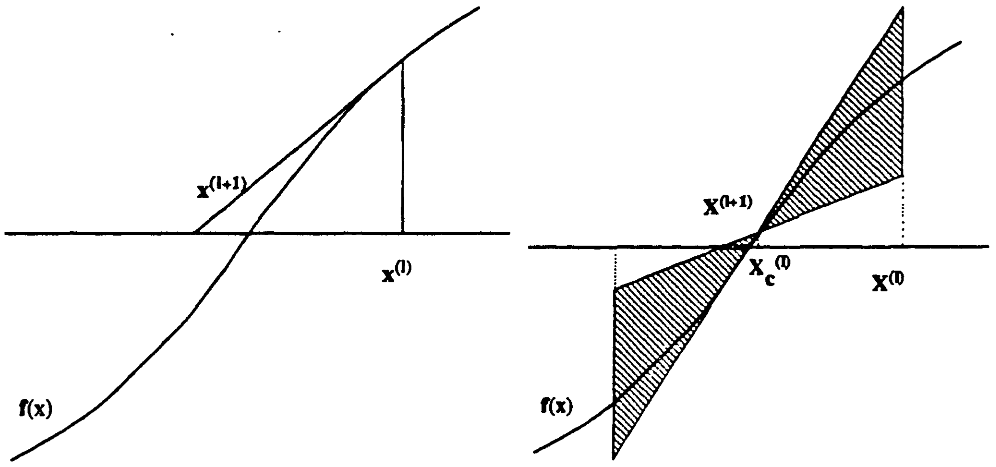


Figure 3.5: Interval extension of Newton's method

proximates the function near the current estimate $x^{(l)}$:

$$f(x) \approx f(x^{(l)}) + f'(x^{(l)})(x - x^{(l)}) \tag{3.21}$$

here superscripts indicate iteration numbers. The solution of $f(x) = 0$ is then approximated with the solution of this linear equation, which is used as next iterate:

$$x^{(l+1)} = x^{(l)} - \frac{f(x^{(l)})}{f'(x^{(l)})} \tag{3.22}$$

The interval extension of this method uses the mean value form instead of approximation (3.21):

$$f(X) \subseteq f(X_c^{(l)}) + f'(X^{(l)})\,(X - X_c^{(l)}) \tag{3.23}$$

to obtain an enclosure of the function range, and uses the corresponding interval solution as next iterate to bound the solution domain:

$$X^{(l+1)} = X_c^{(l)} - \frac{f(X_c^{(l)})}{f'(X^{(l)})} \tag{3.24}$$

Starting from the function value at the midpoint of the interval, the bounds on the derivatives limit the function to lie within the shaded area. This means that any solution has to belong to the intersection of this area with the zero-axis, which becomes the solution interval for the next iteration. As mentioned earlier, in the case of finite precision computations, the last significant digit is always rounded conservatively, i.e. downwards for the lower bound and upwards for the upper bound.

For a system of nonlinear equations $X \in \mathbf{I}^n$ and the interval derivative in equation (3.23) is replaced by an interval Jacobian $J(X) \in \mathbf{I}^{n\times n}$:

$$f(X_c^{(l)}) + J(X^{(l)})\,(X - X_c^{(l)}) \doteq 0 \tag{3.25}$$

As was originally discussed in [161], a solution to this interval equation can be used as next iterate. The Newton operator uses the exact solution of (3.25) and is defined as:

$$N(f, X) = X_c - \Sigma(J(X), f(X_c), \Delta X) \tag{3.26}$$

the corresponding Newton iterates are therefore obtained by:

$$X^{(l+1)} = N(f, X^{(l)})$$

The quadratic convergence of these iterates:

$$\exists Q \in \mathbf{R} \,:\, w(X^{(l+1)}) \leq Q w(X^{(l)})^2$$

is well known, and already dates back to Bartle [15].

We illustrate the approximation introduced by the Newton operator with a simple example.

Example 3.19

The system of equations:

$$\begin{array}{rcl} x_1^2 + x_2^2 - 1 & = & 0 \\ (x_1 - 1)^2 + x_2^2 - 1 & = & 0 \end{array} \tag{3.27}$$

has two solutions $(1/2 \;\; \sqrt{3}/2)^T$ and $(1/2 \;\; -\sqrt{3}/2)^T$. With initial-domain:

$$X^{(0)} = \begin{pmatrix} [0.2\,,\,0.8] \\ [0.6\,,\,0.9] \end{pmatrix}$$

equation (3.25) gives the following linear interval approximation:

$$\begin{aligned}[0.4\,,1.6]\Delta X_1+[1.2\,,1.8]\Delta X_2 &= [0.1875]\\ [-1.6\,,-0.4]\Delta X_1+[1.2\,,1.8]\Delta X_2 &= [0.1875]\end{aligned}\tag{3.28}$$

This is illustrated in figure 3.6. The shaded areas represent the feasible domains for each

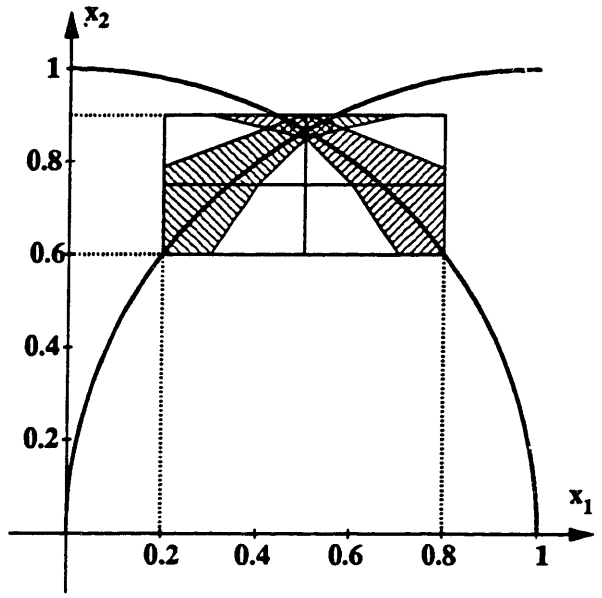


Figure 3.6: Interval Newton approximation for example 3.19

of the functions, and their intersection corresponds with the exact solution of (3.28). The first Newton iterate is:

$$X^{(1)}=\begin{pmatrix}[0.406\,,0.594]\\ [0.854\,,0.900]\end{pmatrix}$$

Note that for this first iteration the exact solution of (3.28) is not contained in $X^{(0)}$.

To achieve a 10 digit accuracy, the remaining Newton iterates are:

```
iteration              X                                    Y
    0            [+0.2v0 +0.8v0]                     [+0.6v0 +0.9v0]
    1            [+0.4v0 +0.6v0]                     [+0.85v0 +0.90v0]
    2          [+0.499v0 +0.501v0]                 [+0.8658v0 +0.8664v0]
    3      [+0.4999999v0 +0.5000001v0]       [+0.86602538v0 +0.86602544v0]
    4  [+0.4999999999v0 +0.5000000001v0]  [+0.8660254037v0 +0.8660254038v0]
```

Linear interval equation (3.25) can be solved using the solution methods discussed in section 3.5. In particular, if for each iterate the solution of (3.25) is approximated by a single application of a linear operator, we obtain the following solution operators.

3.6.2 Krawczyk's Operator

Application of Krawczyk's linear operator (3.13) to solve (3.25) leads to the corresponding nonlinear krawczyk operator:

$$K(f,X) = X_c - \mathcal{K}(J(X), f(X_c), \Delta X) \quad (3.29)$$

Krawczyk's operator was originally proposed in [126].

Figure 3.7 shows the geometric interpretation of the corresponding 1–dimensional Newton interval method. Instead of finding the exact solution, Krawczyk's operator intersects

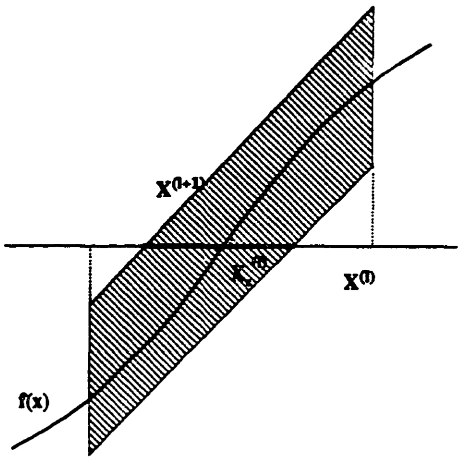


Figure 3.7: Geometric interpretation of the Krawczyk's operator

more conservative, but less complicated boundaries. Indeed, for a system of equations, the complexity of the preconditioned Krawczyk operator is only $\mathcal{O}(N^3)$, as the matrix to be inverted is not an interval matrix.

Repeated application of Krawczyk's operator leads to the iterates:

$$X^{(l+1)} = K(f, X^{(l)})$$

Example 3.20

If we apply Krawczyk's operator on the same initial box of example 3.19, we obtain the following equations:

$$\begin{array}{rcl} \Delta X_1 + 1.5\Delta X_2 & = & [-0.038\,, 0.413] \\ -\Delta X_1 + 1.5\Delta X_2 & = & [-0.038\,, 0.413] \end{array} \quad (3.30)$$

with solution:

$$X^{(1)} = \begin{pmatrix} [0.275\,, 0.725] \\ [0.725\,, 0.900] \end{pmatrix}$$

This is illustrated in Figure 3.8.

The following iterates are:

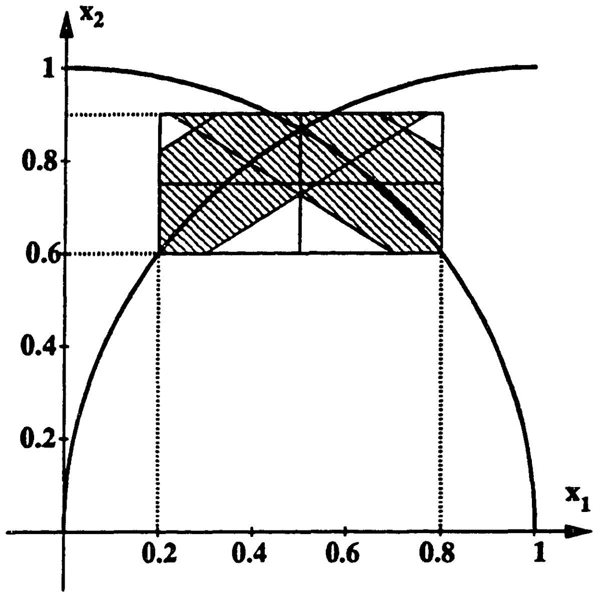


Figure 3.8: Krawczyk's approximation for example 3.19

iteration	X	Y
0	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	[+0.2v0 +0.8v0]	[+0.7v0 +0.9v0]
2	[+0.3v0 +0.7v0]	[+0.7v0 +0.9v0]
3	[+0.4v0 +0.6v0]	[+0.8v0 +0.9v0]
4	[+0.47v0 +0.53v0]	[+0.85v0 +0.89v0]
5	[+0.497v0 +0.503v0]	[+0.864v0 +0.868v0]
6	[+0.49997v0 +0.50003v0]	[+0.86601v0 +0.86605v0]
7	[+0.499999997v0 +0.500000003v0]	[+0.866025402v0 +0.866025406v0]
8	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

Moore [163] proposed a small modification to Krawczyk's iterative solution and proved superlinear convergence with fixed interval Jacobian. For variable Jacobians we have quadratic convergence, as shown in [166]. Alefeld [7] discovered that Gauss elimination could replace the inversion of the preconditioning matrix by modifying (3.29):

$$A(f,X) = X_c - e(J(X)_c, f(X_c) + \Delta J(X)\Delta X, \Delta X) \tag{3.31}$$

Although this *Alefeld Operator* is more efficient, Gauss elimination is performed with an interval right hand side. Therefore, by subdistributivity, this modification can produce wider intervals than (3.29):

$$K(f,X) \subseteq A(f,X)$$

Note also that in (3.31) intersection with the original domain ΔX can be performed during the backsubstitution step.

3.6.3 Hansen-Sengupta Operator

The Hansen-Sengupta operator is the non-linear version of the Gauss-Seidel operator for linear systems [89]:

$$H(f,X) = X_c - \gamma(J(X), f(X_c), \Delta X) \tag{3.32}$$

Since linear Gauss-Seidel iterates are better than Krawczyk iterates (3.18), we have:

$$H(f,X) \subseteq K(f,X)$$

The Hansen-Sengupta operator is therefore always preferred to Krawczyk's operator, as the following example shows.

Example 3.21

In the same conditions as example 3.20 the first Hansen-Sengupta iterate:

$$X^{(1)} = \begin{pmatrix} [0.387\,,0.613] \\ [0.816\,,0.900] \end{pmatrix}$$

is better as the first Krawczyk iterate and the convergence is faster:

```
iteration              X                                  Y
    0          [+0.2v0  +0.8v0]                  [+0.6v0  +0.9v0]
    1          [+0.3v0  +0.7v0]                 [+0.81v0  +0.90v0]
    2         [+0.49v0  +0.51v0]               [+0.864v0  +0.869v0]
    3       [+0.49998v0  +0.50002v0]        [+0.866024v0  +0.866028v0]
    4  [+0.4999999999v0  +0.5000000001v0] [+0.8660254037v0  +0.8660254038v0]
```

A symmetric version of the Hansen-Sengupta operator is proposed in [222].

3.6.4 Gauss-Newton Operator

In [217], Schwandt first introduced an operator that uses Gauss elimination to solve (3.25), the *Gauss-Newton operator*:

$$G(f,X) = X_c - \varepsilon(J(X), f(X_c), \Delta X) \tag{3.33}$$

Alefeld proves its quadratic convergence and discusses the conditions under which the Gauss-Newton iterates converge [5].

Example 3.22

If we apply the Gauss-Newton operator on equations (3.27), we obtain:

$$X^{(1)} = \begin{pmatrix} [0.387\,,0.613] \\ [0.833\,,0.900] \end{pmatrix}$$

and

iteration	X	Y
0	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	[+0.3v0 +0.7v0]	[+0.83v0 +0.90v0]
2	[+0.4998v0 +0.5002v0]	[+0.8659v0 +0.8662v0]
3	[+0.49999999v0 +0.50000001v0]	[+0.86602539v0 +0.86602541v0]
4	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

which is an improvement over the Hansen-Sengupta operator.

3.6.5 Other Newton Operators

Various hybrid approaches are possible, see for example [88] for a good combination of Gauss-Seidel iterations with Gauss elimination.

Shearer and Wolfe improve the convergence of Krawczyk's method by performing a number of inner iterations with the same interval Jacobian at each step [265]. In [224] they describe a method where the number of inner iterations are dermined automatically and apply it to the Alefeld operator. Inner iterations for the Hansen-Sengupta operator are discussed [221].

Neumaier in [169] shows how quadratic convergence can be maintained with a fixed interval Jacobian, when point Newton iterates are performed on the center.

Another class of Newton operators relies on inclusion of the inverse Jacobian. These operators are discussed in [6]. A very elegant operator is found in [6][Chapter 20]. Here iterates for inclusion of the inverse Jacobian are performed at the same time as the corresponding Newton iterates, while maintaining quadratic convergence.

3.6.6 Interval Slopes Operator

Krawczyk's interval slopes can be used to bound the range of functions. Similarly to Newton's method this set of equations can be inverted to obtain a bound on the solution [87, 134]:

$$f(X_c^{(l)}) + S(f, X^{(l)}, X_c^{(l)})(X - X_c^{(l)}) = 0 \tag{3.34}$$

As interval slopes produce sharper enclosures then derivatives (3.7), better solution bounds are obtained. The iterates are however not necessarily enclosed by the corresponding Newton iterates, as the following example shows:

Example 3.23

Using Krawczyk's slope evaluator described in section 3.3.7 and the Gauss-Newton operator we obtained the following results for (3.27):

$$X^{(1)} = \begin{pmatrix} [0.468\,, 0.532] \\ [0.859\,, 0.896] \end{pmatrix}$$

iteration	X	Y
0	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	[+0.46v0 +0.54v0]	[+0.85v0 +0.90v0]
2	[+0.4997v0 +0.5003v0]	[+0.8659v0 +0.8663v0]
3	[+0.49999998v0 +0.50000002v0]	[+0.86602539v0 +0.86602542v0]
4	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

These iterates are clearly better then the corresponding Newton iterates.

A drawback of this approach is that the slopes inclusions depend on the center of the interval equation (3.34). It is therefore not possible to improve the convergence by corresponding point iterations for the center. Furthermore the uniqueness result associated with Newton's method does not apply for interval slopes, as is discussed below.

3.7 Existence and Uniqueness Tests

If f is Lipschitz continuous, the interval operators $I(f, X)$ discussed above are conservative:

$$x \in X, f(x) = 0 \Rightarrow x \in I(f, X) \tag{3.35}$$

and as a direct consequence, we have:

$$I(f, X) \cap X = \phi \Rightarrow \forall x \in X \ : \ f(x) \neq 0 \tag{3.36}$$

which allows us to check for the existence of solution in a given domain. Furthermore it can be shown that if there are no solutions in X either (3.36) will hold after a finite number of iterations or $I(f, X) = X \Rightarrow f(X) = 0$. This property is called *Strong Convergence.* Uniqueness of a solution can be determined by:

$$I(f, X) \subseteq i(X) \Rightarrow \exists! x \in X \ : \ f(x) = 0 \tag{3.37}$$

For Newton's operator the existence test is due to Moore [162], whereas the uniqueness test was discovered by Nickel [172]. The strong convergence result is due to Neumaier [169]. Similar results hold for Krawczyk's operator [116, 126, 128, 163]. In case of the Hansen-Sengupta operator existence was proven by Hansen [89], uniqueness by Moore [167] and Qi [189], and strong convergence by Alefeld [4]. Existence and uniqueness tests for the symmetric Hansen-Sengupta operator are given by Shearer and Wolfe [223]. As these operators produce increasingly better enclosures these tests are also increasinlgy sharper. Similar results hold for the slope solution operator [170]. Unfortunately the uniqueness result is not valid, instead we have:

$$I(f, X) \subseteq i(X) \Rightarrow \exists x \in X \ : \ f(x) = 0 \tag{3.38}$$

3.8 Initial Domains

The convergence properties of the interval Newton method described above are only valid asymptotically. Indeed, in general there is no guarantee that the sequence of intervals generated by the Krawczyk's operator (3.29) will converge at all.

This implies that any practical implementation requires methods to find appropriate initial domains. This problem is very similar to the one of finding good starting points for iterative solution techniques. In order to ensure 100% reliability, it is however required that that *all* possible solution domains are found. The classic approach is to recursively bisect the original domain in the direction corresponding with the largest interval [118, 166], while attempting to eliminate these newly created subdomains. Testing if a domain could contain a solution usually is done using an interval function evaluation, i.e. $0 \in F(X^{(l)})$, or by using one of the above existence tests. Unfortunately, the subdivision technique can very quickly generate an exponential number of candidate domains when the starting domain is large.

This problem has received only very little attention in the literature. It is only recently that researchers actively started investigating optimal subdivision strategies. Their approach is to bisect in the direction that maximizes the chance of eliminating new candidate domains [195]. Although the fundamental complexity of the global solution of nonlinear algebraic equations can not be reduced, results to date indicate that significant improvements can be made over existing subdivision techniques.

Example 3.24

To illustrate the amount of work involved we apply this approach to equation (3.27) with:

$$X^{(0)} = \begin{pmatrix} [-2, 3] \\ [-2, 2] \end{pmatrix}$$

and exact values of $f(X)$ and $J(X)$. The bisections are illustrated in figure 3.9 We used

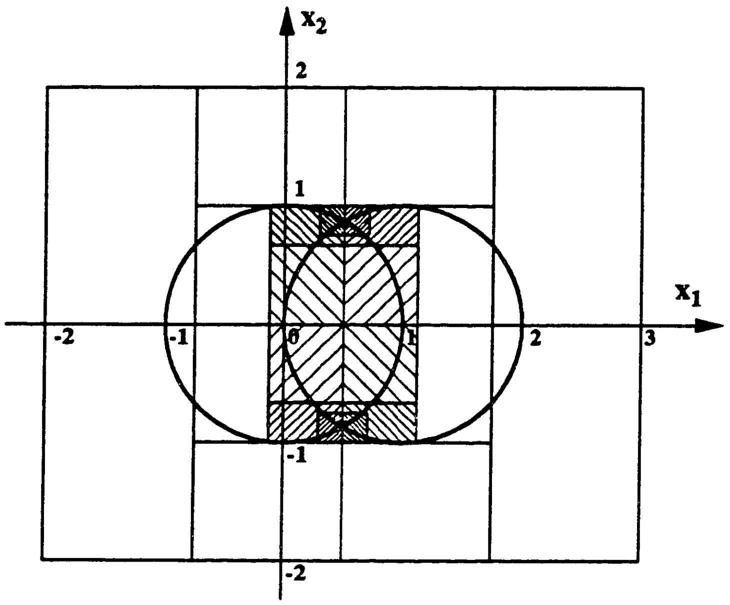


Figure 3.9: Bisections for example 3.24

the Gauss-Seidel operator and resorted to bisection when width reduction was smaller than 90%. The shaded areas indicate the converging Gauss-Seidel iterations.

The initial domains for typical design problems however, are usually not very large. We therefore have decided on the one hand to improve accuracy and speed of interval enclosure methods, and on the other to address the consistency problem.

Chapter 4

On Solving Midpoint Preconditioned Linear Interval Equations

4.1 Introduction

In this chapter we present a number of new results on the solution of midpoint preconditioned linear interval equations. A geometric interpretation of the midpoint preconditioning operation and the different solution algorithms is presented in section 4.2. In section 4.3 we propose a modification of the Gauss elimination solution technique. This method never fails, uses current domain information and *always* produces better enclosures then the Gauss-Seidel method. In section 4.4, we describe an $\mathcal{O}(N^3)$ algorithm to find exact solutions in case of regular matrices. This algorithm is modified in section 4.5 to find good enclosures when an initial domain is considered, including the case of singular matrices.

4.2 Geometric Interpretation of the Solution of Linear Interval Equations

4.2.1 Introduction

To give a geometric interpretation of both the Gauss-Seidel and Gauss elimination methods, we will again use example 3.15 borrowed from [86]. After midpoint preconditioning equations (3.12) become:

$$\begin{aligned} [0.7272\,, 1.2728]X_1 + [-0.2728\,, 0.2728]X_2 &= [-21.819\,, 49.091] \\ [-0.3637\,, 0.3637]X_1 + [0.6363\,, 1.3637]X_2 &= [-5.4546\,, 109.091] \end{aligned} \tag{4.1}$$

The effect of this operation is shown in figure 4.1. As is the case here, when $B_c \neq 0$ some overestimation can occur. However, when $B_c = 0$ no accuracy will be lost by this operation. So. as far as solving linear interval equations is concerned, it therefore pays to find a good center.

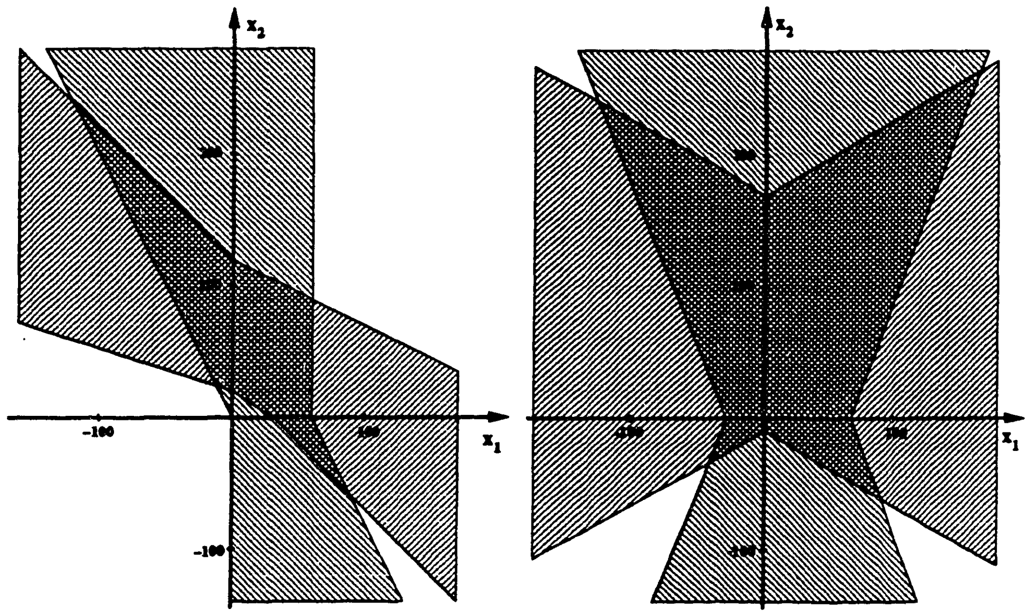


Figure 4.1: Midpoint Preconditioning of Linear Interval Equations

4.2.2 Gauss-Seidel Method

For equation (3.12), with initial domain:

$$X^{(0)} = \begin{pmatrix} [-200\,,\,200] \\ [-200\,,\,300] \end{pmatrix}$$

the Gauss-Seidel method is illustrated in figure 4.2. Using $X_2 = [-200, 300]$ the first equation is solved for X_1, resulting in $X_1 = [-142.5\,,\,180.0]$. With these bounds one then solves the second equation for X_2. The Gauss-Seidel solution therefore is:

$$X_\gamma^{(1)} = \begin{pmatrix} [-142.5\,,\,180.0] \\ [-111.429\,,\,274.286] \end{pmatrix}$$

Since we have applied the Gauss-Seidel operator only once, these results are not as good as the Gauss-Seidel fixed point solution X_γ.

Note that the geometric interpretation presented in this section is also valid for linear interval systems, which are not midpoint preconditioned.

4.2.3 Gauss Elimination Method

The geometric interpretation of the Gauss elimination method is depicted in figure 4.3. Elimination of the first variable corresponds to a projection of the solution set onto the corresponding subspace. This —for now unknown— projection is then used in the second equation to solve for the remaining variable X_2. This variable is then backsubstituted in the first equation to find X_1. The overestimation occurred by elimination is associated with

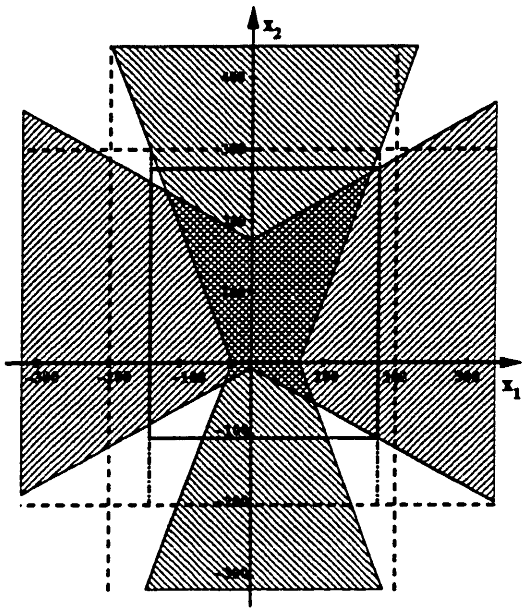


Figure 4.2: Geometric Interpretation of Gauss Seidel Solution of equations (3.12)

the projection operation.

For equation (3.12) the Gauss elimination method produces:

$$X_\varepsilon = \begin{pmatrix} [-130.228\,,\, 167.728] \\ [-60.000\,,\, 267.273] \end{pmatrix}$$

It is now clear why Gauss elimination produces better results then Gauss-Seidel on midpoint preconditioned systems, i.e. $X_\varepsilon \subseteq X_\gamma$.

The geometric interpretation presented in this section does not apply for linear interval systems which are not midpoint preconditioned. The special structure of midpoint preconditioned systems indeed ensures that the correspondence between the worst case interval bounds is not lost, which is not the case for general linear interval systems.

4.3 Modified Gauss Elimination Solution

Gauss elimination is superior to Gauss-Seidel in case of midpoint preconditioned matrices [171][Thm 4.5.11]. However, it can not be carried out on a singular matrix, in which case a zero diagonal interval element is encountered. Furthermore, it does not make good use of the initial domain. For these reasons the midpoint preconditioned Gauss-Seidel operator is usually preferred.

To overcome these shortcomings we propose the following modification to the Gauss elimination scheme. Before a variable is eliminated, the Gauss-Seidel interval of that variable is computed. If this interval is within the corresponding domain interval, the current equation is used to eliminate the variable in the remaining equations. Otherwise, the Gauss-Seidel interval is used to eliminate (i.e. substitute for) that variable. This modified

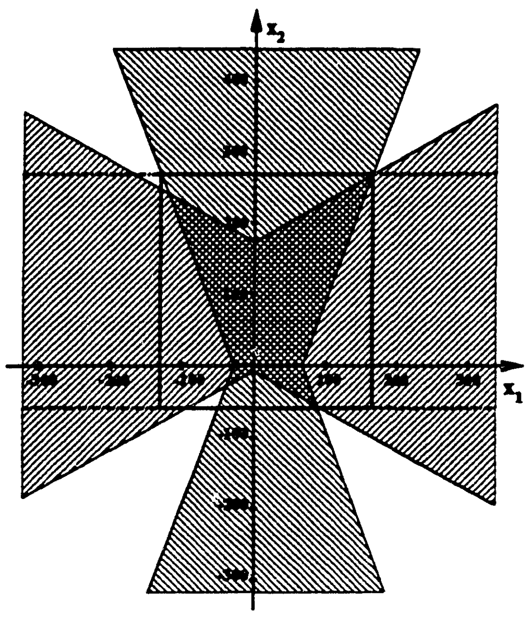


Figure 4.3: Geometric Interpretation of Gauss Elimination Solution of equations (3.12)

elimination scheme is recursively applied to the remaining equations.

This modification allows for zero interval pivots, since in that case one will automatically resort to the Gauss-Seidel solution. In fact, when the initial domain is small enough, this algorithm will use Gauss-Seidel for each variable and therefore be equivalent the Gauss-Seidel operator. On the other hand when the initial domain is large, no Gauss-Seidel interval will ever be used and the algorithm simply performs Gauss elimination.

We now show that application of the Gauss-Seidel operator on the resulting system *always* produces better results than on the original system. To do this we first show that this holds after elimination of only one variable.

Let us first introduce some notation. Consider the following linear system of interval equations:

$$\sum_{j=1}^{n} A_{ij} X_j = B_i \quad i = 1, \ldots, n \tag{4.2}$$

The Gauss-Seidel solution of this system is:

$$Y_i = \gamma(A_{ii}, B_i - \sum_{j<i} A_{ij} Y_j - \sum_{j>i} A_{ij} X_j, X_i) \tag{4.3}$$

After elimination of X_1 the Gauss-Seidel solution is the same as in (4.3) for $i = 1$ and for $i, j > 1$ we have:

$$Y_i^{(1)} = \gamma(A_{ii}^{(1)}, B_i^{(1)} - \sum_{j<i} A_{ij}^{(1)} Y_j^{(1)} - \sum_{j>i} A_{ij}^{(1)} X_j, X_i) \tag{4.4}$$

where:

$$A_{ij}^{(1)} = A_{ij} - \frac{A_{i1}A_{1j}}{A_{11}} \tag{4.5}$$

$$B_i^{(1)} = B_i - \frac{A_{i1}B_1}{A_{11}} \tag{4.6}$$

Without loss of generality, we can assume that $Y_1 \subseteq X_1$. Otherwise the proposed algorithm would simply perform one Gauss-Seidel step, and the solution of both systems would be identical.

First, we need the following two lemmas. They are not new and similar results can be found in the literature [171].

Lemma 1 *Let $A', A'', B, X \in \mathbf{I}$ then $\gamma(A' + A'', B, X) \subseteq \gamma(A', B - A''X, X)$*

Proof: If $x \in X, (a' + a'')x = b$ then $a'x = b - a''x \in (B - A''X)$ □

Lemma 2 *Let $A', A'', X \in \mathbf{I}$ and $A'_c = A''_c = 0$ then $(A' + A'')X = A'X + A''X$*

Proof: $(A' + A'')X = |(A' + A'')||X|[-1, 1] = |A'||X|[-1, 1] + |A''||X|[-1, 1] = A'X + A''X$ □

We now have the following theorem:

Theorem 1 *For a midpoint preconditioned system, using definitions (4.3) and (4.4) if $Y_1 \subseteq X_1$ we have $Y_i^{(1)} \subseteq Y_i$.*

Proof: For $i = 1$, $Y_1^{(1)}$ and Y_1 are the same and so:

$$Y_1^{(1)} \subseteq Y_1 \tag{4.7}$$

For $i, j > 1$ by applying lemma 1 to definition (4.3) and using (4.5) we have:

$$Y_i^{(1)} \subseteq \gamma(A_{ii}, B_i^{(1)} - \sum_{j<i} A_{ij}^{(1)} Y_j^{(1)} - \sum_{j>i} A_{ij}^{(1)} X_j + \frac{A_{i1}A_{1i}}{A_{11}} Y_i^{(1)}, X_i) \tag{4.8}$$

By subdistributivity and using (4.5) and (4.6):

$$Y_i^{(1)} \subseteq \gamma(A_{ii}, B_i - \sum_{j<i} A_{ij} Y_j^{(1)} - \sum_{j>i} A_{ij} X_j - (\frac{A_{i1}B_1}{A_{11}} - \sum_{j\leq i} \frac{A_{i1}A_{1j}}{A_{11}} Y_j^{(1)} - \sum_{j>i} \frac{A_{i1}A_{1j}}{A_{11}} X_j), X_i) \tag{4.9}$$

By construction we know that $Y_j^{(1)} \subseteq X_j$, so using lemma 2, we can write:

$$Y_i^{(1)} \subseteq \gamma(A_{ii}, B_i - \sum_{j<i} A_{ij} Y_j^{(1)} - \sum_{j>i} A_{ij} X_j - A_{i1} \frac{B_1 - \sum A_{1j} X_j}{A_{11}}, X_i) \tag{4.10}$$

But by assumption:

$$Y_1 = \frac{B_1 - \sum A_{1j} X_j}{A_{11}}$$

furthermore for $j < i$ we can induce that:

$$Y_j^{(1)} \subseteq Y_j$$

so that:

$$Y_i^{(1)} \subseteq \gamma(A_{ii}, B_i - \sum_{j<i} A_{ij}Y_j - \sum_{j>i} A_{ij}X_j - A_{i1}Y_1, X_i) = Y_i \tag{4.11}$$

□

Recursive application of the theorem on $A^{(1)}$ and $B^{(1)}$ shows that application of the Gauss-Seidel operator on the eliminated system is better as compared to the original one:

$$Y_i^{(n)} \subseteq Y_i$$

By inclusion monotonicity this inequality still holds when both a forward and a backward sweep of the Gauss-Seidel operator is performed. As a consequence the proposed modified Gauss operator, is better then the symmetric Gauss-Seidel operator.

Due to the proposed modification, it is not needed to perform the intersection with the original domain at backsubstitution time. The additional cost incurred by this modification is at most $\frac{N^2}{2}$, which is less than N^2 of a simple Gauss-Seidel sweep.

The algorithm is not always better then Gauss elimination as the selection of a Gauss-Seidel solution is not guaranteed, but rather likely, to be better. So there are cases where the sequential application of the Gauss-Seidel operator and the Gauss elimination operator produces a better result. The converse is also true as the elimination projections can be better when information of the initial domain is used. This gain can help reduce the exponential complexity associated with the bisection process for the solution of nonlinear equations.

4.4 An $\mathcal{O}(N^3)$ Algorithm for the Exact Solution

As illustrated in figure 4.4 the exact solution for a midpoint preconditioned system of linear interval equations (4.1) can be seen as the intersection of the point sets corresponding to n interval equations. In this section, we ignore any initial domain information and assume that the interval matrix is regular. Both assumptions will be relaxed in the next section. Without loss of generality we also assume that $B_{ic} \geq 0$. Indeed if it is not the case we simply redefine $X_i = -X_i$ and multiply the i-th equation by -1.

Furthermore, as the matrix A is midpoint preconditioned, the solution set of the i-th equation is symmetric w.r.t. the origin in the j-th coordinate direction $j \neq i$. The bounding surfaces of the i-th equation corresponding to B_l and B_u bound the lower, respectively upper values of x_i. Since by assumption $B_{ic} \geq 0$ the values of x_i on the upper surface are positive and in absolute values they are larger then the corresponding x_i on the lower bounding surface. Furthermore, the x_i increase with increasing $|x_j|$. X_u therefore corresponds with the vertex v and can be found as the intersection of the n upper surfaces:

$$\langle A \rangle v = B_u$$

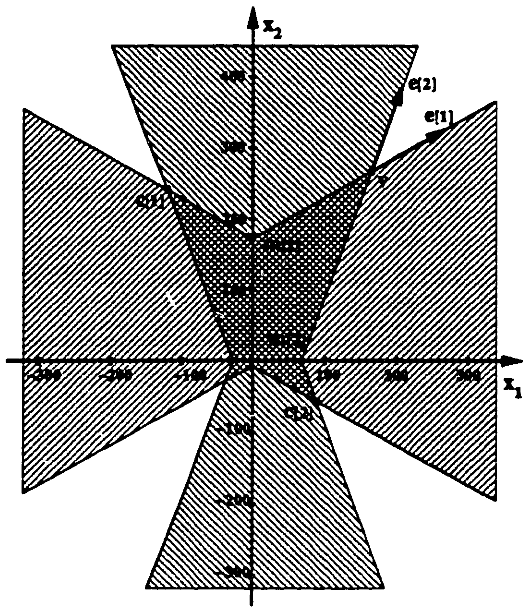


Figure 4.4: Exact Solution Algorithm for Equations (4.1)

The lower values of x_i are bounded by the lower bounding surface. In addition they decrease with increasing $|x_j|$. X_{il} can therefore be obtained at $c[i]$, the intersection of the lower bounding surface corresponding with B_{il} and the upper bounding surfaces corresponding to B_{j_u} for $j \neq i$.

Going back to figure 4.4 we now know that the vertex v produces X_u, and so:

$$v = \langle A \rangle^{-1} B_u \tag{4.12}$$

To compute the location of each of the remaining corners $c[i]$ of the hypercube-like solution, we will follow the edges of this hypercube until we reach the opposite surface. The initial directions correspond to the vectors $e[i]$ emanating from v, which are the columns of $\langle A \rangle^{-1}$. We then reach the mirror points $m[i]$ where $m[i]_i = 0$:

$$m[i] = v - e[i] \frac{v_i}{e[i]_i}$$

Let us assume for now that the lower bounding surface is entirely below the origin, so that we can follow the image of the ray past the mirror point until we reach $c[i]$. Using the mirrored direction:

$$e'[i]_j = \begin{cases} -e[i]_j & \textit{for } i = j \\ e[i]_j & \textit{for } i \neq j \end{cases}$$

$c[i]$ can be computed as the intersection of the edge with the surface:

$$\begin{aligned} \langle A \rangle_i^* c[i] - B_{il} &= 0 \\ c[i] - (m[i] + e'[i]s') &= 0 \end{aligned} \tag{4.13}$$

where $\langle A\rangle^*$ is the adjoint comparison matrix, with:

$$\begin{aligned}\langle A\rangle^*_{ii} &= \langle A_{ii}\rangle \\ \langle A\rangle^*_{ij} &= |A_{ij}| \quad \textit{for } i \neq j\end{aligned}$$

Since $c[i]$ produces the lower bound in the of i-th coordinate, we are interested only in $c[i]_i$:

$$X_{il} = c[i]_i$$

Using equations (4.13) we obtain:

$$X_{il} = \alpha_i \tag{4.14}$$

where:

$$\alpha_i = 2B_{i_c}e[i]_i - v_i \tag{4.15}$$

The derivation of this result is presented in appendix A.1.

For equations (4.1) we have:

$$\langle A\rangle^{-1} = \begin{pmatrix} 1.75 & 0.75 \\ 1 & 2 \end{pmatrix} \tag{4.16}$$

so that:

$$X_u = \langle A\rangle^{-1}B_u = \begin{pmatrix} 167.728 \\ 267.273 \end{pmatrix}$$

and:

$$X_l = \begin{pmatrix} (-21.818 + 49.091)1.75 - 167.728 \\ (-5.455 + 109.091)2 - 267.273 \end{pmatrix} = \begin{pmatrix} -120 \\ -60 \end{pmatrix}$$

As the following example shows, the lower bounding surfaces are not necessarily below the origin. A change of B in equation (3.12):

$$\begin{aligned}[2\,,3]X_1 + [0\,,1]X_2 &= [0\,,120] \\ [1\,,2]X_1 + [2\,,3]X_2 &= [200\,,240]\end{aligned} \tag{4.17}$$

produces the following midpoint preconditioned system:

$$\begin{aligned}[0.7272\,,1.2728]X_1 + [-0.2728\,,0.2728]X_2 &= [-21.819\,,36.364] \\ [-0.3637\,,0.3637]X_1 + [0.6363\,,1.3637]X_2 &= [58.182\,,109.091]\end{aligned} \tag{4.18}$$

which is shown in figure 4.5. In this case we encounter $c[2]$ before $m[2]$. This situation happens when:

$$\langle A\rangle^*_i m[i] - B_{il} < 0 \tag{4.19}$$

which, using definition (4.15) corresponds to the condition:

$$\alpha_i > 0$$

as is shown in the appendix A.1. Not only do we need to use the original direction $e[i]$ but the equation of the bounding surface also is different. In this case the corner $c[i]$ described

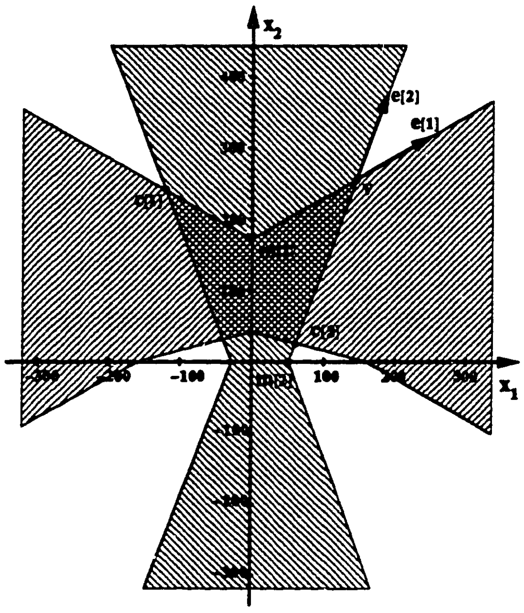


Figure 4.5: Exact Solution Algorithm for Equations (4.18)

by:

$$\begin{aligned} |A|_i c[i] - B_{il} &= 0 \\ c[i] - (m[i] + e[i]s) &= 0 \end{aligned} \tag{4.20}$$

We obtain:

$$X_{il} = c[i]_i = \frac{\alpha_i}{2e[i]_i - 1}$$

as shown in appendix A.1.

We illustrate these computations for equations (4.18).

$$X_u = \langle A \rangle^{-1} B_u = \begin{pmatrix} 145.4545 \\ 254.5454 \end{pmatrix}$$

and:

$$\alpha = \begin{pmatrix} (-21.818 + 36.364)1.75 - 145.45 \\ (58.1818 + 109.091)2 - 254.545 \end{pmatrix} = \begin{pmatrix} -120 \\ 80 \end{pmatrix}$$

while $\alpha_1 < 0$, $\alpha_2 > 0$ so that:

$$X_l = \begin{pmatrix} -120 \\ \frac{80}{4-1} \end{pmatrix} = \begin{pmatrix} -120 \\ 26.666 \end{pmatrix}$$

The complexity of this algorithm is $\mathcal{O}(N^3)$ corresponding to the inversion of $\langle A \rangle$. Indeed the computation of all the lower bounds requires $\mathcal{O}(N)$, and is very fast. Figure 4.6 illustrates this method on a three dimensional problem.

Note that up to now we have assumed $B_{i_c} > 0$. For the general case we would have to

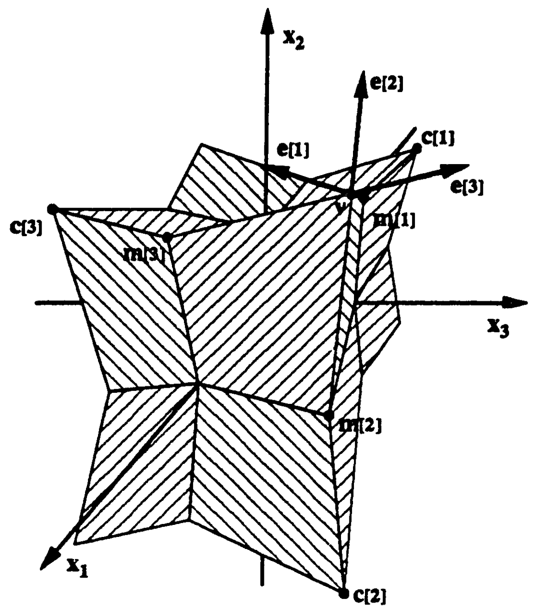


Figure 4.6: Three dimensional solution of Midpoint Preconditioned System

make a simple change to the above results. We now have:

$$X'_{iu} = v'_i = \langle A \rangle_i^{-1} |B_u|$$

Defining:

$$\alpha'_i = 2|B_{ic}|e[i]_i - v'_i$$

we have:

$$X'_{il} = \begin{cases} \alpha'_i & \textit{for } \alpha'_i \le 0 \\ \dfrac{\alpha'_i}{2e[i]_i - 1} & \textit{for } \alpha'_i > 0 \end{cases}$$

and finally:

$$X_i = sgn(B_{ic}) X'_i$$

4.5 General Solution of Midpoint Preconditioned Interval Equations

The results of the preceding section can be modified to take advantage of initial domains and to deal with singular interval matrices. Unfortunately, at the time of writing it remains an open question whether the exact intersection of the initial domain with the solution can be computed in polynomial time. So in the algorithm described below, some overestimation can occur if the interval solution domain intersects the initial domain. Indeed, as was the case in section 4.3, the complexity of this problem doesn't allow us to select the best solution method at each step. Again, we have opted to improve on the Gauss-Seidel operator, while still obtaining the exact solution for large initial domains.

We start out by taking advantage of information concerning the initial domain only, i.e. we will assume for now that A is not singular. This implies that we can perform LU-decomposition on $\langle A \rangle$ so that the linear equation for v becomes:

$$\langle A \rangle v = LUv = B_v \tag{4.21}$$

In this equation we have adopted a more general notation B_v for the bounds of B which determine v. Using forward substitution equation (4.21) can be written as:

$$Uv = RB_v + C \tag{4.22}$$

where $R = L^{-1}$ also a lower triangular matrix. For reasons given below, we have included the constant vector $C \in \mathbf{R}^n$, which is zero initially. As before we can, without loss of generality, make the assumption that $B_{ic} > 0$. We now are in a position to determine the exact bounds on the n-th variable of the solution of the linear system. Indeed, we see from equation (4.22) that:

$$e[n]_n = \langle A \rangle^{-1}_{nn} = \frac{R_{nn}}{U_{nn}}$$

furthermore:

$$v_n = \frac{R_n B_v + C_n}{U_{nn}} \tag{4.23}$$

This allows us to compute α_n using equation (4.15):

$$\begin{aligned} \alpha_n &= 2B_{nc} e[n]_n - v_n \\ &= \frac{1}{U_{nn}}(2B_{nc}R_{nn} - R_n B_v - C_n) \end{aligned}$$

and so:

$$X_{nl} = \begin{cases} \alpha_n & \text{for } \alpha_n \le 0 \\ \dfrac{\alpha_n}{2e[n]_n - 1} & \text{for } \alpha_n > 0 \end{cases} \tag{4.24}$$

We now have to decide whether to use information provided by the initial domain or by the n-th equation for the bounds on X_n in order to obtain the tightest solution. Unfortunately, this cannot always be decided perfectly, so that some overestimation can occur. We will return to this selection criterion later.

When the bounds on the exact solution are determined by the n-th equation, we simply project the solution on the corresponding subspace using backward substitution, where B_{nv} is either equal to $|B_{nl}|$ or to B_{nu}:

$$\begin{aligned} U^{(1)}_{ij} &= U_{ij} \\ R^{(1)}_{ij} &= R_{ij} - \frac{U_{in}}{U_{nn}} R_n \\ C^{(1)}_i &= C_i - \frac{U_{in}}{U_{nn}} C_n + (R_{in} - \frac{U_{in}}{U_{nn}}) B_{nv} \end{aligned}$$

for $i, j < n$.

If on the other hand the bounds of the initial domain constrain the exact solution, we proceed to use a fixed domain bound X_{nf} instead of the current equation to eliminate X_n:

$$\begin{aligned} U_{ij}^{(1)} &= U_{ij} \\ R_{ij}^{(1)} &= R_{ij} \\ C_i^{(1)} &= C_i - U_{in}X_{nf} \end{aligned}$$

for $i, j < n$.

In both cases the original system is reduced to the following system of dimension $n-1$:

$$U^{(1)}v^{(1)} = R^{(1)}B_v^{(1)} + C^{(1)} \tag{4.25}$$

where the n-th variable has been eliminated, i.e. :

$$\begin{aligned} v_i^{(1)} &= v_i \\ {B_i}_v^{(1)} &= B_{iv} \end{aligned}$$

where $i < n$.

We now apply this algorithm recursively to equation (4.25) until the solution has been bounded in all dimensions. Geometrically this can be interpreted as follows. When the n-th variable is eliminated by backsubstitution, one simply projects the exact solution of the linear interval system on the corresponding subspace. This is equivalent to the algorithm described in section 4.4 except for the occasional use of $|B_{nl}|$. The case where the variable is directly substituted for, is pictured in figure 4.7. Here the hypercube like solution is

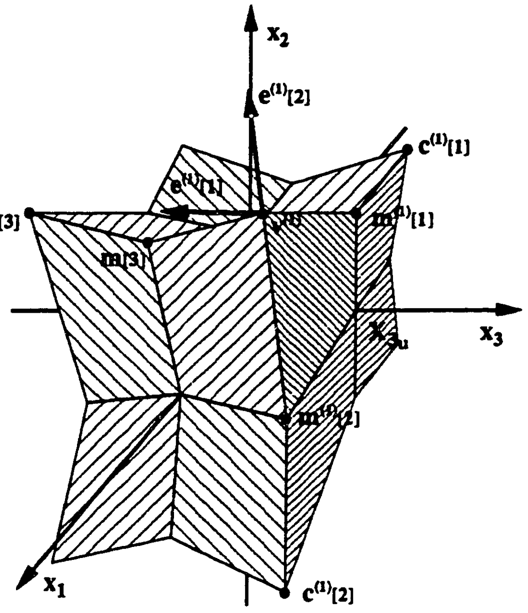


Figure 4.7: General Solution of 3 Dimensional Interval Equations

intersected with a hyperplane perpendicular to the n-th coordinate axis, located at X_{nf}. The solution in the corresponding subspace is therefore of a different shape then the one obtained by projection (see also figure 4.8).

Let us now deal with de selection criterion and the determination of the bounds of the n-th variable. We denote the bounds of X_n determined by equations (4.23) and (4.24) by $X_{n\sigma}$ and the bounds of the initial domain X_{nb}. Clearly, when:

$$X_{n\sigma u} \in X_{nb} \tag{4.26}$$

it is better to use the exact solution, i.e. to use:

$$X_n = X_{n\sigma} \cap X_{nb}$$

and perform backsubstitution using:

$$B_{nv} = B_{nu}$$

like before. If relation (4.26) does not hold, our decision will depend on the position of the intersecting hyperplane. As X_{nf} moves from $X_{n\sigma l}$ to $X_{n\sigma u}$ different solution shapes arise. This is illustrated in figure 4.8. The picture on the left shows a projection of the

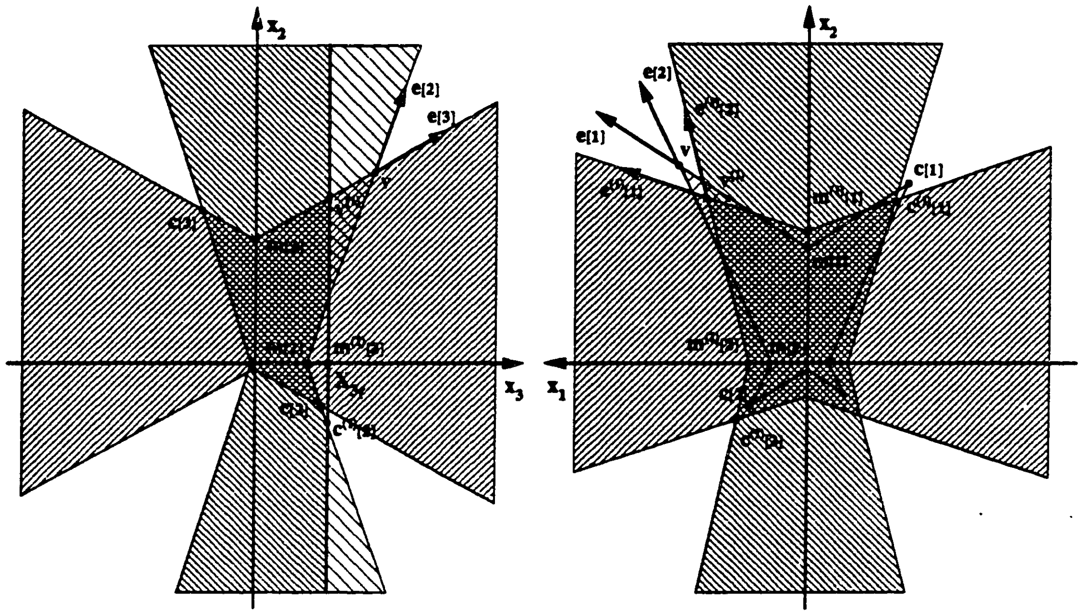


Figure 4.8: Intersection of the Exact Solution with a Hyperplane

hypercube like solution along coordinate X_1, the projection of this solution along coordinate X_3 and the shape of the intersection itself are shown on the right. Because the shape of the projection is not the same as the shape of intersection, partial overlap can occur. To obtain the exact solution we would need to use information of both shapes in further calculations. Unfortunately such a combined shape is not exactly representable within the framework of interval arithmetic, and would lead to exponential complexity.

As we indicated earlier, we set out to obtain a solution that is consistently better than the Gauss-Seidel solution. So in cases of partial overlap, we need to use the *inner* solution, corresponding with the shape of the intersection obtained by direct substitution rather than the *outer* solution corresponding with the projection obtained by backsubstitution.

With this in mind, one can verify that if:

$$X_{nb} \subseteq X_{n\sigma}$$

we should use:

$$X_n = X_{nb}$$

and perform direct substitution with:

$$X_{nf} = |X_{nb}|$$

This leaves with the situation where:

$$X_{n\sigma l} \in X_{nb}$$

Here we will use:

$$X_n = [X_{n\sigma l}\,,\, X_{nbu}]$$

but we need to consider four different cases as far as the subspace solution is concerned. First, when:

$$X_{n\sigma l} > 0$$

any intersection will contain the exact solution and we need to perform a direct substitution with:

$$X_{nf} = X_{nbu}$$

When:

$$X_{nbu} > |X_{n\sigma l}|$$

an overlap situation is possible, this is the case illustrated in figure 4.8. We have to use:

$$X_{nf} = X_{nbu}$$

and perform direct substitution. If it is the case that:

$$|X_{nbu}| \in [\beta_n\,,\, |X_{n\sigma l}|]$$

where we have defined β_n as:

$$\beta_n = \begin{cases} \dfrac{|B_{nl}|}{\langle A\rangle_{nn}} & for\ B_{nl} \leq 0 \\ \dfrac{B_{nl}}{|A|_{nn}} & for\ B_{nl} > 0 \end{cases}$$

we have no choice but to perform direct substitution with:

$$X_{nf} = |X_{n\sigma l}|$$

due to possible overlap with the projection of the opposite face of our hypercube like solution. Finally we are left with:

$$|X_{nbu}| < \beta_n$$

in which case we can safely perform backsubstitution using:

$$B_{nv} = |B_{nl}|$$

hereby projecting the face of the hypercube corresponding to B_{nl}.

The overestimation incurred by this method is due partly to the possible overestimation of the solution shape at each step, indeed depending on the bounds in the other directions it is still possible that the elimination operation produces better bounds then the substitution operation and vice versa. But also to the nature of the projection operation since the vertices of the exact solution might not belong to the projected solution.

After this analysis it is clear that the performance of Gauss-Seidel and modified Gauss elimination are not comparable, i.e. there are cases where one will be better then the other and vice versa. Due to the complexity of the exact solution, we tend to believe that there is no $\mathcal{O}(N^3)$ time algorithm that is consistently better than either of them. We could however conceive an $\mathcal{O}(N^3)$ algorithm that can be expected to yield best results *on average.* Instead of choosing direct or backward substitution, we could indeed generate an interval enclosure which minimizes the area of the solution in the current subspace.

Until now we have assumed that A is not singular. Indeed, it does not make sense to perform an LU-decomposition on $\langle A \rangle$ when A is singular; some L_{ii} would become negative. When during the LU-decomposition process negative pivots are encountered, we can simply use direct substitution for that variable, and reduce the dimension of the system by one. In this case the vector C will not be necessarily be zero at the beginning of our solution algorithm.

Geometrically, a negative pivot means that the corresponding equation does not bound the solution in that direction, so that the bounds of the initial domain need to be used.

Chapter 5

Fast Evaluation of Partial Derivatives

5.1 Introduction

Solving nonlinear systems requires evaluation of partial derivatives at every iteration and for every domain generated by the bisection strategy. We therefore believe that incorporation of efficient evaluation strategies in the compilation of partial derivatives can contribute significantly towards the reduction of the overall solution time.

We would like to note that the material proposed in this chapter is not restricted to arithmetic expressions. All algorithms apply without modification when more general operators replace the arithmetic ones, as long as local derivative information is available. In fact, they can be particularly useful for computational graphs involving vector and matrix operations.

The automatic computation of partial derivatives is usually done in a recursive fashion. As discussed in section 5.2 the backward approach can be more efficient. The problem of finding optimal evaluation strategies for partial derivatives on computational graphs is addressed in section 5.3.

5.2 Backward Evaluation Algorithm

In section 5.2.1 we describe the backward evaluation algorithm and how it can be used for interval differentiation. In section 5.2.2 we present the chain rule for interval slopes and use it to for backward evaluation in section 5.2.3.

5.2.1 Evaluation of Interval Partial Derivatives

As proposed in [190] automatic computation of partial derivatives is usually done recursively. This bottom up approach unfortunately has a complexity of $\mathcal{O}(nF)$ where n is the number of arguments and F is the complexity of one function evaluation. The forward computation of partial derivatives is not as efficient as it could be, indeed it involves incremental operations on vectors, and many of these operations would need to be performed only once per step. This consideration led researchers to the more efficient backward, top down, computation of partial derivatives [232], a strategy very similar to the backpropagation algorithm in neural networks [210]. The complexity of evaluating both the function value and the partial derivatives reduces to CF, where the constant factor $C = 3$ as shown by Baur and Strassen [16]. Note that this result assumes that all operators are at most binary. Iri [108] analyzes this algorithm from the perspective of graph theory, and shows how the value of C depends on the different measures of complexity. A comparison between the different approaches together with some practical applications of this technique can be found in [77, 78].

Let us first introduce some terminology and notation. A directed graph G is a pair (N, A), where N is a finite set of nodes and A a finite set of arcs. An arc from x to y is an ordered pair (x, y). The notation $\alpha_G^-(x)$ is used to indicate arcs incident to node x, $\alpha_G^+(x)$ for the arcs leaving node x. Accordingly, the in-degree of x is $|\alpha_G^-(x)|$, the out-degree $|\alpha_G^+(x)|$. The maximum in-degree of G is denoted by $|\alpha_G^-|$, maximum out-degree by $|\alpha_G^-|$. Neighbors of x in the negative direction are denoted by $N_G^-(x)$, by $N_G^+(x)$ in the positive direction. Similarly, $R_G^-(x)$ is the set of nodes reachable from x in the negative direction, $R_G^+(x)$ in the positive direction. Finally, the sign will be dropped when we want to indicate the union of both positive and negative contributions, for example $R_G(x) = R_G^-(x) \cup R_G^+(x)$.

We will limit ourselves here to non-recursive computations, and therefore define computational graphs, such as arithmetic expressions, as directed acyclic graphs where nodes correspond to operators $\circ$ and arcs to data dependencies. Variables are associated with the result of each operator as well as with the inputs. Although strictly speaking there is a difference between the two, it is not needed to make that distinction here.

Evaluation of an computational graph is done through a bottom up traversal, starting from the independent variables the arcs are followed in the positive direction and arithmetic operations are performed as their arguments become available. This is illustrated in figure 5.1. For interval evaluation, all operators can simply be replaced by their interval arithmetic

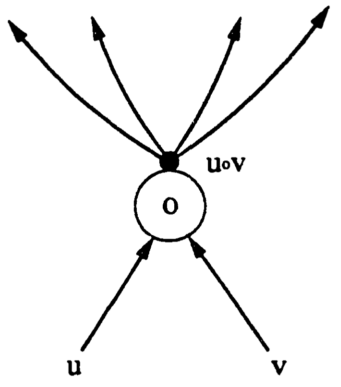


Figure 5.1: Evaluation of a Computational Graph

counterparts:

$$(u \circ v) \mapsto (U \circ V)$$

Using the chain rule of partial derivatives, backward evaluation can be expressed as follows:

$$\partial_x^f = \sum_{w \in N_G^+(x)} \partial_w^f \partial_x^w \tag{5.1}$$

The summation is performed over the nodes which directly depend on x, i.e. $N_G^+(x)$. This is illustrated in figure 5.2. First the partial derivatives of the function value f with respect

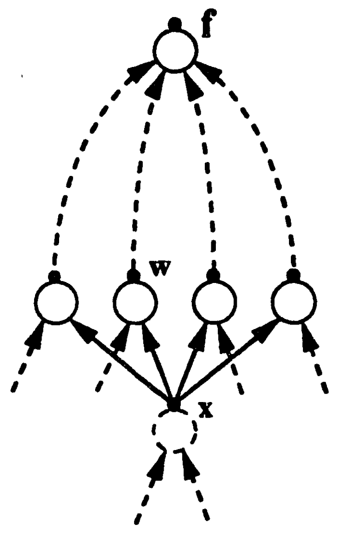


Figure 5.2: The Chain Rule for Partial Derivatives

to each node w are multiplied by the partial derivative of that node w.r.t. to its input x. The resulting contributions are then accumulated to yield the partial derivative of f w.r.t. to node x.

Equation (5.1) forms the basis of the backpropagation algorithm, which is illustrated in figure 5.3. Partial derivative contributions $(\partial^f_{u \circ v})_w$ of all $w \in N_G^+(u \circ v)$ are added when

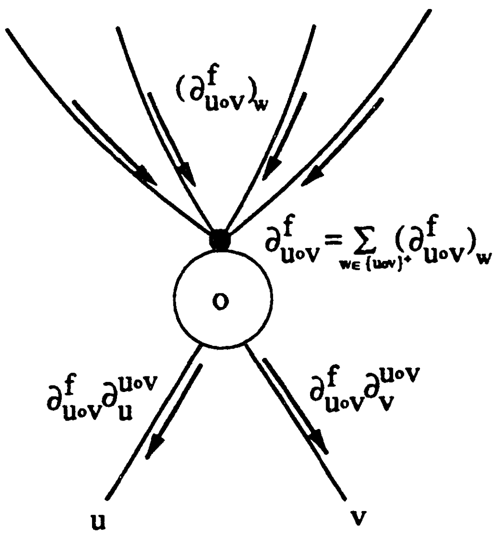

$(\partial^f_{u \circ v})_w$
$\partial^f_{u \circ v} = \sum_{w \in \{u \circ v\}^+} (\partial^f_{u \circ v})_w$
o
$\partial^f_{u \circ v} \partial^{u \circ v}_u$
$\partial^f_{u \circ v} \partial^{u \circ v}_v$
u
v

Figure 5.3: The Backward Evaluation Algorithm

node $u \circ v$ is reached. Backpropagation over that node corresponds with multiplication of the current value $\partial^f_{u \circ v}$ with the partial derivatives $\partial^{u \circ v}_u$ and $\partial^{u \circ v}_v$. The partial derivatives for $\circ \in \{+, -, *, /\}$ are:

$$\begin{aligned}
\partial_u^{u+v} &= 1 & \partial_v^{u+v} &= 1 \\
\partial_u^{u-v} &= 1 & \partial_v^{u-v} &= -1 \\
\partial_u^{u*v} &= v & \partial_v^{u*v} &= u \\
\partial_u^{u/v} &= 1/v & \partial_v^{u/v} &= -u/v^2
\end{aligned}$$

Note that it is not needed to perform two divisions when backpropagating over a division node. As shown in [108] for example, one can perform local surgery on the computational graph to avoid unnecessary operations.

The backward evaluation algorithm requires that the values at the nodes be known. Evaluation of interval partial derivatives therefore consists of two parts. First, a forward evaluation is performed to initialize all nodes, including the initialization of the function node with the interval value of the function. This step is then followed by backward evaluation of the partial derivatives from the function node f, starting with:

$$\partial_f^f = [1]$$

towards the independent variables. At this point, the partial derivative of the function w.r.t. each particular variable is available at the corresponding node.

Example 5.1

The execution of this algorithm on the expression `(+ (* (* x y) (+ y z)) x)` with $X = [2, 3]$, $Y = [4, 5]$ and $Z = [10]$ is illustrated in figure 5.4. First interval evaluation is performed, as shown on the left. It is followed by backward evaluation of the partial derivatives, depicted on the right.

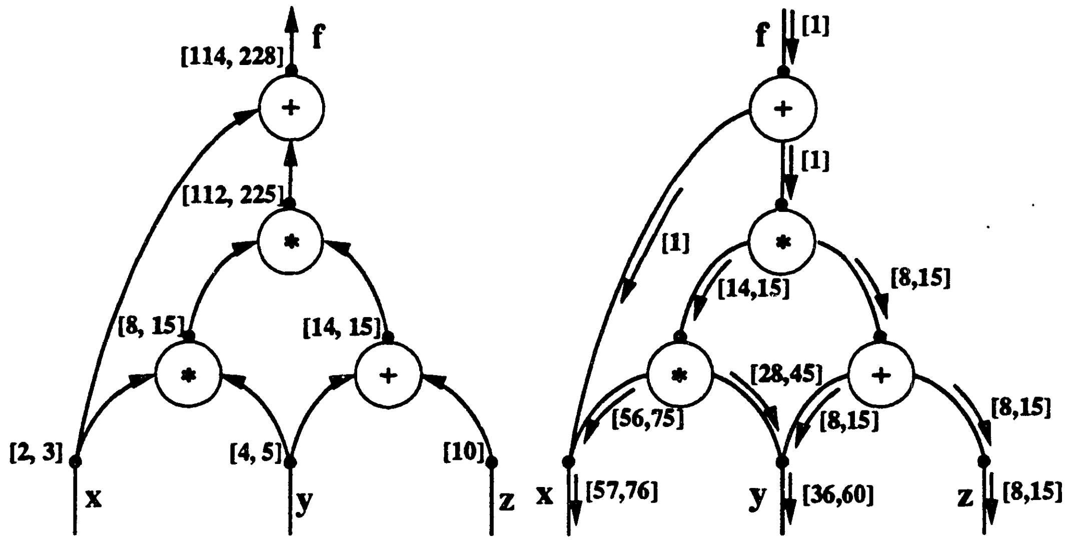


Figure 5.4: Execution of the backward evaluation algorithm on example 5.1

Finally, we would like to observe that syntactic transformation of the compiled expressions can be used to avoid unnecessary multiplications or summations involving the constants 0 and 1, and single element summations.

Example 5.2

Compilation of the function:

```
    (lambda (x y z) (+ (* (* x y) (+ y z)) x))
```

including the optimizing syntactic transformation, could for example produce the following `lambda`-expression:

```
    (lambda (x y z)
      (let* ((v1 (* x y))
             (v2 (+ y z))
             (v3 (* v1 v2))
             (f (+ v3 x))
             (dfx-v1 (* v2 y))
             (dfx (+ dfx-v1 1))
             (dfy-v1 (* v2 x))
             (dfy (+ dfy-v1 v1)))
        (list f dfx dfy v1)))
```

which return a list of the function value and partial derivatives with respect to its arguments.

5.2.2 The Chain Rule for Interval Slopes

As the original definition by Krawczyk [134, 170] and the inclusion algebra of [171] suggests, slopes are to be computed recursively. In this section we establish an equivalent to the chain rule of partial derivatives for interval slopes. This allows us to use the more efficient backward evaluation strategy described in the previous section for their computation.

Recall that the slope $S(f,X,c)$ of a function f over an interval X and center c is defined such that:

$$\begin{aligned} f(x)-f(c) &= S(f,x,c)(x-c) \\ &\in S(f,X,c)(x-c) \end{aligned}$$

To simplify some of the notation below we will write $S(f,X,c)=S^f_X$, leaving the dependence on c implicit.

We now show that in general, taking into account the above definitions, we can write:

$$(u\circ v)-(U_c\circ V_c)\in S^{u\circ v}_U(u-U_c)+S^{u\circ v}_V(v-V_c) \tag{5.2}$$

where $\circ\in\{+,-,\cdot,/\}$. The validity of equation (5.2) is shown by explicitly computing $(u\circ v)-(U_c\circ V_c)$ for the primitive operators.

For the addition we simply have:

$$(u+v)-(U_c+V_c)=(u-U_c)+(v-V_c)$$

which means that $S^{u+v}_U=1$ and $S^{u+v}_V=1$.

The slope computation for the subtraction is very similar:

$$(u-v)-(U_c-V_c)=(u-U_c)-(v-V_c)$$

Here $S_U^{u-v} = 1$ and $S_V^{u-v} = -1$.

For the multiplication, unlike Krawczyk's slopes evaluator, we introduce a symmetric slope:

$$\begin{aligned} uv - U_cV_c &= \frac{u+U_c}{2}(v-V_c) + \frac{v+V_c}{2}(u-U_c) \\ &\in \frac{U+U_c}{2}(v-V_c) + \frac{V+V_c}{2}(u-U_c) \end{aligned}$$

In this case $S_U^{uv} = \frac{V+V_c}{2}$ and $S_V^{uv} = \frac{U+U_c}{2}$.

And finally for division, we have:

$$\begin{aligned} \frac{u}{v} - \frac{U_c}{V_c} &= \frac{1}{v}(u - U_c - \frac{U_c}{V_c}(v-V_c)) \\ &\in \frac{1}{V}(u-U_c) - \frac{U_c}{VV_c}(v-V_c) \end{aligned}$$

The partial slope functions for the division are $S_U^{u/v} = \frac{1}{V}$ and $S_V^{u/v} = \frac{-U_c}{VV_c}$.

Having established the validity of equation (5.2) we proceed to establish the chain rule for interval slopes:

$$\begin{aligned} (u \circ v) - (U_c \circ V_c) &\in S_U^{u\circ v}(u-U_c) + S_V^{u\circ v}(v-V_c) \\ &= S_U^{u\circ v}S_x^u(x-c) + S_V^{u\circ v}S_x^v(x-c) \\ &\in S_U^{u\circ v}S_X^u(x-c) + S_V^{u\circ v}S_X^v(x-c) \end{aligned}$$

So by definition:

$$S_X^{u\circ v} = S_V^{u\circ v}S_X^v + S_U^{u\circ v}S_X^u \tag{5.3}$$

which is the equivalent to the chain rule of partial differentiation.

The rule applies for elementary functions in exactly the same way. For example, for square roots we find:

$$\sqrt{f} - \sqrt{F_c} \in \frac{1}{\sqrt{F_c}+\sqrt{F}}(f - F_c)$$

so that $S_F^{\sqrt{f}} = \frac{1}{\sqrt{F_c}+\sqrt{F}}$

5.2.3 Evaluation of Interval Slopes

We are now in a position to discuss the backward evaluation of interval slopes. As was the case with the computation of partial derivatives, we divide the computation up in two parts. During forward evaluation, illustrated in figure 5.5, we now not only compute the range $U \circ V$, but also the centers $U_c \circ V_c$, including the computation of the function range and function center. This is followed by the backward evaluation of interval slopes, based on:

$$S_X^f = \sum_{y \in N_G^+(x)} S_Y^f S_X^y \tag{5.4}$$

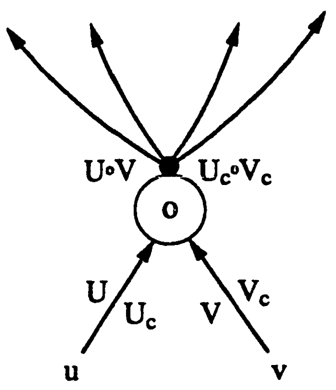
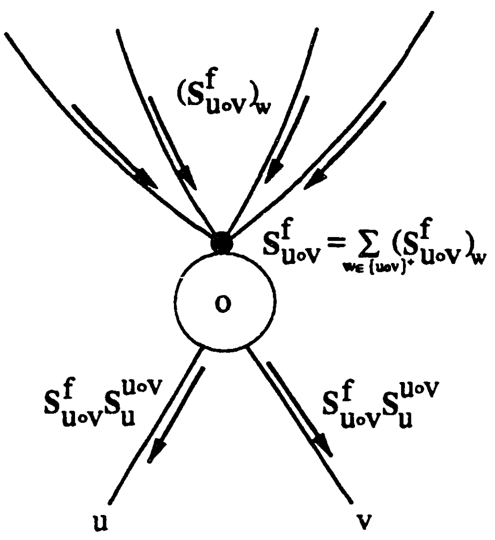


Figure 5.5: Interval and Center Evaluation

which is the slope equivalent of equation (5.2). The resulting backpropagation algorithm is shown in figure 5.6. The initial value to be propagated from the function node is:



$(S^f_{u\circ v})_w$
$S^f_{u\circ v} = \sum_{w\in\{u\circ v\}^+} (S^f_{u\circ v})_w$
o
$S^f_{u\circ v} S^{u\circ v}_u$
$S^f_{u\circ v} S^{u\circ v}_u$
u
v

Figure 5.6: Backward Evaluation of Slopes

$$S^f_F = [1]$$

After completion of the algorithm the nodes corresponding to the independent variables will contain the interval slope of the function w.r.t. to that particular variable.

The resulting algorithm also has a complexity of $\mathcal{O}(F)$, assuming that all operators are at most binary. Although the constant will be larger, in addition to the interval slopes both center and interval values of the function are computed. In fact, the operation counts are similar to the backward evaluation of interval partial derivatives, including the interval evaluation of the function, combined with a separate evaluation for the center.

Example 5.3

In figure 5.7 we illustrate the execution of the algorithm on the same expression as in example 5.1. Center and interval evaluation are shown on the left, the backward evaluation of the interval slopes is illustrated on the right.

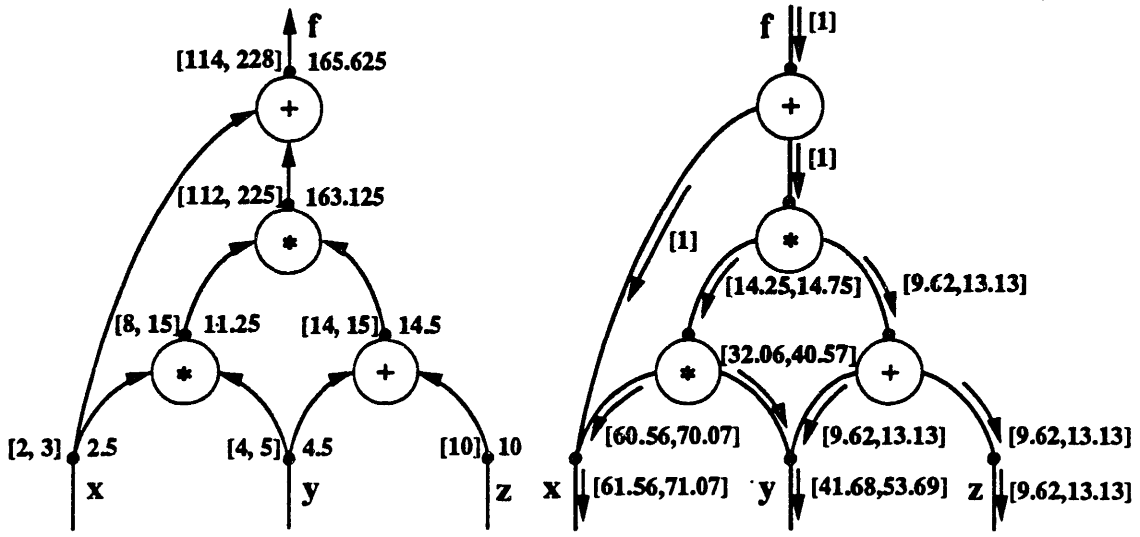


Figure 5.7: Execution of the backward slope evaluation algorithm on example 5.3

Example 5.4

Compilation of the function of example 5.2 now produces:

```
(lambda (x xc y yc z zc)
  (let* ((v1 (* x y))
         (v1c (* xc yc))
         (v2 (+ y z))
         (v2c (+ yc zc))
         (v3 (* v1 v2))
         (v3c (* v1c v2c))
         (f (+ v3 x))
         (fc (+ v3c xc))
         (sfv1 (/ (+ v2 v2c) 2))
         (sfv2 (/ (+ v1 v1c) 2))
         (sv1x (/ (+ y yc) 2))
         (sv1y (/ (+ x xc) 2))
         (sfx-v1 (* sfv1 sv1x))
         (sfy-v1 (* sfv1 sv1y))
         (sfx (+ sfx-v1 1))
         (sfy (+ sfy-v1 sfv2)))
       (list fc f sfx sfy sfv2)))
```

which return a list of the function center, function value and interval slopes with respect to its arguments.

In this case additional syntactic transformation can sometimes be performed. Indeed, in addition to the traditional optimizations of multiplications with narrow intervals, some operations on their centerpoint can be redundant.

Note also that when slopes are evaluated in a backward fashion, it is not possible to improve the range by intermediate intersections with the slope centered form, as was the case in the recursive evaluation. This is because the required slope information is not available when the range is computed, and in fact never will be computed explicitly.

5.3 Partial Derivatives of Computational Graphs

Backward evaluation can provide a significant speedup for the computation of a single function with many independent variables. As more functions are computed by a single computational graph the speedup decreases, because each function requires its backward evaluation of derivatives. When many functions are provided by a common computational graph, but there is only one variable, forward evaluation is generally more efficient than backward evaluation. Indeed, in this case the situation is reversed! In this section we set out to find optimal evaluation strategies for partial derivatives.

In section 5.3.1 we discuss how this problem is related the solution of sparse linear systems. We then proceed to present a node condensation model for the computation of partial derivatives in section 5.3.2. In section 5.3.3 we describe a technique to compute lower bounds on multiplication counts and in section 5.3.4 we propose algorithms for finding the optimal condensation order. Finally, in section 5.3.5 we present a number of heuristic algorithms for determining good condensation orders.

5.3.1 Solution Methods for Sparse Matrices

The computation of partial derivatives for the solution of nonlinear systems can be simplified by introducing intermediate variables for each operator. When the operators are of bounded degree the corresponding jacobian is sparse. The increase in the dimension can be offset by the taking advantage of the sparsity of the matrix. As this problem has been studied extensively in the literature, we first review some of the existing techniques.

In [183] Parter uses a vertex condensation model for undirected graphs to analyze variable elimination in sparse linear systems. He discusses various types of matrices and shows that for matrices representing trees, a condensation order can be found that preserves the sparseness of the matrix, i.e. no non zero elements are introduced. Rose in [205] extends this result to triangulated graphs. Unsymmetric sparse matrices can be represented using directed graphs, see for example [92]. Tarjan [246] discusses Gaussian elimination on directed graphs in a more general setting. He considers the problem of finding minimum fill condensation orders for directed graphs and introduces the concept perfect graphs, for which an optimal ordering exist. Finally in [207] it is shown that for general directed graphs the problem of finding a minimum fill ordering is *NP*-complete. Furthermore, it is suggested (without proof) that the even the problem of finding an ordering whose fill-in is within a constant factor of the minimum is *NP*-complete, i.e. that the minimum fill-in problem would be *NP*-complete in the strong sense.

As a result, researchers have focused on heuristic algorithms for finding good condensation orders. In the well known Markowitz strategy [156], for example, the $a_{ij}^{(k)}$ pivot is chosen that minimizes $(r_i^{(k)}-1)(c_j^{(k)}-1)$ where $r_i^{(k)}$ and $c_j^{(k)}$ indicate the number of nonzero entries in the i-th row and j-th column at the k-th elimination step. For reasons of numerical stability this strategy is usually modified so that the selection of very small pivots is avoided [176]. Application of the Markowitz strategy to symmetric matrices is called the minimum degree algorithm [248, 206]. An alternative, more expensive method consists of eliminating the vertex that introduces the least amount of new edges. This is the so called minimum deficiency strategy [156, 248].

A practical implementation of some heuristic algorithms based on the concept of reachable sets of undirected graphs was introduced by George [71]. Marking of eliminated vertices and searching for the reachable ones can be avoided by grouping eliminated vertices in supernodes. The resulting graphs, also called quotient graphs, were used to devise very efficient sparse matrix solvers [70]. This concept can also be used in conjunction with level structures for matrix partioning, leading to the notion of quotient trees [72].

Optimal evaluation of partial derivatives corresponds with node condensation on directed acyclic graphs. It is therefore a special case of the more general vertex elimination problem considered in the literature. Moreover, preconditioning of large interval matrices can be very expensive, see for example [119]. We are therefore not concerned with solving the larger system, but rather with the elimination of the intermediate variables. This problem has not received much attention and is usually handled simply by constraining the renumbering of the variables [51]. Note also that we are interested in optimizing the computational efficiency, and will do so mainly by considering the number of multiplications required for the elimination of intermediate variables. Although this problem is similar to the minimum fill problem of sparse matrix solvers, both problems are not equivalent. Efficiency does not always correspond minimum fill and vice versa [51].

Finally, sparse matrix techniques are usually devised for solving systems with a large number of variables, and as a consequence the complexity of the ordering algorithm itself is very important. Since the number of intermediate variables typically is much smaller in computational graphs, and, as noted in the introduction, the payoff of reduced computation time can be significant, we believe that more complicated polynomial time algorithms can be investigated.

To our knowledge the problem of optimal node condensation on directed acyclic graphs has not yet been addressed.

5.3.2 Node Condensation on Directed Acyclic Graphs

To analyze the optimal computation of partial derivatives we introduce the concept of derivative graphs. Where previously, nodes corresponded with arithmetic operators, they are now associated with intermediate variables. This mapping is illustrated in figure 5.8. In addition we label each arc (x, y) with the local partial derivative, ∂_x^y or S_x^y of y w.r.t. x, here denoted by δ_x^y. Mathematically, derivative graphs express the relation:

$$dy = \sum_{x \in N_G^-(y)} \delta_x^y dx \tag{5.5}$$

Nodes with zero in-degree are called the input nodes $\mathcal{X}(G)$, nodes with zero out-degree function nodes $\mathcal{F}(G)$. The remaining nodes $\mathcal{U}(G) = N - \mathcal{X}(G) - \mathcal{F}(G)$ are called the intermediate nodes. The dimension d of the graph is equal to the number of intermediate

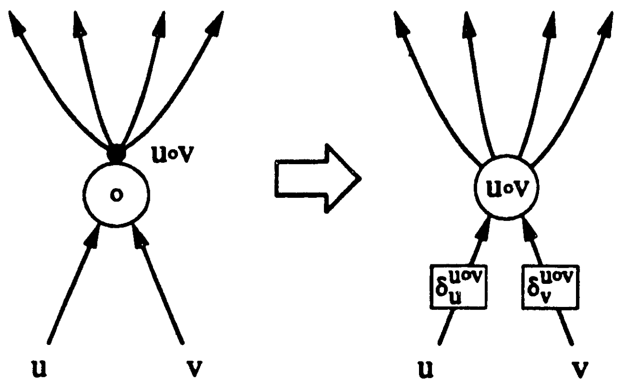


Figure 5.8: Mapping from Computational to Derivative Graphs

nodes $|\mathcal{U}(G)|$.

Iri [108] observes that there is an overhead associated with the computation of the required partial derivatives. We would like to note that this is true only for the division operator. Furthermore, the values of the local partial derivatives are not needed explicitly, only their product, as it appears in the chain rule, has to be computed. In case of binary operators, the overhead can therefore be avoided by proper syntactic transformation of the compiled expressions.

The situation is different for n-ary arithmetic operators. The computation each of the n partial derivatives of an n-ary multiplication operator, for example, involves a product of $n-1$ of its arguments. A more efficient approach would be to replace this operator by a balanced tree of $n-1$ binary multiplication operators. This transformed computational graph now contains $\mathcal{O}(|\alpha_G^-|N)$ nodes. In fact, using techniques similar to the ones used in [16, 108], one can easily verify that, even for the general case, the complexity increases to $\mathcal{O}(|\alpha_G^-|F)$. Alternatively, one could consider n-ary arithmetic operators as supernodes, similar to Speelpenning's funnels [232], containing $n-1$ binary operators and compute both the function value and partial derivatives using the backward evaluation strategy (see section 5.2). In order to avoid the combinatorial problem of optimally transforming n-ary operators in binary ones, and to allow for general, non-arithmetic operators, we will henceforth only consider the problem of finding optimal evaluation strategies for derivative graphs.

Example 5.5

The following lambda-expression defines three functions in terms of three independent variables:

```
(lambda (x1 x2 x3)
   (let* ((v1 (/ x1 x2))
          (v2 (* x1 x2 x3))
          (v3 (+ x1 x2 x3))
          (v4 (+ v1 x2 v2 v3))
          (v5 (sin v3))
          (v6 (* v4 v3 v5))
          (v7 (* x2 v3 v5))
          (v8 (* v3 v5))
          (f1 (exp v6))
          (f2 (+ v6 v7))
          (f3 (+ v7 v8)))
     (list f1 f2 f3)))
```

The associated computational and derivative graphs are shown in figures 5.9 and 5.10 respectively. The input nodes of the derivative graph are $\{x_1, x_2, x_3\}$, $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ are the intermediate nodes and $\{f_1, f_2, f_3\}$ the function nodes.

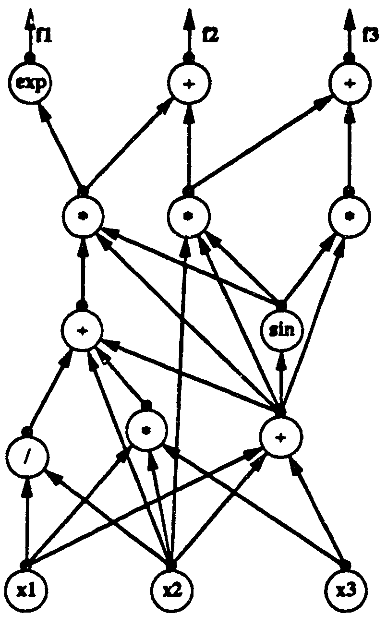


Figure 5.9: Computational Graph of example 5.5

In some cases and as demonstrated in example 5.5, many arcs of the derivative graphs can have unit labels. Although this behavior is typically associated with almost linear arithmetic expressions, it can be taken into account. First, as indicated in section 5.2, unnecessary multiplications can be avoided. Moreover, this information can be incorporated in the optimization problem simply by modifying the multiplication count to leave out the trivial operations.

Condensation is the process of removing a node from a graph, and replacing its arcs with new arcs connecting its negative to its positive neighbors, unless they already are part of the graph. The label of a new arc is the product of the labels of the arcs of the eliminated path. For an existing arc its label is incremented by the same product. The condensation process is illustrated in figure 5.11. Algebraically, condensation of node u corresponds to

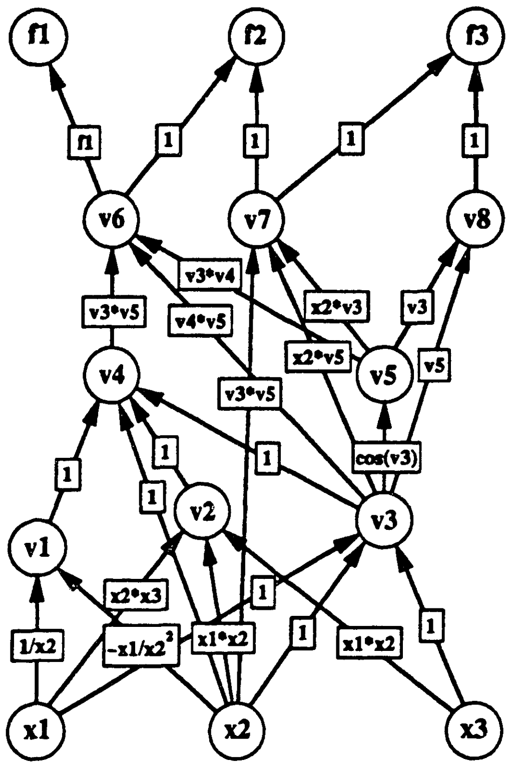


Figure 5.10: Derivative Graph of example 5.5

the elimination of du from the set of equations (5.5) for $y \in N_G^+(u)$. Here we often will take the cost $C(u, G)$ of condensing a node to be the number of required multiplications:

$$M(u,G) = |\alpha_G^-(u)| * |\alpha_G^+(u)| \tag{5.6}$$

However, for most algorithms presented below, there is no additional difficulty in considering the required additions as well. The arcs created by the condensation of an internal node $u \in \mathcal{U}(G)$ are called the deficiency of u:

$$D(u,G) = \{(x,y) \notin A \mid (x,u),(u,y) \in A\}$$

Condensation of u in graph G yields to following condensation graph:

$$G/\{u\} = (N - \{u\}, (A \cup D(u,G)) - \alpha_G(u)) \tag{5.7}$$

A useful property of node condensation is that *the condensation graph is independent of the condensation order.* We refer the reader to [207][Thm 1] for a more formal statement. As a result we will generalize definition (5.7) and call G/U the condensation-graph associated with the condensation of node set U.

The labels of $G/\mathcal{U}(G)$ are the partial derivatives of $\mathcal{F}(G)$ w.r.t. $\mathcal{X}(G)$. The computation of partial derivatives therefore can be viewed as condensation of $\mathcal{U}(G)$ in the derivative graph

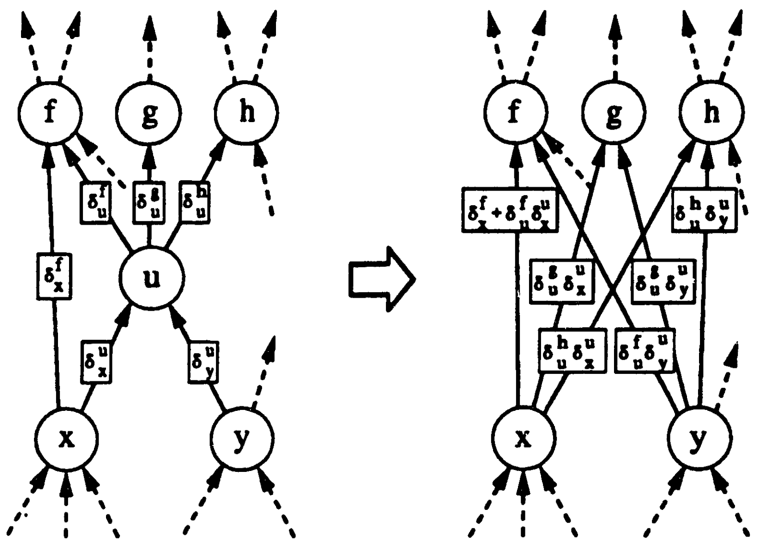


Figure 5.11: Node Condensation

G. If $\nu : \{1,2,\ldots,d\} \leftrightarrow N$, then the cost of the computation can be expressed as:

$$C_\nu(\mathcal{U}(G),G) = \sum_{i=1}^{d} C(\nu(i), G/\bigcup_{j<i}\{\nu(i)\})$$

This leaves us with the problem of determining an optimal condensation order, formally defined as:

$$\nu^* = \min_{\nu} C_\nu(\mathcal{U}(G),G)$$

One particular type of ordering we will consider, is called layered order. Layers l_i are obtained by incrementally removing all the inputs from the graph:

$$l_i = \mathcal{X}(G - \bigcup_{j<i} l_j) \tag{5.8}$$

A chain L of a graph is itself a graph, where each node represents a layer and is connected only to the node representing the subsequent layer. The internal length of the chain $c = |\mathcal{U}(L)|$ is equal to the length of the critical path of $\mathcal{U}(G)$. Note that c represents the minimum required condensation steps with unbounded parallelism. The width w of a graph is the maximum number of nodes in a layer:

$$w = \max_{1\leq i\leq c} |l_i|$$

A property of directed acyclic graphs (N, A) is that nodes $n_i \in N$ can be topologically sorted, i.e. they can be ordered such that $(n_i, n_j) \in A \Rightarrow i < j$. In the algorithms below we will assume that the numbering of the nodes is compatible with in-order numbering of L. As we will see, the ordering of the nodes within each layer is not important.

Note that the forward evaluation of partial derivatives corresponds with an in-order (ν^+)

condensation, while backward evaluation (see section 5.2) corresponds with reverse-order (ν^-) condensation.

Example 5.6

Forward evaluation of the derivative graph of example 5.5 (see figure 5.10) correspond with the condensation order:

$$\nu^+ = \langle v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8 \rangle$$

The associated condensation cost $M_{\nu^+}(\mathcal{U}(G), G)$ is 44. For reverse evaluation we have:

$$\nu^- = \langle v_8, v_7, v_6, v_5, v_4, v_3, v_2, v_1 \rangle$$

with $M_{\nu^+}(\mathcal{U}(G), G) = 42$. An optimal condensation order is:

$$\nu^* = \langle v_1, v_2, v_4, v_5, v_8, v_7, v_3, v_6 \rangle$$

the minimum cost for computing $G/\mathcal{U}(G)$ is $M_{\nu^*}(\mathcal{U}(G), G) = 31$.

Example 5.7

In a layered graph the nodes are organized in layers. Each node, except for the function nodes, is connected to all the nodes of the next layer. One can easily verify that this corresponds with definition (5.8). The $\mathcal{U}(G)$ of the layered graph shown in figure 5.12 has a critical path length $c = 3$. Here we have $M_{\nu^+}(\mathcal{U}(G), G) = 40$. The backward evaluation

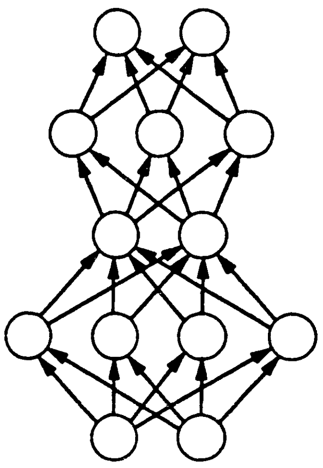


Figure 5.12: Layered Graph of example 5.7

strategy results in $M_{\nu^-}(\mathcal{U}(G), G) = 44$, while $M_{\nu^*}(\mathcal{U}(G), G) = 36$ when the middle layer is condensed last.

We found that for layered graphs, optimal condensation orders are compatible with layered orders, although we were not able to formally verify this claim. Surprisingly, the most expensive condensation orders turned out not to be layered.

Since layered condensation of layered graphs corresponds with chain multiplication of matrices, the optimal condensation problem would then correspond to the matrix chain multiplication problem. For a chain of m this problem can be solved using dynamic programming in $\mathcal{O}(m^3)$ time, see for example [40]. If more sophisticated techniques are used this result can even be improved to $\mathcal{O}(m \log m)$ [103].

Example 5.8

For trees, layered order corresponds with height order, and the length of the critical path is equal to the height of the tree. The $\mathcal{U}(G)$ of the tree in figure 5.13 has a critical path length $c = 2$. For trees reverse-order condensation is optimal. Indeed, condensation of any

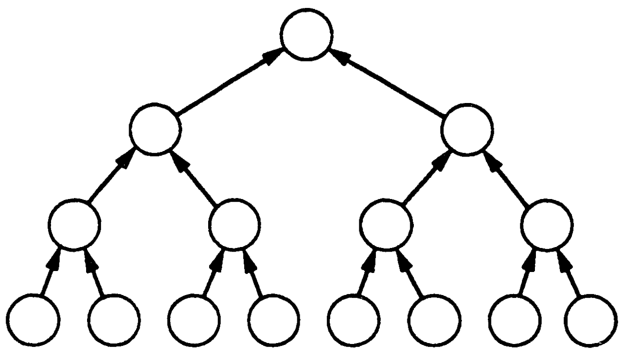


Figure 5.13: Tree Graph of example 5.8

node other than the root of the tree, increases the in-degree of its parent. For this example we have $M_{\nu^*}(\mathcal{U}(G),G) = M_{\nu^-}(\mathcal{U}(G),G) = 12$ and $M_{\nu^+}(\mathcal{U}(G),G) = 16$. Note that similar statements can be made if the direction of the arcs is reversed.

5.3.3 Lower Bounds for Condensation

In this section we will present a technique to compute lower bounds on the number of multiplications required for condensation. We would like to mention that there are some results on lower bounds for gaussian elimination of sparse matrices associated with finite elements or finite difference equations. Indeed, as the associated undirected graphs satisfy an isoperimetric inequality, lower bounds can be computed using so the called element model [55]. Here we consider general directed acyclic graphs, where these bounds can not be used. We therefore develop a model for the computation of exact lower bounds.

The number of multiplications required to condense node u is equal to the in-degree times the out-degree (see equation (5.6)). A lower bound can thus be found by:

$$M^*(u,G) = \min_{U \subseteq \mathcal{U}(G)-\{u\}} |\alpha^-_{G/U}(u)| * |\alpha^+_{G/U}(u)|$$

However, only condensation of the reachable nodes of u can alter its degree. More specifically, $|\alpha^-_G(u)|$ depends only condensation in $R^-_G(u)$, $|\alpha^+_G(u)|$ on condensation in $R^+_G(u)$. The left part of figure 5.14 illustrates the situation for $R^+_G(u)$. Furthermore, as G is acyclic, $R^-_G(u) \cap R^+_G(u) = \emptyset$. So, the problem of minimizing the number of multiplications is decoupled and $|\alpha^-_G(u)|$ can be minimized over $R^-_G(u)$, $|\alpha^+_G(u)|$ over $R^+_G(u)$:

$$M^*(u,G) = \min_{U^- \subseteq R^-_G(u)} |\alpha^-_{G/U^-}(u)| * \min_{U^+ \subseteq R^+_G(u)} |\alpha^+_{G/U^+}(u)|$$

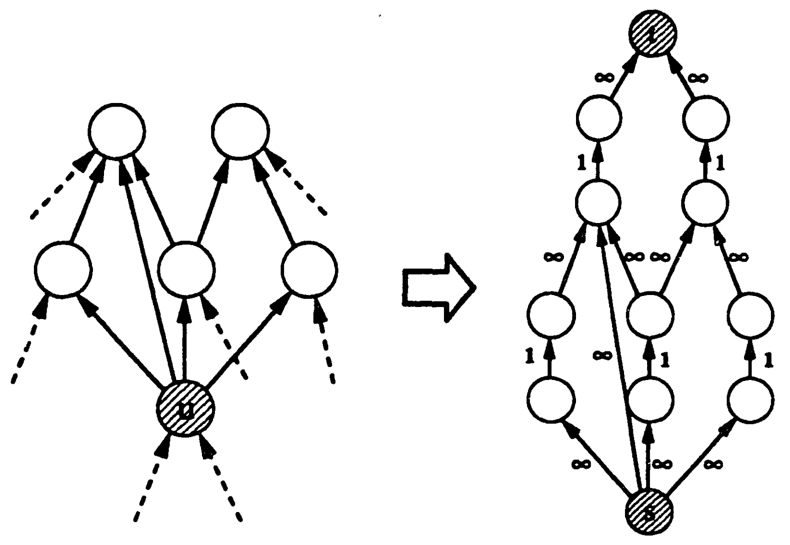


Figure 5.14: Minimum Degree and Maximum Flow

Except for the sign there is no fundamental difference between the positive and negative direction. To simplify the discussion, we will therefore focus only on the minimization of the out-degree. As condensation is path preserving, we know that:

$$v \in R_G^+(u) \Rightarrow R_{G/\{v\}}^+(u) = R_G^+(u) - \{v\}$$

The minimum out-degree of a node is therefore equal to the minimum number of nodes that need to be removed to disconnect u in the subgraph G_u^+ induced by $\{u\} \cup R_G^+(u)$. By transforming G_u^+ this problem can readily be re-casted as a maximum flow problem, as shown in figure 5.14. All nodes of $R_G^+(u)$ are split in two, connected by new arcs with unit capacity. Existing arcs have infinite capacity. The source s corresponds to u and all function nodes $\mathcal{F}(G_u^+)$ are connected to a supersink t. By the max-flow-min-cut theorem, the maximum flow through the network is equal to the capacity of the minimum cut, by construction equal to the minimum number of nodes that need to be removed to disconnect u.

A lower bound for condensation of a set of nodes U can now simply be obtained by summation of the lower bounds of the individual nodes:

$$M_\nu^*(U,G) = \sum_{u \in U} M^*(u,G)$$

As the maximum flow is $\mathcal{O}(w)$, we can simply use the Ford-Fulkerson algorithm [63] to find the maximum flow in $\mathcal{O}(w|A|)$ time. Using this technique a lower bound on the number of multiplications to condense $\mathcal{U}(G)$ can therefore be found in $\mathcal{O}(wd|A|)$. Improvements might be possible since there is partial overlap in the maximum flow computations for each of the nodes. Indeed, a flow path through the network can be decomposed and used for in- and out-degree minimization of each node on that path. Unfortunately, we don't believe that the computation of lower bounds based on maximum flow can readily be extended to include additions or to leave out trivial multiplications.

In the algorithms below we will sometimes only allow for a subset $U \subseteq \mathcal{U}(G)$ to be condensed. In that case the bordering elements in the positive direction are included as outputs for computation of the minimum out-degree, bordering elements in the negative direction is inputs for the minimum in-degree.

Finally, we would like to indicate that this approach can be used to determine a lower bound on the dimensionality of the solution set associated with a computational graph. To see this, let us define the minimum width w^* of a graph G to be the minimum number of nodes to be removed to disconnect $\mathcal{X}(G)$ from $\mathcal{F}(G)$. As before, the minimum width problem can be recast as a maximum flow problem and be computed using the Ford-Fulkerson algorithm. The following example illustrates the required transformation.

Example 5.9

To compute w^* of the derivative graph of example 5.5 (figure 5.10) every node in G is split and connected by an arc of unit capacity. A supersource s is connected to all input nodes $\mathcal{X}(G)$, while all output nodes $\mathcal{F}(G)$ are connected to a supersink t. The resulting flow graph is shown in figure 5.15. For this example the maximum flow is 3, corresponding with the

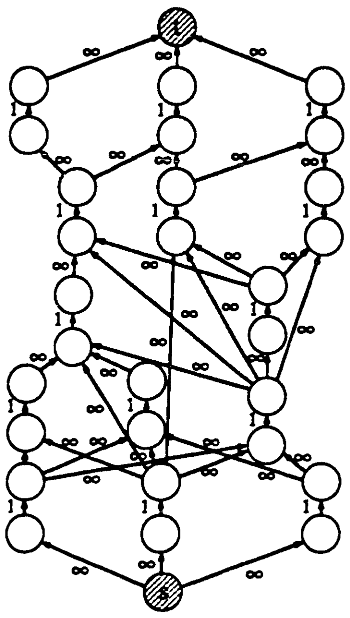


Figure 5.15: Flow Graph of example 5.9

minimum number of nodes that need to be removed to disconnect $\mathcal{X}(G)$ from $\mathcal{F}(G)$.

The minimum width is an upper bound on the rank of the differential of the map from $\mathcal{X}(G)$ to $\mathcal{F}(G)$. Indeed, by definition of the minimum width $\mathcal{F}(G)$ can be written in terms of w^* variables. The rank of the differential can therefore not be bigger then w^*. This implies that if the solution set is not empty, $n - w^*$ is a lower bound on its dimensionality. We illustrate this point with an example.

Example 5.10

Removal of arc (x_2, v_7) from the derivative graph of example 5.5 (figure 5.10) reduces the maximum flow to 2. Indeed, in this case one can write $\{f_1, f_2, f_3\}$ in terms of $\{v_3, v_4\}$:

```
(lambda (v3 v4)
  (let* ((v5 (sin v3))
         (v6 (* v4 v3 v5))
         (v7 (* v3 v5))
         (v8 (* v3 v5))
         (f1 (exp v6))
         (f2 (+ v6 v7))
         (f3 (+ v7 v8)))
    (list f1 f2 f3)))
```

As the rank of the differential is at most 2, the dimensionality of the solution is at least 1, provided there is a solution.

This method provides an a priori criterion to determine whether a system of nonlinear equations is singular. As there are many ways to define the same set of equations, the quality of this criterion depends on the correspondence between the computational graph and the functions they represent. In the context of design (see chapter 2), it should however prove extremely useful. Indeed, here general computational graphs are intended to capture the essence of the functionality. It also nicely complements symbolic or numerical approaches to analyze strong connected components in constraint management [218].

5.3.4 Exact Algorithms for Optimal Condensation

The apparent difficulty of finding exact polynomial algorithms, and the similarity to the NP-complete minimum fill problem suggests that finding optimal condensation orders on directed acyclic graphs might be NP-complete as well. However, as many NP-complete problems of constant tree width can be solved in polynomial time, we would not want to jump to that conclusion without a formal proof.

Nevertheless, non-polynomial algorithms can be effective for small problems, or for off line optimization on frequently used computational graphs. In this section we therefore present two exact algorithms for optimal condensation.

Dynamic Condensation

Dynamic programming can be used to solve the optimal condensation problem. We call the corresponding method dynamic condensation. Since the condensation graph is independent of the condensation order, the problem has an optimal substructure property. Indeed, an optimal condensation order can be associated with each subset $U \subseteq \mathcal{U}(G)$. If we recursively consider to condense each node last:

$$C_{\nu^*}(U,G) = \min_{u \in U} C_{\nu^*}(U - \{u\}, G) + C(u, G/U - \{u\})$$

the solution associated with each set U can be memoized, hereby avoiding the enumeration of all possible condensation orders. Furthermore, if U is not connected we can consider its connected components independently, i.e. the connected components of the corresponding graph G' induced by U. Indeed, the cost of condensing a node u only depends on $R_{G'}(u)$. If we denote a k node connected component of G of by Q_G^k, we can use the following procedure. For $k = 1, \ldots, d$ and for all $Q_G^k \subseteq \mathcal{U}(G)$:

$$C_{\nu^*}(Q_G^k, G) = \min_{u \in Q_G^k} C_{\nu^*}(Q_G^k - \{u\}, G) + C(u, G/Q_G^k - \{u\})$$

To apply this procedure recursively, one needs to add the contributions of each of the connected components $Q_G^{k'}$ of $Q_G^k - \{u\}$. Note that as $k' < k$, $C_{\nu^*}(Q_G^{k'}, G)$ has already been memoized and can be looked up. Based on the numbering discussed earlier, we can represent the connected components as sorted lists, so that memoization can be done using hash tables. The algorithm terminates after all connected components have been processed. If G is connected, $Q_G^d = \mathcal{U}(G)$ and the solution can be found by lookup. Otherwise, the contributions of all connected components of $\mathcal{U}(G)$ have to be added.

The number of possible subsets of $\mathcal{U}(G)$, equal to $2^d - 1$, is an upper bound on the number of connected components. As every node of each component is considered last, the worst case complexity of dynamic condensation is $\mathcal{O}(d2^d)$. For total graphs, where each node is connected to all subsequent nodes, this bound is tight. For sparse graphs it is however very conservative. In fact, we believe that for particular types of graphs, better bounds can be found.

Example 5.11

Let us illustrate this method on the derivative graph of example 5.5 shown in figure 5.10. We first consider all connected components Q_G^1, i.e. every node on its own. For example, v_5 has a multiplication count of 3, v_7 of 6 and v_8 of 2. We then continue with the connected components of two elements, i.e. Q_G^2. Consider for example $\{v_5, v_7\}$. If v_5 is condensed last we have $6+4 = 10$ multiplications, whereas condensing v_7 last yields $3+4 = 7$. The optimal order ν^* for this connected component is thus $\langle v_5, v_7 \rangle$ and $M_{\nu^*}(\{v_5, v_7\}, G) = 7$. Similarly for $\{v_5, v_8\}$, condensing v_5 last will need $2+3 = 5$ multiplications, v_8 last $3+1 = 4$. Here ν^* is $\langle v_5, v_8 \rangle$ and $M_{\nu^*}(\{v_5, v_8\}, G) = 4$. Observe that $\{v_7, v_8\}$ will not be considered, as it is not connected. Let us see how the algorithm works on $\{v_5, v_7, v_8\}$. If v_7 is condensed last, we already know that the optimal cost for condensing $\{v_5, v_8\}$ is 4. We therefore count $4+4 = 8$ multiplications, which happens to be equal to $7+1 = 8$ when v_8 is condensed last. Now, condensing v_5 last, results in two connected components $\{v_7\}$ and $\{v_8\}$. So here we have $6+2+3 = 11$. An optimal condensation sequence for this connected component is therefore $\langle v_5, v_7, v_8 \rangle$ with $M_{\nu^*}(\{v_5, v_7, v_8\}, G) = 8$.

To find an optimal solution (see example 5.6) for $Q_G^8 = \mathcal{U}(G)$, the dynamic condensation algorithm considered 85 connected components . Note that this is much less then $2^8 - 1 = 255$.

In many cases there is more then one optimal order. Indeed, only the relative order each node u w.r.t. $R_G(u)$ is of importance. So ν^* really only corresponds with a partial order. This implies that while the problem optimizes sequential operations, some parallelism can be available.

Example 5.12

We illustrate this point by considering the optimal sequence of example 5.6. As $v_2 \notin R_G(v_1)$ their relative order is not important. However, $v_4 \in R_G(v_1)$ and $v_4 \in R_G(v_2)$. The relative order of v_1 and v_2 therefore needs to be preserved. The optimal partial order is illustrated in figure 5.16, here the arrows indicate the precedence relation. Observe that the minimum

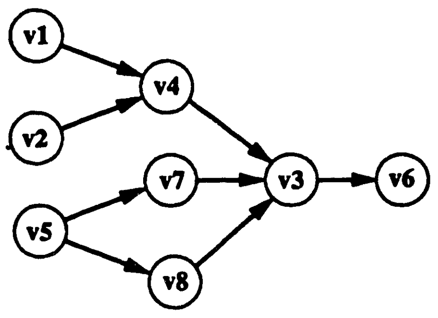


Figure 5.16: Optimal Partial order of example 5.6

number of condensation steps is 4, whereas $c = 3$, demonstrating the natural compromise between sequential and parallel optimization.

Constraint Condensation

Constraint condensation uses additional information to attempt to reduce the number of connected components to be considered in dynamic condensation. The first observation is that it is not needed to consider all $u \in U$ last. Indeed, as indicated earlier, if $v \notin R_G(u)$, the relative order of u and v is not important and we can for example enumerate them in ascending order. To minimize $C_{\nu^*}(\mathcal{U}(G) - V, G)$ order reversal is required only if:

$$u \in \bigcup_{v \in V} R_G(v)$$

The second observation is that lower bounds (see section 5.3.3) can be used in a branch and bound strategy to prune the search space. As upper bounds we can for example take the cost associated with a heuristic cheapest first strategy (see section 5.3.5 below). In addition we can explore the search space by considering the cheapest node first, as in a best first branch and bound strategy. Since this corresponds with our upper bound, the search space is constrained only by the lower bound:

$$C_\nu(U, G) \geq C_\nu^*(U - \{u\}, G) + C(u, G/U - \{u\})$$

where $C_{\nu^*}(Q_G^k, G)$ can replace $C_\nu^*(Q_G^k, G)$ as the computation proceeds.

Example 5.13

As we did in example 5.11, let us consider again the condensation of $\{v_5, v_7, v_8\}$ of the derivative graph shown in figure 5.10. The cheapest first strategy, would produce $\langle v_8, v_5, v_7 \rangle$. So, in our best first strategy, we begin by considering to condense v_7 last, costing 4 multiplications, preceded by v_5, since $v_5 \in R_G(v_7)$, costing an additional 3 multiplications. v_5 in turn will be preceded by v_8 with 2 multiplications. So this order has a multiplication count of 9. Then v_8, respecting ascending order, will precede v_7 with 1 multiplication, itself preceded by v_5 as $v_5 \in R_G(v_8)$, for an additional 3 multiplications. The optimal solution for the set $\{v_5, v_8\}$ is therefore $\langle v_5, v_8\rangle$ with a cost of 4 and can now be memoized. So the cost associated with $\langle v_5, v_8, v_7\rangle$ is 8 multiplications. We then proceed to consider to condense v_5 last for 3 multiplications. At this point two connected components remain $\{v_7\}$ and $\{v_8\}$. However, condensation of $\{v_7\}$ now has lower bound of 6, already adding up to at least $3 + 6 = 9$ multiplications, which is higher then 8 multiplications we found before. As a result we will not even consider $\{v_8\}$ and abandon this order. Remains then to consider v_8 last for 1 multiplication. The lower bound to condense $\{v_5, v_7\}$ is 7, so that again we already know that we can not improve on $\langle v_5, v_8, v_7\rangle$ with 8 multiplications.

For $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\} = \mathcal{U}(G)$, constraint condensation found the optimal solution by considering only 17 connected components.

Although often only very few connected components are considered, more work is done for each one. In fact, the worst case complexity of constraint condensation, $\mathcal{O}(wd^2|A|2^d)$ is higher then the complexity of dynamic condensation.

Example 5.14

Dynamic condensation of the tree of example 5.8 analyzed 12 connected components and in this case constraint condensation considered all 12 as well. For the layered graph of example 5.7 390 connected components were considered by dynamic condensation, while constraint condensation only considered 11.

5.3.5 Heuristic Algorithms for Optimal Condensation

As was noted earlier, chances are that the optimal condensation problem on directed acyclic graphs is NP-complete. So, as for sparse matrix solvers, unless $P = NP$, we have to resort to heuristic algorithms. In this section we describe a number of polynomial time algorithms that usually find a good condensation order.

Local Strategies

A simple heuristic, often used on undirected graphs is the Markowitz strategy. Although we have indicated in section 5.3.1 that minimization of the fill-in is not equivalent to minimization of the multiplication count, the local minimization done by Markowitz strategy is approximately equivalent. In this section we generalize the idea of local minimization and apply it to directed acyclic graphs.

All local strategies follow the pattern:

$$\min_{u \in \mathcal{U}(G)} H(u, G) \tag{5.9}$$

to select the next node u to condense and are recursively applied to the resulting condensation graph $G/\{u\}$. They differ only by the heuristic $H(u, G)$ used. We have considered a number of different heuristics. The *cheapest first* strategy uses:

$$H(u, G) = C(u, G)$$

and has a complexity of $\mathcal{O}(d)$. A different approach is to select the node with cost the closest to its lower bound. For this *best buy* strategy we have:

$$H(u,G) = C(u,G) - C^*(u,G)$$

When cost is taken to be the number of required multications, we can simply use the results of section 5.3.3:

$$H(u,G) = M(u,G) - M^*(u,G)$$

The corresponding time complexity is $\mathcal{O}(wd|A|)$. Finally, the *minimum deficiency* strategy often used in sparse matrix solvers has the heuristic:

$$H(u,G) = |D(u,G)|$$

and runs in $\mathcal{O}(d)$ time.

Example 5.15

Consider again the derivative graph (figure 5.10) of example 5.5. The cheapest first strategy would first choose either v_1 or v_8 since both of them have a cost of 2 multiplications. We break the ties say by choosing ascending order and select to condense v_1 first. After condensation v_8 still is the cheapest node and we condense it next. We then choose v_2 at 3 multiplications by breaking the tie with v_5. Now however, the cost of v_4 is 3 so we choose it next, and so on. The resulting order is:

$$\nu = \langle v_1, v_8, v_2, v_4, v_5, v_7, v_6, v_3 \rangle$$

with $M_\nu(\mathcal{U}(G), G) = 34$. The best buy first strategy proceeds in a similar fashion, we find:

$$\nu = \langle v_1, v_2, v_4, v_5, v_8, v_7, v_3, v_6 \rangle$$

Here, $M_\nu(\mathcal{U}(G), G) = 31$. By comparison the cost associated with the minimum deficiency ordering is 38 again indicating that minimum fill-in and minimum cost are not the same.

In general, we found that the best buy strategy is not necessarily better then the simpler cheapest first strategy. Also, as the following example indicates, neither of them works well on trees.

Example 5.16

Both cheapest first and best buy condensation of the tree of example 5.8 resulted in 16 multiplications, instead of 12. On the layered graph of example 5.7 however both of them found an optimal order.

Look Ahead Strategies

Formal analyses of local strategies for sparse matrix solvers, carried out on limited problem classes, revealed that tie breaking algorithms can have a strong influence on the results [51]. Based on this observation, and our experience with the local strategies of section 5.3.5, we propose a number of more powerful techniques that fall under the category of *look ahead* strategies [255]. Not only do they take each node u into consideration, they also take advantage of the corresponding condensation graph $G/\{u\}$. In general they conform to the following pattern:

$$\min_{u \in \mathcal{U}(G)} C(u, G) + H_\nu(\mathcal{U}(G) - \{u\}, G/\{u\}) \tag{5.10}$$

Again, various heuristics can be used on the condensation graph. A fairly simple estimate of the remaining cost of the condensation graph is to add the cost of each node:

$$H_\nu(U, G) = \sum_{u \in U} C(u, G)$$

This *count ahead* strategy has a complexity of $\mathcal{O}(d^2)$. Alternatively, we can use the cheapest first strategy on the condensation graph. This leads to the following, recursive definition:

$$H_\nu(U, G) = \min_{u \in U} C(u, G) + H_\nu(U - \{u\}, G/\{u\})$$

again with a $\mathcal{O}(d^2)$ time complexity. Let us call this the *cheapest ahead* strategy. Finally, we can asses how the lower bounds on the multiplication count will be affected by the condensation of each node:

$$H_\nu(U, G) = M_\nu^*(U, G)$$

The running time for this *bound ahead* strategy is $\mathcal{O}(wd^2|A|)$.

In general we have found that all three of them performed very well, although the bound ahead strategy, for a higher cost, would be tend to be a little more consistent. In addition, for trees bound ahead will always find an optimal sequence.

Example 5.17

For the graph of example 5.5, both the count ahead and cheapest ahead strategy found orders with 32 multiplications, whereas bound ahead did find an optimal order with 31 multiplications.

Example 5.18

All three strategies found an optimal order both for the tree of example 5.8 and for the layered graph of example 5.7.

Although we have experimented with combinations of the above heuristics, for example by averaging lower and upper bounds on the remaining cost, we found that it generally did not pay off.

Binary and Layered Condensation

The basic idea of the binary condensation strategy, is that while it is expensive to enumerate all $d! = \mathcal{O}(2^{d \log d})$ orders, 2^d of them can be enumerated in $\mathcal{O}(d)$ time. This can be done, for example by constructing a binary tree where each node indicates whether the left subtree precedes the right subtree, and the leaves are associated with the nodes. For node condensation this corresponds to recursively split sets of nodes in half and consider whether or not to condense one half before the other. So here, as the amount of work is proportional to the number of nodes in the set, this condensation strategy will have a time complexity of $\mathcal{O}(d \log d)$. As we indicated before, the numbering of the nodes is compatible with in-order layered order. This will limit the number of trivial exchanges performed in the binary condensation strategy.

Example 5.19

Let us illustrate this strategy on example 5.5. We first split $\mathcal{U}(G)$ into two halves:

$$\begin{aligned} U_1 &= \{v_1, v_2, v_3, v_4\} \\ U_2 &= \{v_5, v_6, v_7, v_8\} \end{aligned}$$

The problem is then split into two subproblems, one where we recursively consider to optimally condense U_1 first, then U_2 and vice versa. For this example binary condensation found a condensation order with a multiplication count of 32.

Example 5.20

For both the tree of example 5.8 and the layered graph of example 5.7 binary condensation found an optimal order.

As illustrated in example 5.19, the recursive division sometimes splits nodes belonging to the same layer, hence performing trivial exchanges. We now introduce a technique, called layered condensation, designed to take advantage of the layered structure. This concept is akin to the level structure methods for quotient graphs [69] or to the optimal multiplication of factors considered in [232].

Layered condensation considers only layered orders, i.e. only to condense nodes one layer at a time. As a consequence, it is equivalent to condensation of the chain L associated with G with $C(l,L) = C_{\nu^+}(l,G)$, where we arbitrarily have chosen in-order condensation within each layer. This problem can readily be solved by dynamic condensation with a complexity $\mathcal{O}(wc^3)$, which can be reduced to $\mathcal{O}(wc\log c)$ as mentioned in example 5.7. Observe that as the definition of layers in equation (5.8) is not unique, other definitions can produce different results.

Layered condensation finds ν^* for trees and as mentioned in example 5.7 for layered graphs. In general we have found it to work very well on layered-like graphs, less so on dense and unstructured ones.

Example 5.21

The layers of example 5.5 are:

$$\begin{aligned} l_1 &= \{v_1, v_2, v_3\} \\ l_2 &= \{v_4, v_5\} \\ l_3 &= \{v_6, v_7, v_8\} \end{aligned}$$

We therefore consider to condense respectively l_1, l_2 and l_3 last and apply the same technique to the remaining layers. Note that in fact some arcs can cross layers, e.g. (v_3, v_6). So, when l_2 is considered to be condensed last the ordering of $\{l_1\}$ w.r.t. $\{l_3\}$ does make a difference. Although this can easily be resolved by only considering in-order condensation, in our implementation we have chosen to check which order effectively produced the lowest number of multiplications. The layered order found by layered condensation is:

$$\nu = \langle v_4, v_5, v_1, v_2, v_3, v_6, v_7, v_8 \rangle$$

with $M_\nu(\mathcal{U}(G), G) = 35$.

We would like to note that this method only considers condensation orders with minimum critical path length, i.e. it attempts to optimize the number of operations, while not compromising on parallel execution time.

Example 5.22

Figure 5.17 illustrates the partial order associated with the layered order of example 5.21. Observe that the minimum number of condensation steps is indeed $c = 3$.

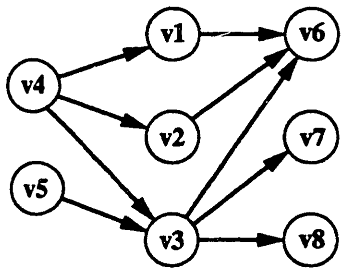


Figure 5.17: Optimal layered order of example 5.21

Local Search

A different approach to the solution of the optimal condensation problem is to incrementally improve a given ordering. The resulting ordering, while not guaranteed to be a minimum ordering, is a minimal ordering [207].

2-opt condensation recursively considers to invert two nodes in a condensation order. More specifically, it considers to invert a node u with its next neighbor $v \in N_G(u)$ in the current graph G. If this results in a lower cost the inversion is performed, otherwise one proceeds with the following node. When the last node is reached, the process is repeated. The algorithm terminates when the multiplication count cannot further be reduced. Typically, this type of algorithm is used on an ordering obtained by a simple heuristic. For simplicity, we have chosen the use the cheapest first ordering as initial ordering.

Example 5.23

The cheapest first ordering as discussed in example 5.15 can first be improved by inverting v_5 and v_8:

$$\nu = \langle v_1, v_2, v_4, v_5, v_8, v_7, v_6, v_3 \rangle$$

reducing $M_\nu(\mathcal{U}(G), G)$ to 33. Note that we have used the fact that unless $v \in N_G(u)$ their mutual order is not important. The multiplication count is further reduced by inversion of v_4 and v_6 to the minimal ordering:

$$\nu = \langle v_1, v_2, v_5, v_8, v_7, v_6, v_4, v_3 \rangle$$

with $M_\nu(\mathcal{U}(G), G) = 32$.

Although, 2-opt condensation can be implemented fairly efficiently, it potentially can take a long time to terminate. Indeed, if the multiplication count reduces only by one each iteration, an upper bound on the number of iterations is $\mathcal{O}(M_\nu(\mathcal{U}(G),G))$. In some cases this can be excessive. Fortunately, lower bounds on the multiplication count can be used to control this process.

Other Strategies

A very simple strategy is to sort the nodes by increasing cost. This strategy, which chooses the most expensive node last, ignores topological changes associated with the condensation process. Although, in some cases this can lead to rather inefficient condensation orders, often it does produce good results.

When parts of graphs are trees, one could choose to condense them first by reverse-order condensation, resorting to other techniques afterwards. This, in fact corresponds to grouping trees into supernodes (see section 5.3.2).

Chapter 6

Variable Precision in Interval Computations

6.1 Introduction

Variable precision representations are particularly well suited to be used in conjunction with interval arithmetic techniques. Indeed, a wide interval usually does not require a very accurate representation. Solution techniques for nonlinear equations for example, need only to represent the significant digits. These can be determined by comparing upper and lower interval bounds.

At first the gains may seem insignificant, but variable precision is very useful when rational arithmetic is used. Indeed, it allows us to control the growth of the rational number representation by upward and downward rounding at the required accuracy. Furthermore, although for arithmetic expressions it is possible to do all intermediate computations in full precision, this is no longer possible for extended arithmetic expressions, which can include transcendental functions. Variable precision, still allows us to achieve arbitrary accuracy, while using finite representations.

Algorithms for the solution of linear systems to arbitrary precision can be found in [20, 47]. A comparison of traditional, interval arithmetic and variable precision implementations of numerical methods is presented in [110]. It does however appear that the availability of high-accuracy floating point processors, has reduced the need for multiple precision in numerical computations and in recent years very little publications have appeared on this subject.

The main reason for addressing this issue here however, is that fc. .nore general operators, associated with more complicated models, it allows for automatic control of the modeling accuracy. In this case selecting the appropriate model can indeed be of crucial importance. The basic idea is to use as much precision as needed to maintain quadratic convergence of the nonlinear solver. This approach has for example been applied in conjunction with discretizations of nonlinear operator equations, where it is called *the Mesh-Independence Principle* [9, 11].

In this chapter we present a formal approach to the problem of controlling the precision of interval bounds for the solution of nonlinear algebraic equations. Multiple precision methods where the machine precision is increased at each iteration are presented in section 6.2. Variable precision on the other hand uses as much precision as needed based on the width on the interval, this is discussed in section 6.3. Finally, in section 6.4 we combine both approaches in a controlled precision method.

6.2 Multiple Precision

While traditional floating point arithmetic always uses a fixed machine precision, with multiple precision arithmetic the machine precision can be changed as the computation proceeds. The use of multiple precision arithmetic in numerical computations has been investigated from a theoretical point of view in [23, 25], while practical implementation aspects are presented in [24, 107]. Complete multiple precision packages are available and are described in [27, 26, 215, 106]. The multiple precision approach is to incrementally increase the machine precision as more accuracy is needed. For the solution of nonlinear equations, for example, the precision can simply be doubled after each iterate of a quadratically convergent nonlinear solution operator. Using multiple precision methods, Brent [25] showed that if it takes $w(m)$ time to evaluate $f^{(i)}(x)$ with an absolute error $\mathcal{O}(2^{-m})$, a simple zero $\zeta \neq 0$ of $f(x)$ can be evaluated to precision m in $\Theta(w(m))$ time, with an asymptotic constant of 1.

A m-digit finite interval representation with base b would be obtained by defining $P_{l,u} = b^{p_f}$ with $p_f = r + 1 - m$ and and $r = \lfloor \log_b |X_{l,u}| \rfloor$:

$$X^\circ = [\lfloor \frac{X_l}{P_l} \rfloor P_l \, , \, \lceil \frac{X_u}{P_u} \rceil P_u] \tag{6.1}$$

The relative machine accuracy is defined to be $\varepsilon_M = b^{1-m}$, m is the machine precision. Observe that

$$d(X \, , X^\circ) \leq (|X| + 1)\varepsilon_M$$

assuming the smallest machine representable number is smaller then ε_M.

Example 6.1

The 3 digit fixed precision representation of $X = [1.2345 \, , 1.2356]$ with $b = 10$ is:

$$X^\circ = [1.23 \, , 1.24]$$

the corresponding relative accuracy $\varepsilon_M = 10^{-2}$.

Here, and in the discussion below we assume that elementary binary operators of an implementation obey:

$$f^\circ(X) = (g(X) \circ h(X))^\circ$$

Similarly, for unary operators $\phi \in \Phi$ we assume:

$$\phi^\circ(g(X)) = (\phi(g(X)))^\circ$$

Let us now investigate how interval arithmetic computations are affected by the relative machine accuracy ε_M. Based on Lipschitz continuity of arithmetic expressions, Moore [160] showed that :

$$\exists l_f \text{ and } \varepsilon_f \in \mathbf{R} \; : \; d(f(X), f^\circ(X)) \leq l_f \varepsilon_M \tag{6.2}$$

provided that $\varepsilon_M \leq \varepsilon_f$. Where $f^\circ(X)$ denotes the interval value of the arithmetic expression evaluated with relative accuracy ε_M.

Example 6.2

To illustrate the above result, we computed the solution to equations (3.12) of example 3.15 using a midpoint preconditioned Gauss-Seidel operator. If exact arithmetic is used we find:

$$\gamma(A,B,X) = \begin{pmatrix} [-405\,,\, 442.5] \\ [-261.428571\,,\, 424.285715] \end{pmatrix}$$

where only 9 digits have been shown. If computations are performed with 2-digit precision, the solution is:

$$\gamma^\circ(A,B,X) = \begin{pmatrix} [-440\,,\, 480] \\ [-330\,,\, 500] \end{pmatrix}$$

with 4-digit precision:

$$\gamma^\circ(A,B,X) = \begin{pmatrix} [-405.4\,,\, 442.9] \\ [-262.1\,,\, 425.1] \end{pmatrix}$$

Finally, with 8-digit precision we have:

$$\gamma^\circ(A,B,X) = \begin{pmatrix} [-405.00004\,,\, 442.50004] \\ [-261.42864\,,\, 424.285795] \end{pmatrix}$$

hereby verifying that *doubling the machine precision approximately doubles the accuracy of the interval bounds.*

On the other hand a fixed machine precision places limitations on the obtainable accuracy. Solutions of nonlinear equations, for example, might not be exactly machine representable. This implies that one cannot expect to achieve a better relative accuracy then ε_M. Let us illustrate this point with and example.

Example 6.3

Consider the Hansen-Sengupta operator to solve equations (3.27) of example 3.19. If exact arithmetic is used, the following iterates are obtained:

iteration	X	Y
0	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	[+0.3v0 +0.7v0]	[+0.81v0 +0.90v0]
2	[+0.495v0 +0.505v0]	[+0.865v0 +0.868v0]
3	[+0.499998v0 +0.500002v0]	[+0.8660253v0 +0.8660256v0]
4	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

up to 10-digit accuracy. If now we use only 2-digit precision, we have:

iteration	X	Y
0	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	[+0.3v0 +0.7v0]	[+0.7v0 +0.9v0]
2	[+0.4v0 +0.6v0]	[+0.85v0 +0.89v0]
3	[+0.4v0 +0.6v0]	[+0.85v0 +0.88v0]

Here, termination occurred when two subsequent iterates were identical, and indeed at most two digits are accurate. Finally, if we double the machine precision to 4:

```
iteration          X                        Y
    0        [+0.2v0 +0.8v0]          [+0.6v0 +0.9v0]
    1        [+0.3v0 +0.7v0]         [+0.81v0 +0.90v0]
    2      [+0.495v0 +0.505v0]      [+0.865v0 +0.868v0]
    3      [+0.499v0 +0.501v0]     [+0.8655v0 +0.8662v0]
    4     [+0.4998v0 +0.5002v0]    [+0.8659v0 +0.8662v0]
```

the accuracy of corresponding results doubles as well.

Let us now, based on the proof presented in [171], modify this theorem slightly for the case of multiple precision

Theorem 2 *If an arithmetic expression f is Lipschitz at X, $\exists\,\lambda_f$ and l_f such that:*

$$d(f(X)\,,f^{\circ}(X)) \leq l_f w(X)^{\alpha}$$

for relative machine accuracy $\varepsilon_M \leq l^{\circ} w(X)^{\alpha}$ with $l^{\circ} \leq \lambda_f$ and $\alpha \geq 1$.

Proof: Since $\varepsilon_M \leq l^{\circ} w(X)^{\alpha}$, this is certainly true for variables:

$$d(X\,,X^{\circ}) \leq (|X|+1)\lambda_f w(X)^{\alpha}$$

and a similar inequality can be written for constants, as well as for the midpoint operator. Assume now that the theorem holds for two subexpressions g and h:

$$\begin{aligned} d(g(X)\,,g^{\circ}(X)) &\leq l_g w(X)^{\alpha} \\ d(h(X)\,,h^{\circ}(X)) &\leq l_h w(X)^{\alpha} \end{aligned}$$

Since f is Lipschitz at X we know that $g^{\circ}(X) \circ h^{\circ}(X)$ is defined for sufficiently small l°, say $l^{\circ} \leq \lambda_f$. By Lipschitz continuity of the arithmetic operator $\circ$ we have:

$$d(f(X)\,,g^{\circ}(X)\circ h^{\circ}(X)) \leq l_{g^{\circ}\circ h^{\circ}} w(X)^{\alpha}$$

where the Lipschitz constant $l_{g^{\circ}\circ h^{\circ}}$ is slightly bigger then $l_{g\circ h}$ due to the rounding. Furthermore, with fixed precision:

$$d(g^{\circ}(X)\circ h^{\circ}(X)\,,f^{\circ}(X)) \leq (|g^{\circ}(X)\circ h^{\circ}(X)|+1)\lambda_f w(X)^{\alpha}$$

so that by the triangle inequality:

$$l_f = l_{g^{\circ}\circ h^{\circ}} + (|g^{\circ}(X)\circ h^{\circ}(X)|+1)\lambda_f$$

The proof for elementary unary operators is almost the same. □

All linear solution operators $\Lambda(X)$ discussed in 3 are Lipschitz continuous arithmetic expressions if the midpoint preconditioned Lipschitz matrix is regular. In fact, their convergence implies that their Lipschitz constant l_{Λ} is smaller then 1. When used as nonlinear solution operators $\Psi(X)$ with a variable Lipschitz matrix, e.g. $J(X)$, the convergence is quadratic:

$$w(\Psi(X)) \leq q_{\Psi} w(X)^2 \tag{6.3}$$

The above theorem implies that if the accuracy is controlled by $w(X)^2$ quadratic convergence is maintained. Indeed, if:

$$d(\Psi(X), \Psi^\circ(X)) \leq l_\Psi w(X)^2$$

we have:

$$w(\Psi^\circ(X)) \leq (q_\Psi + 2l_\Psi)w(X)^2 \tag{6.4}$$

by letting $\varepsilon_M = l_\Psi w(X)^2$. Note that if $q_\Psi w(X) < 1$, l_Ψ can be chosen small enough to ensure $(q_\Psi + 2l_\Psi)w(X) < 1$.

The above result is the interval equivalent of the multiple precision approach used in conjunction with classical numerical solution methods. In fact, interval width provides a very effective measure for controlling the precision, unlike the often used convergence estimates.

Example 6.4

We illustrate this method on the same iteration as in example 6.3. We have taken $\lambda_\Psi = 10^{-1}$. We obtain the following iterates:

iteration	digits	X	Y
0	2	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	2	[+0.3v0 +0.7v0]	[+0.7v0 +0.9v0]
2	3	[+0.48v0 +0.52v0]	[+0.86v0 +0.88v0]
3	5	[+0.4998v0 +0.5002v0]	[+0.86601v0 +0.86609v0]
4	9	[+0.49999999v0 +0.50000001v0]	[+0.866025398v0 +0.866025407v0]
5	17	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

up to 10-digit accuracy. Observe, that initially the precision does not need to be doubled due to the slower convergence.

For traditional implementations, which usually provide single and double precision, this method can be used to automatically increase the precision when it becomes necessary. Finally, depending on the order of convergence, which can be different then 2, and how the computation cost depends on the accuracy, the constant α can be adjusted to reduce the overall computation time.

6.3 Variable Interval Precision

It is however not necessary to perform all the arithmetic with the same precision at each iteration. The number of significand digits of interval bounds can be estimated in a straightforward fashion, see for example [115] for a simple significance rule.

We would like to indicate that this type of variable precision representations is not suited for all types of interval computations. In particular, it should not be used if the accuracy of the bounds is not related to the width of the interval. This occurs for example in optimization where typically only one of the function bounds is refined.

A variable interval representation is obtained by defining $P = b^{p_v}$ with $p_v = \lfloor \log_b \lambda_M w(X) \rfloor$ so that the following operation:

$$X^\circ = [\lfloor \frac{X_l}{P} \rfloor P \, , \lceil \frac{X_u}{P} \rceil P] \tag{6.5}$$

retains only the significant digits of X while still allowing for arbitrary precision. When $w(X) = 0$, we define $X^\circ = X$. The machine Lipschitz constant $0 < \lambda_M \leq 1$ has been introduced to control the loss of significance. Let us define $\lfloor -\log_b \lambda_M \rfloor$ to be the number of guard digits, see [115] for a similar definition.

Example 6.5

Let us take the same interval as example 6.1 to illustrate the variable interval representation. If we use a machine Lipschitz constant of 1 and $b = 10$ we have:

$$X^\circ = [1.234\,,\,1.236]$$

Here the number of guard digits is zero.

Example 6.6

Computation of the Gauss-Seidel solution of example 6.2 using variable interval precision with $\lambda_M = 10^{-1}$ gives:

$$\gamma^\circ(A,B,X) = \begin{pmatrix} [-440\,,\,470] \\ [-310\,,\,470] \end{pmatrix}$$

as compared to the variable precision rounding of the exact result:

$$(\gamma(A,B,X))^\circ = \begin{pmatrix} [-410\,,\,450] \\ [-270\,,\,430] \end{pmatrix}$$

One can easily verify that the variable precision representation defined in equation (6.5) satisfies:

$$d(X\,,\,X^\circ) \leq \lambda_M w(X) \tag{6.6}$$

and equivalently:

$$w(X^\circ) \leq (1 + 2\lambda_M) w(X) \tag{6.7}$$

Theorem 3 *If the arithmetic expression f is Lipschitz at X, $\exists\, \lambda_f$ and l_f such that:*

$$d(f(X)\,,\,f^\circ(X)) \leq l_f w(f(X))$$

for variable precision with $\lambda_M \leq \lambda_f$.

Proof: For variables the statement is true by equation (6.6) and the same reasoning applies to constants. For the midpoint operator we have $d(X_c\,,\,X_c^\circ) = 0$. We now proceed inductively and assume that for two subexpressions g and h:

$$\begin{aligned} d(g(X)\,,\,g^\circ(X)) &\leq l_g w(g(X)) \\ d(h(X)\,,\,h^\circ(X)) &\leq l_h w(h(X)) \end{aligned}$$

As before, we know that $g^\circ(X) \circ h^\circ(X)$ is defined for sufficiently small λ_M, say $\lambda_M \leq \lambda_f$. Furthermore $\exists\, l'_f$ such that:

$$d(f(X)\,,\,g^\circ(X) \circ h^\circ(X)) \leq l'_f w(f(X))$$

In addition, if variable interval precision is used $\exists\, l''_f$:

$$d(g^\circ(X) \circ h^\circ(X)\,,\, f^\circ(X)) \le l''_f w(f(X))$$

so that by the triangle inequality:

$$l_f = l'_f + l''_f$$

The values of the constants l'_f and l''_f for arithmetic operators, as well as for unary elementary operators are derived in appendix B.1. □

For a quadratically converging Lipschitz operator $\Psi(X)$ (cf. equation (6.3)) we therefore have:

$$\begin{aligned} d(\Psi(X)\,,\,\Psi^\circ(X)) &\le l_\Psi w(\Psi(X)) \\ &\le l_\Psi q_\Psi w(X)^2 \end{aligned}$$

so that:

$$w(\Psi^\circ(X)) \le q_\Psi(1 + l_\Psi) w(X)^2 \tag{6.8}$$

This result allows us to locally adjust the required precision relative to the interval width, such that quadratic convergence is maintained.

Example 6.7

Using $\lambda_M = 10^{-1}$ we apply this method to the same iteration as example 6.3 and find:

iteration	X	Y
0	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	[+0.3v0 +0.7v0]	[+0.8v0 +0.9v0]
2	[+0.49v0 +0.51v0]	[+0.864v0 +0.869v0]
3	[+0.49998v0 +0.50002v0]	[+0.866024v0 +0.866027v0]
4	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

We have again terminated the iteration at 10-digit accuracy. However, as mentioned earlier, this approach allows us to achieve arbitrary accuracy using finite representations.

Clearly, the above result is valid for any order of convergence. This means that *variable interval precision preserves the convergence of the solution operator.* Furthermore, as in 6.2 the precision of the computation approximately doubles at each iteration. The results of Brent [25] therefore apply so that if $w(m)$ denotes the time to evaluate $f(\_)$ with precision m, using variable accuracy it takes $O(w(m))$ to solve a nonlinear system to the same precision.

6.4 Controlled Precision

In case of zero interval width the previous method would require exact precision. In this section we combine both multiple and variable interval precision methods into a controlled precision method. We also discuss practical implementation aspects for some operators.

We can combine definitions (6.1) and (6.5) to obtain a m-digit controlled variable interval precision representation. We simply define $P_{l,u} = b^{p_c}$ where $p_c = \max(p_v, p_f)$:

$$X^\circ = [\lfloor \frac{X_l}{P_l} \rfloor P_l\,,\, \lceil \frac{X_u}{P_u} \rceil P_u]$$

with the assumption that zero bounds are preserved. Note that when $w(X) = 0$, $X^\diamond$ has a m-digit finite representation.

Example 6.8

Using again the interval of example 6.1, $b = 10$ and $\lambda_M = 1$, for $m = 3$ we now have:

$$X^\diamond = [1.23\,,\, 1.24]$$

which is the same as the fixed precision interval representation of X. On the other hand for $m = 5$ we have:

$$X^\diamond = [1.234\,,\, 1.236]$$

corresponding to the variable interval representation.

By controlling the precision of a nonlinear solution operator by $w(X)^2$, quadratic convergence is maintained. Indeed, every intermediate computation satisfies either theorem 2 or theorem 3.

Example 6.9

Application of this method to example 6.3 with $\lambda_\Psi = 10^{-1}$ and $\lambda_M = 10^{-1}$ produces the following iterates:

iteration	digits	X	Y
0	2	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	2	[+0.3v0 +0.7v0]	[+0.7v0 +0.9v0]
2	2	[+0.4v0 +0.6v0]	[+0.84v0 +0.90v0]
3	3	[+0.48v0 +0.51v0]	[+0.863v0 +0.871v0]
4	5	[+0.4998v0 +0.5002v0]	[+0.8660v0 +0.8661v0]
5	9	[+0.49999998v0 +0.50000002v0]	[+0.86602539v0 +0.86602541v0]
6	17	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

As compared to example 6.4 and example 6.7 one more iteration was required. Indeed, this approach uses the minimum required precision to maintain the convergence.

Not only does controlled precision avoid infinite representations associated with $w(X) = 0$, it also allows us to take the actual size of the domain into account. This can be done by imposing a lower bound on the precision in terms of $w(X)$. While it will have no effect on the convergence rate, it can be important for the accuracy of initial iterates. Indeed, by ensuring that sufficient precision is used, it will prevent bisection from taking place due to rounding errors. This additional control mechanism is also useful when the function to be solved is interval valued.

If we consider the specific structure of the solution operators we can reduce required precision even further. To that end, let us consider the nonlinear operators discussed in 3:

$$\Psi(f, X) = X_c - \Lambda(CJ(X), Cf(X_c), \Delta X) \tag{6.9}$$

As already indicated in [166] it is sufficient to compute an approximate midpoint preconditioner $C \approx J(X)_c^{-1}$, for if:

$$d(C^\diamond\,,\, J(X)_c^{-1}) = \mathcal{O}(w(X))$$

the quadratic convergence will not be affected. In addition, if the precision is controlled by $w(X)$, we know that $d(J^\diamond(X)\,,\, J(X)) = \mathcal{O}(w(X))$. So that the multiplication $A = C^\diamond J^\diamond(X)$

can be computed using the same precision.

The accuracy of $f^\circ(X_c)$ on the other hand is required to be $\mathcal{O}(w(X)^2)$, but as $f(X_c) = \mathcal{O}(w(X))$ the actual multiplication can again be performed at the lower precision [263]:

$$d(B^\circ\,, C^\circ f^\circ(X_c)) = \mathcal{O}(w(X)^2) \qquad .$$

Finally, one can view equation (6.9) as an improvement iteration of an approximate solution. In fact, some multiple precision linear solvers are based on this technique [20]. The associated error analysis [264] reveals that if Λ is computed with an accuracy of $w(X)$, the above assumptions imply:

$$d(\Psi^\circ(f, X)\,, \Psi(f, X)) = \mathcal{O}(w(X)^2)$$

so that only $f(x_c)$ and the final addition needs to be computed with precision p_f controlled by $w(X)^2$. Let us illustrate this method with an example.

Example 6.10

Consider again the system of example 6.3. As in example 6.9 we use controlled precision, but now with $\lambda_M = 10^{-3}$ so that the precision of the computation will essentially be controlled by p_f:

iteration	digits	X	Y
0	4	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
1	4	[+0.3v0 +0.7v0]	[+0.7v0 +0.9v0]
2	4	[+0.46v0 +0.54v0]	[+0.85v0 +0.89v0]
3	6	[+0.498v0 +0.502v0]	[+0.8657v0 +0.8663v0]
4	8	[+0.499997v0 +0.500003v0]	[+0.8660250v0 +0.8660258v0]
5	14	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

Notice that initially the convergence is slower. In fact we had to increase λ_Ψ to 10^{-2} to ensure the quality of the first iterates. It also takes a little longer before the quadratic convergence sets in. The number of digits shown above is p_f, but as mentioned earlier, only the function evaluation and the final addition is computed at that precision. The remaining calculations are done at $p_f/2$.

As both $f(X_c)$ and $J(X)$ or $S(X)$ are required, the methods described in chapter 5 can be used to compute them efficiently. As a byproduct, one obtains the local partial derivatives of the computational graph, which represent the sensitivity of the computation w.r.t. to the accuracy of the different operators [108, 77]. This provides additional information to control the precision for each operator, as it allows us to locally adjust the precision to the sensitivity of the computation. Although this optimization will not affect the rate of convergence, it can improve the conditioning of the computation. Indeed, as mentioned earlier, the accuracy of $f(X_c)$ is very important. Note however that in order to apply this approach for the optimization of the accuracy of the partial derivates themselves, additional information is required.

Finally, one should keep in mind that both the accuracy of the midpoint preconditioner and the solution of the interval system depend on the corresponding condition number. Although, algorithms for estimation matrix condition numbers are available [175, 84, 99], simpler techniques can be used for this particular problem. Since the main reason for using preconditioners is to improve the performance of the solution techniques, we should only be concerned with the accuracy of the midpoint preconditioned matrix A. The distance $d(A_c, I)$ is in fact a good measure of the accuracy of the preconditioning operation [264]. Starting with a low precision, one can therefore verify the resulting accuracy and estimate the required relative precision using theorem 2.

Estimation of the condition number associated with the interval matrix A on the other hand is relatively easy. As indicated in chapter 4, the exact solution requires inversion of $\langle A \rangle$. Since this is an M-matrix we have [18]:

$$I - r \leq \langle A \rangle \Rightarrow \langle A \rangle^{-1} \leq (I - r)^{-1}$$

where $r_{ij} = w(A)/2$. Furthermore provided $2 - nw(A) > 0$ we have:

$$\begin{aligned} (I - r)^{-1}_{ii} &= \frac{2 - (n-1)w(A)}{2 - nw(A)} \\ (I - r)^{-1}_{ij} &= \frac{w(A)}{2 - nw(A)} \qquad \textit{for } i \neq j \end{aligned}$$

so that:

$$\|\langle A \rangle\|_\infty \|\langle A \rangle^{-1}\|_\infty \leq \|I - r\|_\infty \|(I - r)^{-1}\|_\infty \leq K$$

with:

$$K = \frac{2 + nw(A)}{2 - nw(A)}$$

If for example Crout's algorithm is used to compute $\langle A \rangle^{-1}$, one therefore needs to reserve an additional $\mathcal{O}(\log K + \log n^2)$ computed using digits to accommodate for rounding errors [263, 20]. Observe that the above reasoning can also be used for the computation of the condition numbers proposed by [46, 202].

Chapter 7

Design Computations

7.1 Introduction

In this chapter we illustrate how interval arithmetic can be used for computations with bounded models. We have already indicated that conservative computations can provide the reliability required for automation. More importantly, bounded models allow us to address the consistency problem by considering sets of solutions. While the notion of semantic consistency can be used to extend the automation of parametric design, procedural consistency will guarantee that model refinement does not invalidate the decision process even if heuristic models are used initially. Moreover, since each part of the computation has conservative bounds on partial results computed elsewhere, the use of bounded models allows for asynchronous distributed concurrent computation. Both aspects are particularly useful for the design of large scale systems where refinements are made in a distributed fashion, computations are performed at different speeds and communication delays are unpredictable.

The basic features of interval computations in design are illustrated in section 7.2. In section 7.3 we discuss the hierarchical design methodology and the associated consistency problem in the context of interval arithmetic. A practical implementation of interval computations for distributed concurrent computations is presented in section 7.4. Finally, section 7.5 contains a number of concrete proposals for the Design Society.

7.2 Bounded Models in Design

As discussed in chapter 2, bounded models conservatively enclose sets of solutions. Interval arithmetic, by definition, will not leave out candidate solutions, unless it is guaranteed that they cannot satisfy the equations. Since initial design models typically can be represented by interval equations, nonlinear solution techniques described in this thesis can be used to enclose all design solutions.

Example 7.1

To illustrate the use of interval arithmetic in design we borrow a classic engineering design problem from [147]. The symmetric two bar truss, shown in figure 7.1, has a height h, a span $2b$ and is subjected to a load $2p$. The truss members are made of tubular steel, with

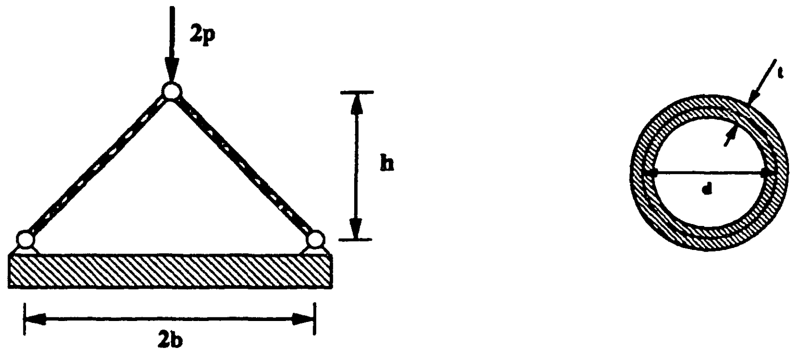


Figure 7.1: Symmetric tubular two bar truss

diameter d and thickness t. The problem is to select h and d so that the weight of the structure is minimized. The length l of each member is given by:

$$l = \sqrt{b^2 + h^2}$$

while the weight w of the structure is:

$$w = 2\rho\pi dtl$$

where ρ is the density of the material. The compressive stress in the members σ_c can be written as:

$$\sigma_c = \frac{pl}{\pi thd} \tag{7.1}$$

and the Euler buckling stress σ_e is given by:

$$\sigma_e = \frac{\pi^2 E(d^2 + t^2)}{8l^2} \tag{7.2}$$

Based on monotonicity analysis [181], one can easily verify that for this optimization problem both constraints are active. The solution therefore can be found by solving equations (7.1) and (7.2) for h and d, while the stresses are taken to be the ultimate material strength σ_{max}. The material properties of steel are:

$$\begin{aligned} \rho &= 8000\,\text{kg/m}^3 \\ E &= 200\,\text{GPa} \\ \sigma_{max} &= 400\,\text{MPa} \end{aligned}$$

and the input parameters are defined to be:

$$\begin{aligned} p &= 150\,\text{kN} \\ b &= 1\,\text{m} \\ t &= 2.5\,\text{mm} \end{aligned}$$

Furthermore, we are only interested in the following solution domain:

$$\begin{aligned} D &= [25\,, 75]\,\mathrm{mm} \\ H &= [75\,, 150]\,\mathrm{cm} \end{aligned} \tag{7.3}$$

Using the above numerical values, we obtain the following system of equations:

$$\begin{aligned} 20\pi dh - 3\sqrt{h^2+1} &= 0 \\ 4(h^2+1) - \pi^2(250d^2 + 1/640) &= 0 \end{aligned} \tag{7.4}$$

which can be solved using the solution techniques presented in chapter 3. With a midpoint preconditioned Gauss-Seidel operator and interval slopes enclosures, we have the following iterates:

iteration	D	H
0	[+2.v-2 +8.v-2]	[+0.7v0 +1.5v0]
1	[+5.v-2 +8.v-2]	[+0.7v0 +1.5v0]
2	[+5.8v-2 +6.6v-2]	[+1.15v0 +1.24v0]
3	[+6.23v-2 +6.26v-2]	[+1.185v0 +1.188v0]
4	[+6.243610v-2 +6.243616v-2]	[+1.1868042v0 +1.1868045v0]
5	[+6.243613215v-2 +6.243613216v-2]	[+1.186804367v0 +1.186804368v0]

Although we are only imposing indirect requirements on w, namely that it has to be minimal, the resulting weight is of interest. For the above solution, we find:

$$W = [12.17641932\,, 12.17641933]\,\mathrm{kg}$$

This information can therefore be considered as a view (see chapter 2) of the design model. We would like to emphasize that while these are accurate solutions of the above mathematical model, one still has to keep in mind that the model itself is only an approximation to the physical one.

In case the value of the load is not yet precisely known:

$$P = [140\,, 160]\,\mathrm{kN} \tag{7.5}$$

some of the coefficients of equation (7.4) will be intervals. Although the exact solution in this case is a set, one can use the same solution methods to obtain:

$$\begin{pmatrix} D \\ H \end{pmatrix} = \begin{pmatrix} [0.0595\,, 0.0654] \\ [1.092\,, 1.282] \end{pmatrix} \tag{7.6}$$

meaning that irrespective of the exact value of the load p given by equation (7.5), the dimensions of the minimum weight structure will not exceed the bounds of equation (7.6).

As the previous example illustrates, some optimization problems can directly be solved using the techniques presented in this thesis. For optimization specific interval techniques however, we refer the reader to [194].

A number of theoretical results concerning the enclosure of the solution of nonlinear equations of the interval type are presented in [129, 136, 133, 135, 68, 132, 131]. We would like to point out that for the global solution of this type of equations some modifications to the bisection strategy are in order. Indeed, the exact solution of interval functions can have nonzero width and it is therefore difficult to impose stopping criteria based on the width of the iterate. Instead, we propose to stop bisecting as soon as the Lipschitz matrix is regular and to use a threshold criterion when it is singular. In fact, the algorithm described in chapter 4 can directly provide this information together with the solution enclosure. Nonlinear solution iterations are then terminated when the decrease in width itself is becoming too small or sufficient accuracy is obtained.

We have already mentioned that interval methods are not limited to functions and can for example be used for point enumerations as well. Indeed, as long as interval bounds are provided, nonlinear solution methods will conservatively enclose all possible solutions. It should however be observed that while for continuous functions we still can prove existence, this is no longer true for discrete point sets. Let us illustrate this idea with an example.

Example 7.2

Consider the beam shown in figure 7.2 on the left. The problem is to determine the height

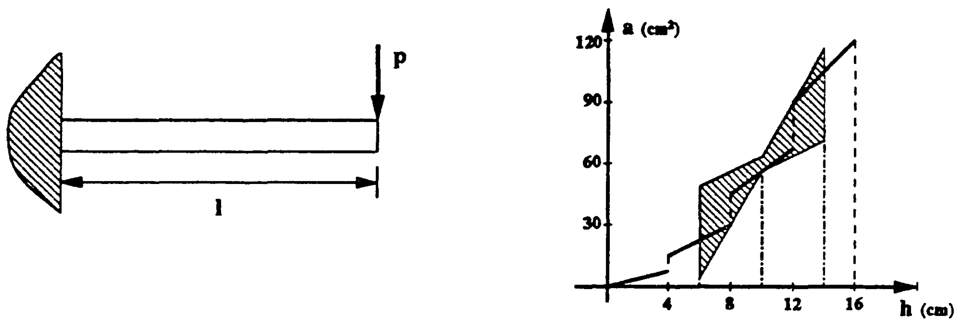


Figure 7.2: Beam Selection by Enclosure

h and width b of the rectangular cross-section, given the length l and the load p. Assuming that yield is determined by the maximum bending stress, we have:

$$ah - \frac{6pl}{\sigma_{max}} = 0 \qquad (7.7)$$

where a is the area of the cross-section and σ_{max} is the maximal allowable stress in the beam. Now let us now consider the case where beams only come in certain widths depending on the required height. As is illustrated in figure 7.2 on the right, a is therefore a piecewise linear function of h. In this example we use the bounds on b associated with each height interval as generalized interval slopes S_h^a, and then determine the centers so as to conservatively enclose the function. The slope of ah can now simply be computed by:

$$S_h^{ah} = a_c + H S_h^a$$

Using the same material and nonlinear operator as in example 7.1, the following configuration:

$$\begin{aligned} p &= 40\,\text{kN} \\ l &= 1\,\text{m} \end{aligned}$$

produces the iterates:

iteration	H
0	[0 +0.2v0]
1	[+8.v-2 +0.16v0]
2	[+8.v-2 +0.12v0]
3	[+0.1031v0 +0.1037v0]
4	[+0.1032794v0 +0.1032798v0]
5	[+0.1032795558v0 +0.1032795559v0]

where H is given in m. The particular width corresponding with this choice of h is:

$$b = 0.01875\,\mathrm{m}$$

Finally, the interval techniques discussed in chapter 3 can be used to overcome the robustness properties of existing solution techniques in the area of geometry. Indeed, unless computations are performed conservatively, algorithms based on approximate values [138] can never provide the 100% reliability of interval arithmetic.

Example 7.3

Consider for example the computation of the critical points of Geisow's multiple crunode equation [188]:

$$g(x,y) = -6x^4 + 21x^3 - 19x^2 - 6x^2y^2 + 11xy^2 + 3y^2 - 4y^4 \tag{7.8}$$

which are defined to be the solutions of:

$$\begin{aligned} \frac{\partial g}{\partial x} &= -24x^3 + 36x^2 - 38x - 12xy^2 + 11y^2 &= 0 \\ \frac{\partial g}{\partial y} &= -12x^2y + 22xy + 6y - 16y^3 &= 0 \end{aligned}$$

The corresponding curves in the domain:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} [-1,3] \\ [-2,2] \end{pmatrix}$$

are shown in figure 7.3. For this problem, we can simply use the techniques of chapter 3 to obtain:

X	Y
[+0.2043643892v0 +0.2043643897v0]	[-0.7903653794v0 -0.7903653792v0]
[+0.2043643892v0 +0.2043643897v0]	[+0.7903653792v0 +0.7903653794v0]
[+0.9999999998v0 +1.000000001v0]	[-1.000000001v0 -0.9999999999v0]
[+0.9999999998v0 +1.000000001v0]	[+0.9999999999v0 +1.000000001v0]
[+1.345635610v0 +1.345635611v0]	[-0.9312344320v0 -0.9312344319v0]
[+1.345635610v0 +1.345635611v0]	[+0.9312344319v0 +0.9312344320v0]
[-7.v-15 +7.v-18]	[-2.v-16 +2.v-16]
[+0.9392401486v0 +0.9392401487v0]	[-3.v-17 +3.v-17]
[+1.685759851v0 +1.685759852v0]	[-3.v-15 +3.v-15]

up to a 10-digit accuracy. Observe that while the roots of this equation can readily be computed by traditional methods, the bounds given above are provably correct.

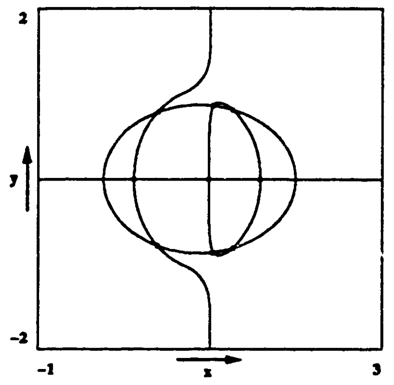


Figure 7.3: Graphic interpretation of Equations (7.8)

7.3 Hierarchical Computations

As indicated in chapter 2, the specification of functional requirements defines a hierarchy of design models. High level models are associated with dominant characteristics, while lower level models represent more detailed information. The relative importance of the different requirements is decided by the designer based on their sensitivity w.r.t. the overall design. This idea is illustrated in figure 7.4. In this case, the gross shape of the design solution is

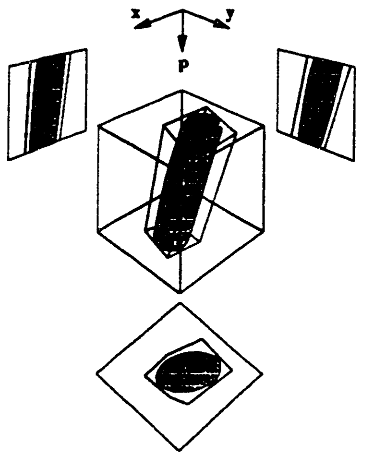


Figure 7.4: Hierarchical Definition of Functional Requirements based on Sensitivity

essentially determined by x and y, whereas only very little information can be obtained by considering p.

Our framework allows the designer to incorporate the uncertainty on lower level variables by leaving some of the modeling information unspecified. More specifically, we can associate nonlinear interval equations to high level models and use the interval techniques proposed in this thesis for enclosing their solution. Additional requirements can be imposed on lower level models, which are allowed to depend on the solution of the higher level models in a parametric fashion. Finally, lower level solutions can be used to refine the higher level equations.

This approach is illustrated in figure 7.5. Models depend parametrically on enclosures

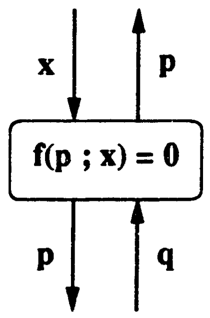


Figure 7.5: Hierarchical Modeling Relations

of higher level solutions x and determine the model variables p. This information is not only used in a parametric fashion by lower level models, but also to refine the equations associated with higher level models. Similarly, lower level models provide refinement information q, so that the uncertainty on the interval coefficients of f can be reduced. Furthermore, weak interactions between models at the same level of hierarchy can be decoupled by providing interval bounds on the coupling variables.

Semantic and procedural consistency (chapter 2) ensure that the refinement process does not allow for additional solutions. When both the parametric and refinement dependencies are explicitly known, semantic consistency is automatically verified by the inclusion monotonicity property of interval arithmetic (see example 7.10). Unfortunately, when bounds are provided in a heuristic fashion, semantic consistency is no longer ensured. In this case, the validity of model refinements can however be verified by procedural consistency checks.

Two types of interval enclosure checks can be used to verify procedural consistency. The first one is simply to ensure that refined values of design parameters are enclosed in the original ones. This method therefore simply views models as intervals enclosing the solution of a system of equations. The second one is to allow for hierarchical refinement of parts of the nonlinear equations by verifying that the enclosures used in the nonlinear solvers can not introduce additional solutions. Consider to this end the i-th equation of the mean value form defined in chapter 3:

$$f_i(X_c) + F_i'(X)(X - X_c) = 0 \tag{7.9}$$

To be a refinement of this function enclosure, all values allowed by the refined bounds should be enclosed by the mean value bounds. This is illustrated in figure 7.6. When the bounds used in the computation contain their refinements, the solutions will be enclosed too. However, when conservative solution bounds are computed, this inclusion is only guaranteed if the interval coefficients of the approximate equations are enclosed. Indeed, by the inclusion monotonicity property we know that conservative interval enclosures to the solution will be consistent if the following inclusions are verified:

$$\begin{aligned} f_i(X_c^{(j+1)}) + F_i'(X^{(j+1)})(X_c^{(j)} - X_c^{(j+1)}) &\subseteq f_i(X_c^{(j)}) \\ F_i'(X^{(j+1)}) &\subseteq F_i'(X^{(j)}) \end{aligned} \tag{7.10}$$

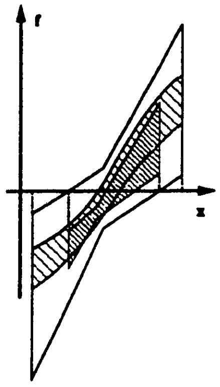


Figure 7.6: Procedural Consistency

The superscript $j + 1$ indicates the quantities related to the refined model, while j corresponds with the original model.

Note that procedural consistency can be enforced by maintaining previous models and intersecting refined bounds with the existing ones. While this would ensure the validity of the decisions, it changes the meaning of the modeling equations. Furthermore, as the original models are likely to be less accurate than the refined ones, overlapping intervals are an indication of the poor quality of the high level models.

7.4 Concurrency in Design

The model used for concurrent computations in design needs to satisfy the following requirements. Since the design process is incremental, the computational model and the associated design graphs should be able to incorporate incremental topological state changes. In addition, it is required that the processes themselves can be modified. Not only does this allow us to incrementally refine existing models and to add new ones, but in addition it gives us the possibility to control consistency in an asynchronous environment. Scalability, which is the guarantee of effectiveness irrespective of the size of the problem was identified by [117] as a crucial property for any distributed problem solving architecture. We therefore favor an object oriented scalable model, so that the complexity of the proposed design systems is allowed to increase arbitrarily. Finally, it should allow for concurrent asynchronous computations so that it accurately reflects real world situations.

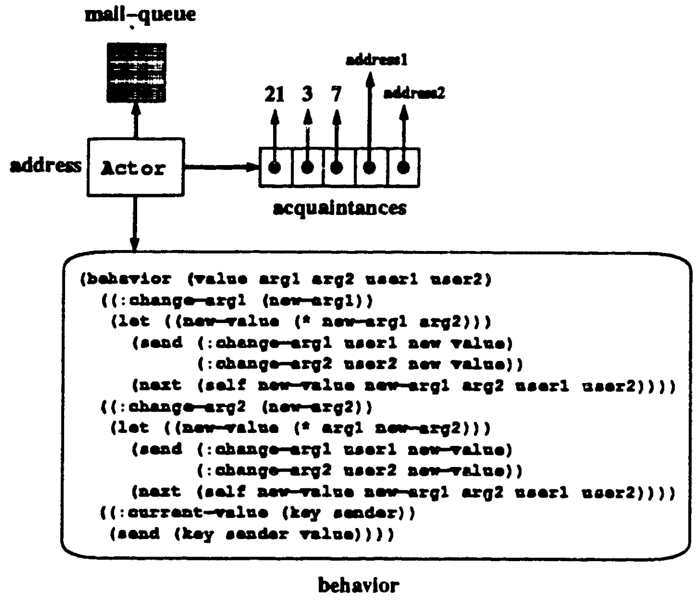
Since the Actor model of computation satisfies the above requirements, we have decided to use it as a basic representation of concurrent computations.

7.4.1 Actor Model of Computation

As indicated in chapter 5, computational graphs are a very powerful tool to represent the data dependencies in computations. However, the flow of the computation is usually controlled by continuations. But as indicated in [94], continuations can be viewed as patterns of passing messages on a graph. In fact, it was this observation that originally led to the concept of actors [1].

Actor graphs constitute a distributed representation of the design process and the associated design model. They therefore define the meaning of design computations, much as dataflow graphs define the semantics for parallel languages [12, 13].

Each actor has an address, a behavior and a number of acquaintances (see figure 7.7). Actors can receive and send messages, create other actors and specify a replacement behavior



```
(behavior (value arg1 arg2 user1 user2)
  ((:change-arg1 (new-arg1))
   (let ((new-value (* new-arg1 arg2)))
     (send (:change-arg1 user1 new value)
           (:change-arg2 user2 new value))
     (next (self new-value new-arg1 arg2 user1 user2))))
  ((:change-arg2 (new-arg2))
   (let ((new-value (* arg1 new-arg2)))
     (send (:change-arg1 user1 new-value)
           (:change-arg2 user2 new-value))
     (next (self new-value new-arg1 arg2 user1 user2))))
  ((:current-value (key sender))
   (send (key sender value))))
```

Figure 7.7: Representation of Actors

with new acquaintances. This allows not only for state changes, but also for topological changes in the computational graph.

The examples below were implemented on a simulated Actor model. This model contains a number of physical devises, such as `Machines`, `Terminals`, `Mouses` and `Printers`, which are connected by a `Network`. The actual execution of the operators takes place in the machine processors. Each machine has a queue of tasks, where:

$$task = \langle actor + message \rangle$$

and processes a number of tasks per cycle. Furthermore, actors are allocated on different machines by a load balancing algorithm. Other devices, such as `Terminal`, `Mouse` and `Printers` have dedicated special actors that handle a standard message protocol, e.g. `:read` and `:print`. We have also included a facility to simulate different types of delays on the network.

For an introduction to programming Actor systems we refer the reader to [154] and [155], while a more detailed treatise of the compilation aspects is given in [153]. A recent overview of the work in this area can be found in [98]. Note that we have deliberately chosen not to use the ABCL [271] nor the Smalltalk [101] approach, as both are subsumed by the Actor paradigm.

7.4.2 Design Networks

Incremental modification

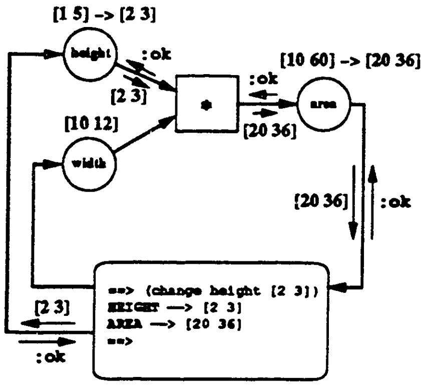
To represent data dependencies with actor graphs, we associate actors with arithmetic operators, variables and constants. Each of these actors knows the addresses of the actors which depend on their results. So we simply need to instruct them to recompute their result and send it to their dependents as soon as input changes occur (see figure 7.7). Procedures are also represented by actors and function application corresponds with sending a :connect message to the procedure actor, who in turn will create the appropriate computation graph given the addresses of the actors representing the arguments.

Example 7.4

Let us consider the following simple computation of the area of the cross-section of a beam:

```
==> (define beam-area
      (lambda (beam-height beam-width)
        (* beam-height beam-width)))
AREA
==> (define height [1 5])
HEIGHT
==> (define width [10 12])
WIDTH
==> (define area (beam-area height width))
AREA
==> area
[10 60]
==> (change height [2 3])
HEIGHT -> [2 3]
AREA -> [20 36]
```

Changing the value of height now triggers a sequence of :change messages on the computational graph shown in figure 7.8. The corresponding parallelism profile is shown in



```
==> (change height [2 3])
HEIGHT ---> [2 3]
AREA ---> [20 36]
==>
```

Figure 7.8: Computational graph of Example 7.4

figure 7.11.

It should be noted that propagating changes through a computational network requires some synchronization. Indeed, since modification messages are not guaranteed to arrive in sequence, different actors might otherwise have a different value for the same quantity. As

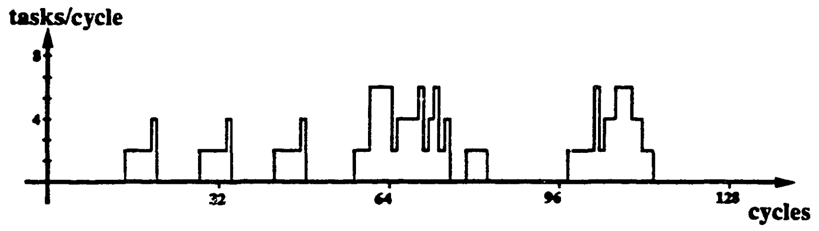


Figure 7.9: Parallelism Profile of Example 7.4

illustrated in figure 7.8, this can be done by extending the behavior shown in figure 7.7 so that incoming messages are buffered with insensitive actors until the acknowledgment :ok message is received.

Observe that frequently used quantities will have to maintain large lists of dependents. Fortunately, this can be alleviated by a hierarchical organization of agencies as illustrated in figure 7.22. In this case, only one update for the entire agency is required. A different approach would be to have computations ask for the latest available information.

Actor graphs can also change topologically, this occurs for example when replacement behaviors have different acquaintances. Runtime changes of model definitions in concurrent systems do however require a number of additional safeguards since computations are performed while the changes occur. To ensure consistency, modeling changes therefore prompt the actors of the old model to become insensitive to further changes, and buffer the incoming messages, so that no inconsistent changes are passed on to dependents. At this point, the consistency of the new model can be verified and the new graph inserted. Only then are we allowed to replace the insensitive behavior of associated actors by the newly defined ones or by forwarders to the inserted model. Let us illustrate this idea with an example.

Example 7.5

When changing the `lambda` expression associated with the model, for example, incoming communications can simply be buffered by the environment actor.

```
==> (define model (lambda (x y) (+ (* [0 6] x) (* [3 4] y))))
MODEL
==> (define a (model [1 2] [5 6]))
A
==> A
[15 36]
==> (change model (lambda (x y) (+ (square x) (* 3 y))))
MODEL
A -> [16 22]
```

So in this example intermediate changes to both `x` and `y` are buffered by the local environment created by the `lambda` expression and sent to the new graph as soon as it is created. The associated topological change is illustrated in figure 7.10, while the parallelism profile is shown in figure 7.11. Note that topological changes require that the number of dependents of referenced parameters be updated as well, so that old models can be garbage collected.

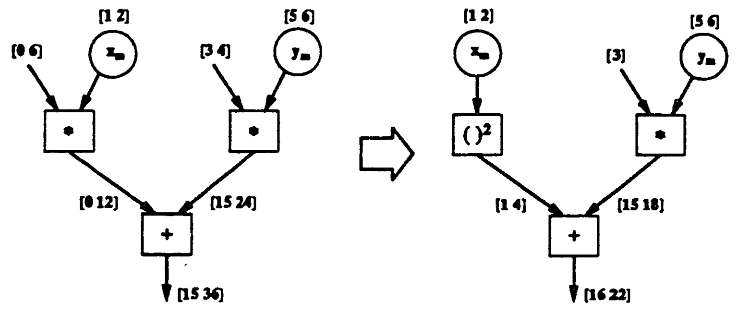


Figure 7.10: Topological Changes of the Computational Graph of of Example 7.5

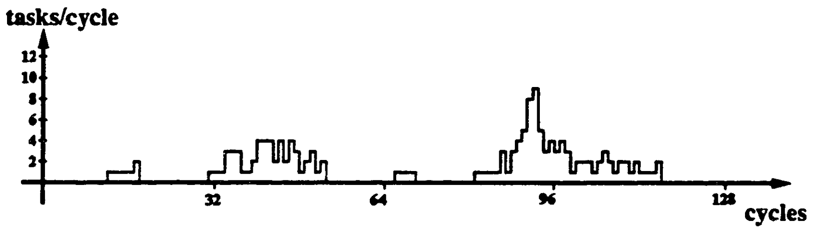


Figure 7.11: Parallelism Profile of Example 7.5

Concurrent Interval Computations

The key element for allowing asynchronous design computations is the conservative nature of interval arithmetic. In addition, late arriving information can only further constrain the design space. The inclusion isotonicity property therefore guarantees that interval bounds can only improve.

One should however make sure that all available enclosures are compatible for solution by linear interval solvers. One simple strategy based on equation (7.9) is to center all equations around the oldest center X_c°:

$$f_i(X_c^{(j)}) + F_i'(X^{(j)})(X_c^\circ - X_c^{(j)}) + F_i'(X^{(j)})(X - X_c^\circ) = 0$$

This operation is illustrated in figure 7.12.

The accompanying assumption is that the oldest information has the largest interval width and that it is preferable not to worsen the corresponding bounds by shifting to a different center. Obviously, it is not necessarily true that the oldest information is constraining and other strategies can easily be devised based on the quality of the associated enclosures.

Although for arithmetic expressions the cost of computing the derivative is very similar to the cost of computing the center (see chapter 5), it is not required that this information be used synchronously. Indeed, interval derivatives are valid over the entire interval and can be used in conjunction with any center, as long as it belongs to the corresponding domain. When concurrent computations are performed using interval slopes however, one should take the dependence of the slopes on the center into account. Results of previous iterations can therefore not directly be used in conjunction with a different center. This

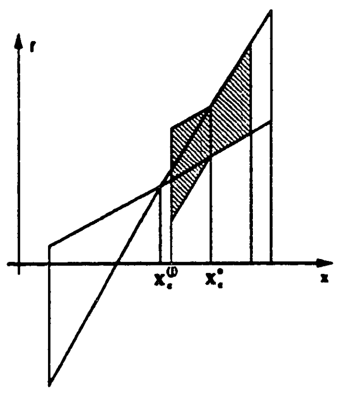


Figure 7.12: Center Shifting for Concurrent Interval Computations

implies that there is a trade-off between synchronization requirements on the one hand and the improvement in accuracy over interval derivatives on the other. In addition, interval derivatives are inclusion monotonic, while this is not necessarily true for slopes. Unlike interval derivatives, it is therefore not allowed to intersect intermediate slopes.

In the examples below we have implemented a special actor to represent the linear solution operator. This implies that the operation count for each iterate it produces is one. Although this is not very realistic, it allows us to illustrate the computations in the network. Furthermore, the problem of solving linear systems in parallel is relatively well understood.

Example 7.6

To illustrate the concurrent solution of equations, we have implemented this approach for equations (3.27). The computational graphs are shown in figure 7.13 and numerical results

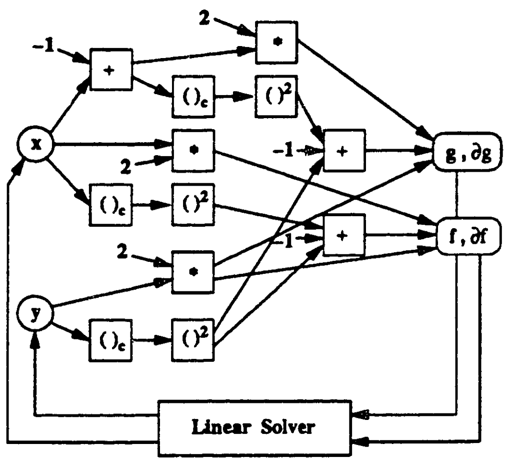


Figure 7.13: Computational Graph for Example 7.6

are as follows:

function	X	Y
f	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
g	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
f	[+0.3v0 +0.7v0]	[+0.81v0 +0.90v0]
g	[+0.3v0 +0.7v0]	[+0.81v0 +0.90v0]
f	[+0.4v0 +0.6v0]	[+0.83v0 +0.90v0]
g	[+0.49v0 +0.51v0]	[+0.864v0 +0.868v0]
f	[+0.498v0 +0.502v0]	[+0.865v0 +0.867v0]
g	[+0.4999v0 +0.5001v0]	[+0.86599v0 +0.86606v0]
f	[+0.4999998v0 +0.5000002v0]	[+0.8660253v0 +0.8660255v0]
g	[+0.49999997v0 +0.50000003v0]	[+0.86602538v0 +0.86602542v0]
f	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]
g	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

Figure 7.14 shows the corresponding parallelism profile. Observe that whereas initial it-

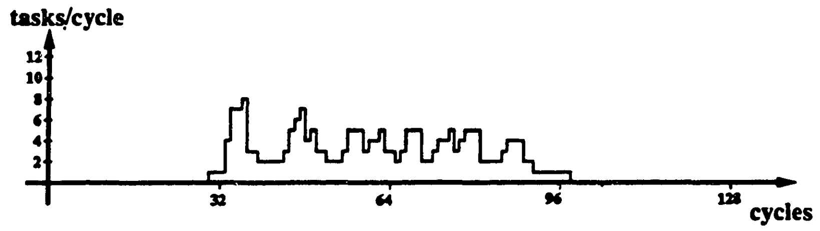


Figure 7.14: Parallelism Profile of Example 7.6

erations are clearly distinguishable in the parallelism profile, this is no longer true at the end. This is due to the small difference in critical path for the two functions. The apparent slower convergence is related to the fact that we have constrained the solution operator to incorporate only one change per iteration as indicated in the first column.

When all local equilibrium conditions are satisfied, we say that *Design Equilibrium* is reached (see chapter 2). We now illustrate the effect of limited computational resources and communication delays on concurrent interval computations.

Example 7.7

The increase in execution time for a parallelism bound of 2 is shown in figure 7.15. However,

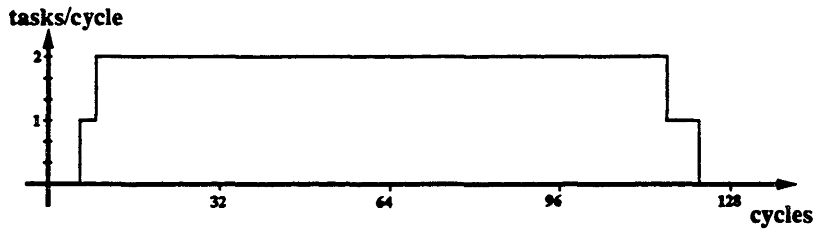


Figure 7.15: Parallelism Profile of Example 7.7 with Bounded Parallelism

as pointed out in [13], additional delays are not as detrimental as compared to the unbounded parallelism case (see figure 7.16). Indeed, the increase in the computation time due to a unit delay in the bounded parallelism case (see figure 7.17) is not as significant. So relatively speaking, the price of additional communication delays with bounded parallelism

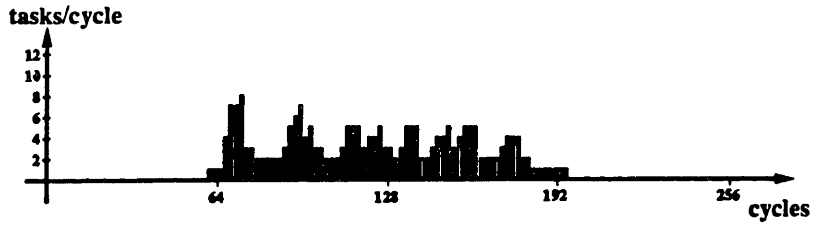


Figure 7.16: Parallelism Profile of Example 7.7 with a Unit Delay and Unbounded Parallelism

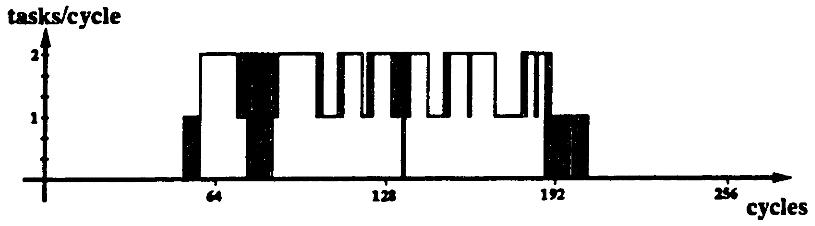


Figure 7.17: Parallelism Profile of Example 7.7 with Unit Delays and Bounded Parallelism

is very small, indicating a natural compromise between the speed of computations and the delay.

To illustrate the effect of different processor speeds we consider two machines, each with different parallelism bounds.

Example 7.8

Consider again the problem of example 7.6 but now with a two machine configuration. The parallelism of machine 1 is unbounded while for machine 2 it is bounded to 1. The interval solver is allocated on machine 1, so that it would not be a constraining factor. The first equation f, which also has a shorter critical path was also allocated on the faster machine, while the second one g was allocated on machine 2. The effect of the different processor speeds on the parallelism profiles is shown in figure 7.18. Notice that after three iterations the solver decided that the solution improvement was too small for further iterations with the function f. This can also be seen in the corresponding iterates:

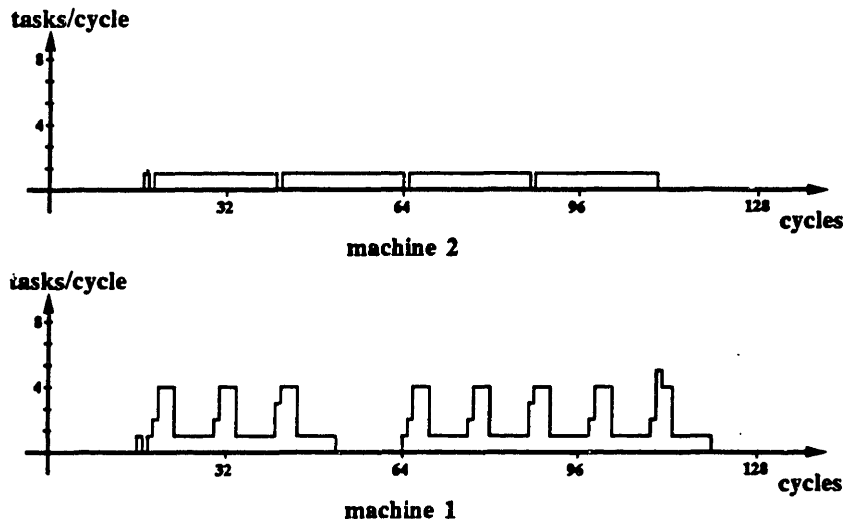


Figure 7.18: Parallelism Profile of Example 7.8

function	X	Y
f	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
g	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
f	[+0.3v0 +0.7v0]	[+0.81v0 +0.90v0]
f	[+0.4v0 +0.6v0]	[+0.83v0 +0.90v0]
g	[+0.4v0 +0.6v0]	[+0.83v0 +0.90v0]
f	[+0.4v0 +0.6v0]	[+0.84v0 +0.90v0]
g	[+0.496v0 +0.504v0]	[+0.865v0 +0.867v0]
f	[+0.498v0 +0.501v0]	[+0.865v0 +0.867v0]
f	[+0.499v0 +0.501v0]	[+0.865v0 +0.867v0]
g	[+0.49999v0 +0.50001v0]	[+0.866022v0 +0.866029v0]
f	[+0.4999999v0 +0.5000001v0]	[+0.86602537v0 +0.86602544v0]
f	[+0.49999997v0 +0.50000003v0]	[+0.86602538v0 +0.86602542v0]
g	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]
f	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

For each iterate for g approximately two iterations were performed for f.

Since the computations are represented as actor graphs, parametric changes can occur at any moment. In particular, they can disturb local equilibrium conditions by providing more accurate information. Let us illustrate this point with an example.

Example 7.9

To demonstrate how simple parameter changes trigger new refinements, we consider the following equation:

$$\begin{aligned} ax^2 + by^2 - r_f &= 0 \\ a(x-1)^2 + by^2 - r_g &= 0 \end{aligned} \tag{7.11}$$

where $a = b = [1]$ and originally $R_f = R_g = [0.95, 1.05]$.

We now first change $r_f = [1]$, wait until equilibrium is reached and then proceed to change $r_g = [1]$.

function	X	Y
f	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
g	[+0.2v0 +0.8v0]	[+0.6v0 +0.9v0]
	⋮	⋮
g	[+0.2v0 +0.8v0]	[+0.7v0 +0.9v0]
f	[+0.2v0 +0.8v0]	[+0.7v0 +0.9v0]
	⋮	⋮
g	[+0.47v0 +0.53v0]	[+0.84v0 +0.89v0]
f	[+0.47v0 +0.53v0]	[+0.85v0 +0.89v0]
	⋮	⋮
g	[+0.4999999998v0 +0.5000000002v0]	[+0.8660254037v0 +0.8660254039v0]
f	[+0.4999999999v0 +0.5000000001v0]	[+0.8660254037v0 +0.8660254038v0]

The corresponding parallelism profile is shown in figure 7.19. Note that the time at which

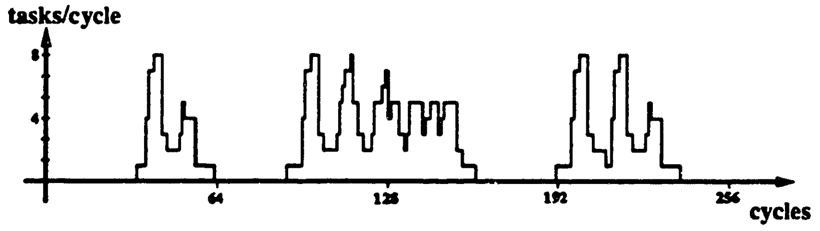


Figure 7.19: Parallelism Profile of Example 7.9

the parametric changes are initiated is immaterial, for if they would have taken place before the equilibria were reached, we would still have convergence to the same final solution.

In general, large distributed systems can exhibit a very wide range of behaviors [105]. Although the computations proposed here provide verified accuracy, we found that the propagation of changes can potentially lead to an increasing number of messages being sent. Also, as this type of design networks are no longer acyclic, it is possible to find quasi periodic behavior where a large number of messages is sent in cycles. This phenomenon can however be controlled by a careful implementation of the solution actor, for example by incorporating changes only by synchronous local iteration loops. So that a maximum of one local iteration per function is allowed. In addition, pile-up of messages corresponding to parametric changes can be avoided by merging. Indeed, except for midpoint and slope computations, interval refinement messages may be intersected. Furthermore, if the interval remains unchanged it is not required to propagate the change any further.

Hierarchical Design Networks

When concurrent computations associated with design models are organized in a hierarchical fashion, both parametric changes and model refinements can take place.

Example 7.10

In this example we illustrate a two level hierarchical approach where in addition to equations (7.11), which make up the top level of the hierarchy, we consider the following lower level model:

$$\begin{aligned} a^2 + 16a + 16b &= \frac{32}{29+16x^2y^2} + 32 \\ (11 + 2x)a - 12b^2 &= 0 \end{aligned} \tag{7.12}$$

we now take A and B to be $[.99\,,1.01]$. Each model only has the authority to update its local variables, and will be notified of parametric changes by other models as soon as new information is available (see figure 7.22 for a similar example). Since only interval values are passed from one model to the other, semantic consistency is verified.

The concurrency graph is shown in figure 7.20. Note that while the hierarchical organization creates additional communication cycles, no additional mechanisms are needed to control the explosion of :change messages. The concurrent hierarchical solution of equa-

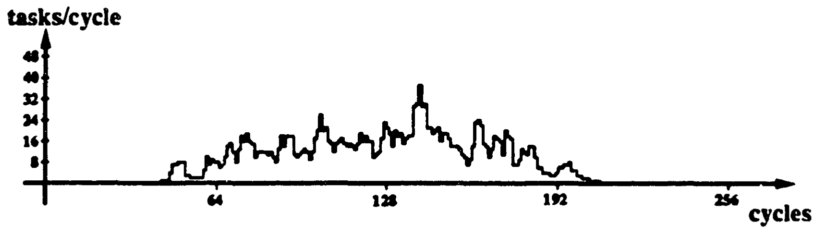


Figure 7.20: Parallelism Profile of Hierarchical Computations (Example 7.10)

tions (7.11) and (7.12) produces the following bounds:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} [0.420\,,0.580] \\ [0.824\,,0.900] \end{pmatrix}$$

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} [0.994\,,1.006] \\ [0.994\,,1.006] \end{pmatrix}$$

Observe that different strategies can be used for coordinating hierarchical computations. When lower levels are forced to iterate more then higher levels, the computations have a strict hierarchical flavor, whereas they are much more democratic if all information is communicated as soon as it is available.

Example 7.11

When external parameter changes occur, they now will trigger iterations and changes throughout the hierarchy. Let us as before first change r_f to $[1]$. Figure 7.21 illustrates how the equilibrium is disrupted and the computation resumes. At the new equilibrium we have:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} [0.470\,,0.530] \\ [0.848\,,0.884] \end{pmatrix}$$

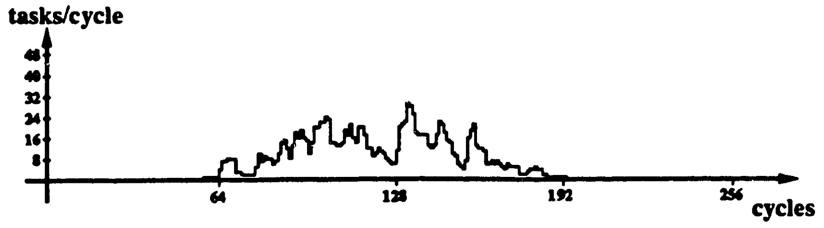


Figure 7.21: Parallelism Profile of a Parametric Changes in Example 7.11

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} [0.9977, 1.0023] \\ [0.9978, 1.0022] \end{pmatrix}$$

If we then change r_g, we find:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} [0.4999999992, 0.5000000008] \\ [0.8660254033, 0.8660254042] \end{pmatrix}$$

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} [0.9999999999, 1.0000000001] \\ [0.9999999999, 1.0000000001] \end{pmatrix}$$

It should be noted that although for these nonlinear equations the convergence at each individual level is quadratic, the hierarchical approach in effect decouples the equations by abstracting parameters by intervals. This implies that the global convergence typically will only be linear.

7.5 The Design Society

We can already put forward a number of practical implementation aspects for the design society. In particular, models can be represented by actor agencies [179] (see for example figure 7.22), which control only the free variables of that model. Parametric changes are communicated to dedicated actors which transmit the change locally. As soon as the values of local variables are updated, this information is sent out to the appropriate dependents. The corresponding refinement processes allow the agencies to provide increasingly better solution bounds in a concurrent fashion.

Intra-agency communication is allowed to be asynchronous and local dispatching and message merging can occur. Inter-agency communication however, usually requires some synchronization to ensure local consistency. Note that models, including their local iteration processes, can be nested. A simple example of this is a model representing a function inverse.

Agency boundaries can be used by the compiler for allocation on physical machines. As intra-agency communication is slower then inter-agency communication, it is desirable to locate the actors of an agency on the same processor. This information is indeed crucial and not always available in traditional parallel programming languages.

Finally, information concerning the design models can be obtained by sending queries to the corresponding actors. The design society therefore is a concurrent distributed representation of the design process.

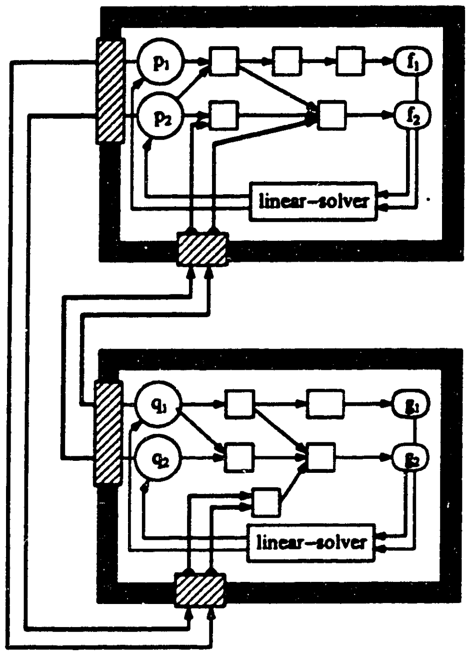


Figure 7.22: Modeling Agencies

Chapter 8

Conclusion

8.1 Summary

Based on a careful study of the existing state of the art in design automation, we have come to the conclusion that further progress in this area required us to rethink some of the basic premises. We therefore have identified the basic issues that need to be addressed and have integrated them in a framework for design.

First and foremost we have decided to use conservatively bounded models to overcome some of the problems associated with traditional approximate models. Indeed, while approximate models provide very little information to verify their accuracy, bounded models are guaranteed to conservatively enclose all solutions. Second, we propose to use a unified representation for design models and computations based on design graphs. Third, concurrency is inherent in design and should, in our opinion, explicitly be taken into account. Bounded models are shown to allow for concurrency in a natural way, since all parts of the computation have conservative bounds on partial results computed elsewhere. Fourth, we believe that progress in design automation necessitates a language for design that concisely captures the information provided by the designer.

We then have addressed a number of these issues. In particular, we have made a number of significant contributions to existing interval arithmetic enclosure techniques. We not only have improved the quality of the enclosures by improving linear interval solvers, but also the efficiency of the computations by presenting fast evaluation methods for partial derivatives and providing for variable accuracy computations. In addition, we have used denotational semantics to unify a number of enclosure languages.

Finally, we have applied interval arithmetic methods for performing design computations. In particular, we have used these techniques to address the consistency problem associated with the incremental nature of design and have demonstrated the advantage of interval computations in concurrent distributed systems.

8.2 Contribution

The contribution of this thesis is twofold, on the one hand our framework provides a promising new approach to some of the existing problems in the area of design automation. On the other hand, the algorithms contained in this thesis are a significant improvement to the state of the art for the solution of nonlinear algebraic equations.

It is our opinion that numerical set computations fill an existing void between traditional numerical analysis and discrete mathematics and promises to be an exiting new area of research in engineering design.

8.3 Future Research

As we have pointed out, a number of elements have to be integrated before substantial advances can be made towards the automation of design. Achieving this goal therefore requires a concerted research effort. The following issues are subject to further research.

Interval arithmetic techniques can be used for enclosure of discrete enumerations as well. The general problem of computing of bounds on discrete point sets however, is a hard one and more research is needed in this area. Good enclosures would allow us to include this type of variables in our framework for design hence avoiding the enumeration of all possibilities in a combinatorial fashion.

We believe that better techniques should be devised to handle singular interval solutions and matrices. This can be done not only by improving existing bisection techniques, but also by incrementally increasing the complexity of the bounds and the order of the nonlinear solver. Inclusion of additional intermediate variables in the computation of nonlinear solutions for example, can help prevent singular interval matrices. Tighter enclosures can also be obtained by recognizing parametric dependencies in interval coefficients and right hand sides [112]. Finally, scaled maximum norm methods [171] can be used in conjunction with line search methods to take advantage of local monotonicity properties.

Alternatively, one could proceed to use more sophisticated techniques then interval arithmetic to represent bounded models, hereby increasing the time complexity. Polytopes are a very interesting generalization of intervals by specifying bounds on vector components and therefore describe tighter enclosures [111]. As they can simply be represented by matrices, we believe that nonlinear solution operators still will have a polynomial time complexity. Unfortunately, the time complexity associated with more powerful representations very rapidly becomes exponential. It should however be emphasised that this can still be allowed if these techniques are used in conjunction with a hierarchical approach. In that case, the number of variables in each subproblem can remain relatively small.

Interval arithmetic methods can easily be extended to include optimization [194]. In addition, optimization specific techniques can be devised to take advantage of the particular structure of the associated nonlinear equations. Optimal computation of Hessians or second order slopes will, in our opinion, prove to be a particularly interesting problem.

While the techniques presented in this thesis focus on the computation of conservative bounds of the solution set, we have indicated in chapters 2 and 7 that conservative interior bounds have a number of important applications in design. Indeed, not only do they allow for design computations with operating ranges, but they can also be used to guarantee feasibility. Although a preliminary discussion of the computation of interior bounds can be found in [46], we believe that this problem deserves more attention.

Computation of enclosures for the solution of partial differential equations is one of the most important outstanding problems. While some solution techniques are already available for general one dimensional equations, i.e. ordinary differential equations. [65, 164, 216, 238, 139], as well as for a number of two dimensional differential equations [10, 213, 226, 38], very little work has been done on systems. Enclosure methods for the solution of systems of three dimensional elliptic partial differential equations would be a significant contribution.

We have included a design language as basic building block of our framework since is not trivial to conceive constructs that allow us to concisely capture the modeling information or functional requirements expressed by the designer. Furthermore, it is understood that the formulation of design as an optimization problem can be demanding. We are however convinced that this problem cannot be avoided by handling tradeoffs in an implicit fashion as is done by most other methods. Furthermore, by allowing for the incremental refinement of objective functions, our approach considerably simplifies this problem.

It is very important to provide this design language with proper semantic definitions in terms of design graphs and available services. While the framework and techniques proposed in this thesis provide a good indication of the specific aspects required for a practical implementation of design graphs, more work is required towards a unified graph representation for engineering models. This work will also include the development of fast activation algorithms, lazy evaluation techniques for design views and the development of standard protocols for information exchange.

The development of our framework will require further research on the automatic handling of inconsistencies. In our opinion, inconsistency handlers should indeed be incorporated both in the design language and the design graphs. This can include user specified strategies, such as backtrack mechanisms for example, that automatically recover the most specific consistent model.

The techniques presented in this thesis really need to be complemented with algorithms for controlling the cost of design computations, which can be decomposed into charges for processor time, communications and memory space. We believe that this can be done by abstracting over numerical solution operators [17] and conceiving appropriate selection methods based on available resources and requirements.

Finally, one should address the fundamental problem of reflexiveness. Indeed, while one can reason about a computation in the same computational system [149], this is always done by a level shift. A well-known problem solving system based on meta-level shifts is SOAR [143]. We believe that reflexive computation is impossible without level abstraction, for it would imply working within the model one is abstracting over. The reflexiveness problem has an important consequence for computations in design. It implies that it is impossible to minimize the required resources, such as cost, if the resources required by the design process are not neglectable w.r.t. to the overall process. By considering this problem at higher levels of abstraction one can indeed only approximately minimize the required resources as it can not include the cost of the optimizing computation itself. While better approximations can be obtained by including more levels of abstraction, the exact solution of this problem would lead to an infinite recursion. It should be noted that this issue is not only of academic but also of practical interest, as further technological developments tend to increase the cost associated with the design process w.r.t. to the manufacturing process.

Appendix A

Exact Solution of Midpoint Preconditioned Equations

A.1 Derivation of Exact Lower Bounds

We begin by deriving some useful identities. By definition we have:

$$\langle A\rangle_i v = B_{i_u} \tag{A.1}$$
$$\langle A\rangle_i e[i] = 1 \tag{A.2}$$

we can rewrite equation (A.1) as:

$$\sum_{j\neq i} |A|_{ij} v_j = \langle A\rangle_{ii} v_i - B_{i_u}$$

so that:

$$\langle A\rangle_i^* v = 2\langle A\rangle_{ii} v_i - B_{i_u} \tag{A.3}$$
$$|A|_i v = 2v_i - B_{i_u} \tag{A.4}$$

Similarly, equation (A.2) implies that:

$$\sum_{j\neq i} |A|_{ij} e[i]_j = \langle A\rangle_{ii} e[i]_i - 1$$

and therefore:

$$\langle A\rangle_i^* e[i] = 2\langle A\rangle_{ii} e[i]_i - 1 \tag{A.5}$$
$$|A|_i e[i] = 2e[i]_i - 1 \tag{A.6}$$

Note also that:

$$\sum_{j\neq i} |A|_{ij} e'[i]_j = -\langle A\rangle_{ii} e'[i]_i - 1 \tag{A.7}$$

leading to:

$$\langle A\rangle_i^* e'[i] = -1 \tag{A.8}$$

Recalling the definition of $m[i]$:

$$m[i] = v - e[i]\frac{v_i}{e[i]_i}$$

we can use equations (A.3) and (A.5) to simplify:

$$\begin{aligned}
\langle A\rangle_i^* m[i] &= 2\langle A\rangle_{ii} v_i - B_{iu} - \frac{v_i}{e[i]_i}(2\langle A\rangle_{ii} e[i]_i - 1) && \text{(A.9)}\\
&= -B_{iu} + \frac{v_i}{e[i]_i} && \text{(A.10)}
\end{aligned}$$

Similarly, we can use equations (A.4) and (A.6) in:

$$\begin{aligned}
|A|_i m[i] &= 2v_i - B_{iu} - \frac{v_i}{e[i]_i}(2e[i]_i - 1) && \text{(A.11)}\\
&= -B_{iu} + \frac{v_i}{e[i]_i} && \text{(A.12)}
\end{aligned}$$

We now can use these relations to solve for $c[i]$. Elimination of $c[i]$ in equations (4.13) results in:

$$\begin{aligned}
\langle A\rangle_i^*(m[i] + e'[i]s') - B_{il} &= 0\\
-B_{iu} + \frac{v_i}{e[i]_i} - s' - B_{il} &= 0
\end{aligned}$$

so that the value of the parameter s' is given by:

$$s' = -2B_{ic} + \frac{v_i}{e[i]_i}$$

Substituting this value in the equation of $c[i]$ and restricting our interest to the i-th component, we have:

$$\begin{aligned}
c[i]_i &= m[i]_i + e'[i]_i s'\\
&= 2e[i]_i B_{ic} - v_i
\end{aligned}$$

In a similar fashion we eliminate $c[i]$ in equations (4.20):

$$\begin{aligned}
|A|_i(m[i] + e[i]s) - B_{il} &= 0\\
-B_{iu} + \frac{v_i}{e[i]_i} + (2e[i]_i - 1)s - B_{il} &= 0
\end{aligned}$$

so that the corresponding value of s is given by:

$$s = \frac{2B_{ic} - \dfrac{v_i}{e[i]_i}}{2e[i]_i - 1}$$

which results in:

$$\begin{aligned} c[i]_i &= m[i]_i + e[i]_i s \\ &= \frac{2e[i]_i B_{ic} - v_i}{2e[i]_i - 1} \end{aligned}$$

Condition (4.19) can be simplified as follows:

$$\begin{aligned} \langle A\rangle_i^* m[i] - B_{il} &= \langle A\rangle_i^* v - \frac{v_i}{e[i]_i}\langle A\rangle_i^* e[i] - B_{il} \\ &= 2\langle A\rangle_{ii} v_i - B_{iu} - \frac{v_i}{e[i]_i}(2\langle A\rangle_{ii} e[i]_i - 1) - B_{il} \\ &= -2B_{ic} + \frac{v_i}{e[i]_i} \end{aligned}$$

so that:

$$\langle A\rangle_i^* m[i] - B_{il} < 0 \Rightarrow 2B_{ic} e[i]_i - v_i > 0$$

Appendix B

Variable Interval Precision

B.1 Constants for Variable Interval Precision

For $f(X) = g(X) \pm h(X)$ we have:

$$\begin{aligned} d(f(X), g^\circ(X) \pm h^\circ(X)) &\leq d(g(X), g^\circ(X)) + d(h(X), h^\circ(X)) \\ &\leq l_g w(g(X)) + l_h w(h(X)) \\ &\leq \max(l_g, l_h) w(f(X)) \end{aligned}$$

so that $l'_f = \max(l_g, l_h)$. We now compute l''_f:

$$\begin{aligned} d(g^\circ(X) \pm h^\circ(X), f^\circ(X)) &\leq \lambda_M w(g^\circ(X) \pm h^\circ(X)) \\ &\leq \lambda_M(1 + 2\lambda_M)(w(g(X)) + w(h(X))) \\ &\leq \lambda_M(1 + 2\lambda_M) w(f(X)) \end{aligned}$$

in other words $l''_f = \lambda_M(1 + 2\lambda_M)$. For the multiplication $f(X) = g(X)h(X)$:

$$\begin{aligned} d(f(X), g^\circ(X)h^\circ(X)) &\leq d(g^\circ(X)h(X), g^\circ(X)h^\circ(X)) + d(g^\circ(X)h^\circ(X), g(X)h^\circ(X)) \\ &\leq |g^\circ(X)| l_h w(h(X)) + |h^\circ(X)| l_g w(g(X)) \\ &\leq (1 + l_g) l_h |g(X)| w(h(X)) + (1 + l_h) l_g |h(X)| w(g(X)) \\ &\leq (l_g + 2 l_g l_h + l_h) w(f(X)) \end{aligned}$$

here $l'_f = l_g + 2l_g l_h + l_h$

$$\begin{aligned} d(g^\circ(X)h^\circ(X), f^\circ(X)) &\leq \lambda_M w(g^\circ(X)h^\circ(X)) \\ &\leq \lambda_M(|g^\circ(X)| w(h^\circ(X)) + |h^\circ(X)| w(g^\circ(X))) \\ &\leq \lambda_M(1 + 2\lambda_M)((1 + l_g)|g(X)| w(h(X)) + (1 + l_h)|h(X)| w(g(X))) \\ &\leq \lambda_M(1 + 2\lambda_M)(2 + l_g + l_h) w(f(X)) \end{aligned}$$

with $l''_f = \lambda_M(1 + 2\lambda_M)(2 + l_g + l_h)$. Finally, for $f(X) = \phi(g(X))$ with $\phi \in \Phi$, including the inverse, we have:

$$\begin{aligned} d(f(X), \phi(g^\circ(X))) &\leq l_g \phi'(\xi^\circ) w(g(X)) \\ &\leq l_g \frac{\phi'(\xi^\circ)}{\phi'(\xi)} w(\phi(g(X))) \end{aligned}$$

for some $\xi^\circ \in g^\circ(X)$ and $\xi \in g(X)$. Since for $\phi \in \Phi$, the ratio $\frac{\phi'(\xi^\circ)}{\phi'(\xi)}$ is bounded, $l'_f = l_g \frac{\phi'(\xi^\circ)}{\phi'(\xi)}$. Similarly,

$$\begin{aligned} d(\phi(g^\circ(X)), \phi^\circ(g^\circ(X))) &\leq \lambda_M w(\phi(g^\circ(X))) \\ &\leq \lambda_M \phi'(\xi^\circ) w(g(X)) \\ &\leq \lambda_M \frac{\phi'(\xi^\circ)}{\phi'(\xi)} w(\phi(g(X))) \end{aligned}$$

so that $l''_f = \lambda_M \frac{\phi'(\xi^\circ)}{\phi'(\xi)}$

Bibliography

[1] G.A. Agha. *Actors : A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Massachusetts, 1986.

[2] G. Alefeld. Das symmetrische Einselschrittverfahren bei linearen Gleichungen mit Intervallen als Koeffizienten. *Computing*, 18:329–340, 1977.

[3] G. Alefeld. Über die Durchführbarkeit des Gaußschen Algorithmus bei Gleichungen mit Intervallen als Koeffizienten. *Computing Suppl.*, 1:15–19, 1977.

[4] G. Alefeld. Intervallanalytische Methoden bei nichtlinearen Gleichungen. In S.D. Chatterji et al., editors, *Jahrbuch Überblicke Mathematik 1979*, pages 63–78, Mannheim, 1979. Bibl. Inst.

[5] G. Alefeld. On the convergence of some interval-arithmetic modifications of Newton's method. *SIAM Journal of Numerical Analysis*, 21(2):363–372, April 1984.

[6] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, 1983.

[7] G. Alefeld and L. Platzöder. A quadratically convergent Krawczyk-like algorithm. *SIAM Journal of Numerical Analysis*, 20(1):210–219, February 1983.

[8] C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, Cambridge, Massachusetts, 1966.

[9] E.L. Allgower, K. Böhmer, F.A. Potra, and W.C. Rheinboldt. A mesh-independence principle for operator equations and their discretizations. *SIAM Journal of Numerical Analysis*, 23(1):160–169, February 1986.

[10] W. Appelt. Fehlereinschließung für die Lösungen einer Klasse elliptischer Randwertaufgaben. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 54:T207–209, 1974.

[11] I.K. Argyros. A mesh-independence principle for nonlinear operator equations and their discretizations under mild differentiability conditions. *Computing*, 45:265–268, 1990.

[12] Arvind and D. E. Culler. Dataflow architectures. *Annual Review in Computer Science*, 1:225–253, 1986.

[13] Arvind, D. E. Culler, and Maa G. K. Assessing the benefits of fine-grained parallelism in dataflow programs. Memo 279, Computation Structures Group, MIT, June 1988.

[14] R.E. Barnhill, editor. *Chapter in Geometry Processing*, Philadelphia, PA, 1991. Society for Industrial and Applied Mathematics. To appear.

[15] R.G. Bartle. Newton's method in Banach spaces. *Proc. Amer. Math. Soc.*, 6:827–831, 1955.

[16] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.

[17] D.G. Bell, D.L. Taylor, and P.D. Hauck. Mathematical foundations of engineering design processes. In Stauffer [236], pages 181–189.

[18] A. Berman and R.J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, New York, 1979.

[19] D. P. Bertsekas and J. N. Tsitsiklis. A survey of some aspects of parallel and distributed iterative algorithms. Technical Report CICS-P-189, Center for Intelligent Control Systems, MIT, January 1990.

[20] Bojańczyk. Complexity of solving linear systems in different models of computation. *SIAM Journal of Numerical Analysis*, 21(3):591–603, June 1984.

[21] A.H. Bond and R.J. Ricci. Cooperation in aircraft design. In D. Sriram, R. Logcher, and S. Fukuda, editors, *Proceedings of the MIT-JSME Workshop on Cooperative Product Development*, Cambridge, Massachusetts, November 1989. track 6.

[22] S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley Publishing Company, 1977.

[23] R.P. Brent. The complexity of multiple-precision arithmetic. In *Proceedings of the Seminar on Complexity of Computational Problem Solving held at the Australian National University in December 1974*, pages 126–165, Brisbane, Australia, 1975. Queensland University Press.

[24] R.P. Brent. Fast multiple-precision evaluation of elementary functions. *Communications of the Association for Computing Machinery*, 23(2):242–251, April 1976.

[25] R.P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In J.F. Traub, editor, *Proceedings of the Symposium on Analytic Computational Complexity*, pages 151–176, New York, 1976. Academic Press.

[26] R.P. Brent. Algorithm 524: MP, a fortran multiple-precision arithmetic package [a1]. *ACM Transactions on Mathematical Software*, 4(1):71–81, March 1978.

[27] R.P. Brent. A fortran multiple-precision arithmetic package. *ACM Transactions on Mathematical Software*, 4(1):57–70, March 1978.

[28] E. Brisson. *Representation of d-Dimensional Geometric Objects.* PhD thesis, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, 1990.

[29] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[30] R. A. Brooks. Intelligence without representation. AI memo, Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts, 1988.

[31] D. C. Brown and R. Breau. Types of constraints in routine design problem-solving. In Sriram and Adey [233], pages 383–390.

[32] D. C. Brown and B. Chandrasekaran. *Design Problem Solving : Knowledge Structures and Control Strategies.* Pitman Publishing, London, U.K., 1989.

[33] L. L. Bucciarelli. An ethnographic perspective on engineering design. *Design Studies*, 9(3):159–168, July 1988.

[34] J.R. Bunch and D.J. Rose, editors. *Sparse Matrix Computations.* Academic Press, September 1976.

[35] S.A. Burns, T.M. Carr, and A. Locascio. Graphical representation of design process optimization processes. In NSF DMSC '90 [173], pages 115–122.

[36] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, Massachusetts, 1988.

[37] C. Chryssostomidis and N. M. Patrikalakis. Geometric modeling issues in computer aided design of marine structures. *Marine Technology Society Journal*, 22(2):15–33, 1988.

[38] L. Collatz. Guaranteed inclusions of solutions of some types of boundary value problems. In Ullrich [253], pages 189–198.

[39] D. D. Corkill and V. R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 748–756, Karlsruhe, West Germany, August 1983.

[40] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[41] G. Dahlquist and A. Bjorck. *Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1974.

[42] L. (Editor) Davis. *Genetic Algorithms and Simulated Annealing.* Pitman Publishing, London, U.K., 1987.

[43] R. Davis. Expert systems : Where are we? and where do we go from here? *The AI Magazine*, 3(2):3–22, Spring 1982.

[44] R. Davis. Expert systems : How far can they go? *The AI Magazine*, 10(1):61–67, Spring 1989.

[45] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[46] A. Deif. *Sensitivity Analysis in Linear Systems.* Springer-Verlag, 1986.

[47] J.W. Demmel and F. Krückeberg. An interval algorithm for solving systems of linear equations to prespecified accuracy. *Computing*, 34:117–129, 1985.

[48] Y.M. Dirickx and P.L. Jennergren. *Systems Analysis by Multilevel Methods, with Applications to Economics and Management.* John Wiley and Sons, 1979.

[49] J. R. Dixon, M. K. Simmons, and P. R. Cohen. An architecture for application of artificial intelligence to design. In *Proceedings of the ACM-IEEE Design Automation Conference*, pages 634–640, Albuquerque, New Mexico, June 1984.

[50] J. Doyle. A truth maintenance system. In B.L. Webber and N.J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 496–516, Los Altos, California, 1985. Morgan Kaufmann Publishers.

[51] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[52] E.H. Durfee. The distributed artificial intelligence melting pot. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1301–1306, November 1991.

[53] P.S. Dwyer. *Linear Computations.* John Wiley and Sons, New York, 1951.

[54] C.M. Eastman, A.H. Bond, and S.C. Chase. A formal approach for product model information. *Research in Engineering Design*, 2:65–80, 1991.

[55] S.C. Eisenstat, M.H. Schultz, and A.H. Sherman. Applications of an element model for Gaussian elimination. In Bunch and Rose [34], pages 85–96.

[56] S.D. Eppinger, D.E. Whitney, and D.A. Gebala. Organizing the tasks in complex design projects: Development of tools to represent design procedures. In NSF DMSC '92 [174], pages 301–309.

[57] L. D. Erman and V. R. Lesser. A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 483–490, Tbilisi, Georgia, USSR, September 1975.

[58] A. Esterline, M. Arnold, D.R. Riley, and A.G. Erdman. Representations for the knowledge revealed in conceptual mechanism design protocols. In NSF DMSC '90 [173], pages 123–135.

[59] D.W. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In R.J. Brachman and H.J. Levesque, editors, *Readings in Knowledge Representation*, pages 329–334, Los Altos, California, 1985. Morgan Kaufmann Publishers.

[60] S. Finger and J. R. Dixon. A review of research in mechanical engineering design. part I: Descriptive, prescriptive, and computer-based models of design processes. *Research in Engineering Design*, 1(1):51–67, 1989.

[61] S. Finger and J. R. Dixon. A review of research in mechanical engineering design. part II: Representations, analysis, and design for the life cycle. *Research in Engineering Design*, 1(1):121–137, 1989.

[62] K.D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[63] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, New Jersey, 1962.

[64] P. Freeman and A. Newell. A model for functional reasoning in design. In Walker [258], pages 621–640.

[65] I. Fried. *Numerical Solution of Differential Equations.* Academic Press, 1979.

[66] D. Friedman. Bringing mechanical design automation closer into alignment with electronic design automation. In *AUTOFACT '89: Conference Proceedings*, pages 16.1–13, Detroit, Michigan, October 1989.

[67] Y. Fujii, K. Ichida, and M. Ozasa. Maximization of multivariable functions using interval analysis. In *Interval Mathematics 1985: Proceedings of the International Symposium*, pages 17–26. Springer-Verlag, September 1985.

[68] J. Garloff and R. Krawczyk. Optimal inclusion of a solution set. *SIAM Journal of Numerical Analysis*, 23(1):217–226, February 1986.

[69] A. George and J.W.H. Liu. Algorithms for matrix partitioning and the numerical solution of finite-element systems. *SIAM Journal of Numerical Analysis*, 15:297–327, 1978.

[70] A. George and J.W.H. Liu. A quotient graph model for symmetric factorization. In I.S. Duff and G.W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 154–175, Philadelphia, PA, 1978. Society for Industrial and Applied Mathematics.

[71] A. George and J.W.H. Liu. A minimal storage implementation of the minimum degree algorithm. *SIAM Journal of Numerical Analysis*, 17:282–299, 1980.

[72] A. George and J.W.H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[73] A. Gerstenfeld, G. D. Gosling, and D. Touretzky. An expert system for managing cooperating expert systems. In Sriram and Adey [234], pages 219–244.

[74] P. E. Gill, W. Murray, and A. Wright. *Practical Optimization.* Academic Press, 1981.

[75] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[76] G. D. Gosling and A. M. Okseniuk. SLICE – a system for simulation through a set of cooperating expert systems. In Sriram and Adey [234], pages 1083–1093.

[77] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming*, pages 83–107. KTK Scientific Publishers, Tokyo, 1989.

[78] A. Griewank. *Automatic Evaluation of First and Higher-Derivative Vectors*, volume 97 of *International Series of Numerical Mathematics*, pages 135–148. Birkhäuser Verlag, 1991.

[79] M. D. Gross, S. M. Ervin, J. A. Anderson, and A. Fleisher. Designing with constraints. In Y. E. Kalay, editor, *Computability of Design*, pages 53–83. John Wiley and Sons, 1987.

[80] M. D. Gross, S. M. Ervin, J. A. Anderson, and A. Fleisher. Constraints : Knowledge representation in design. *Design Studies*, 9(3):133–143, July 1988.

[81] G. Gupta, editor. *Proceedings of the ASME Computers in Engineering Conference*, Chicago, 1986.

[82] H.N. Gursoy. *Shape Interrogation by Medial Axis Transform for Automated Analysis.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, November 1989.

[83] E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. In Gursoz [270], pages 87–106.

[84] W.W. Hager. Condition estimates. *SIAM Journal of Scientific and Statistical Computation*, 5(2):311–316, June 1984.

[85] E. Hansen. Interval arithmetic in matrix computations: Part 1. *SIAM Journal of Numerical Analysis*, 2:308–320, 1965.

[86] E. Hansen, editor. *Topics in Interval Analysis.* Oxford University Press, 1969.

[87] E. Hansen. Interval forms of Newton's method. *Computing*, 20:153–163, 1978.

[88] E. Hansen and R.I. Greenberg. An interval Newton method. *Applied Math. and Comp.*, 12:89–98, 1983.

[89] E. Hansen and S. Sengupta. Bounding solutions of systems of equations. *BIT*, 21:203–211, 1981.

[90] E. Hansen and R. Smith. Interval arithmetic in matrix computations: Part 2. *SIAM Journal of Numerical Analysis*, 4:1–9, 1967.

[91] P. Hansen, B. Jaumard, and S.H. Lu. A framework for algorithms in globally optimal design. *Journal of Mechanisms, Transmissions, and Automation in Design*, 111:353–360, September 1989.

[92] F. Harary. Sparse matrices and graph theory. In J.K. Reid, editor, *Large Sparse Sets of Linear Equations: Proceedings of the Oxford Conference of the Institute of Mathematics and Its Applications*, pages 139–150. Academic Press, 1971.

[93] E.B. Haugen. *Probabilistic Mechanical Design.* John Wiley and Sons, New York, 1980.

[94] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8:323–364, 1977.

[95] C. Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.

[96] C. Hewitt. Open systems architecture. Preprint, 1989.

[97] C. Hewitt and G.A. Agha, editors. *Concurrent Systems for Knowledge Processing*, Cambridge, Massachusetts, 1989. MIT Press.

[98] C. Hewitt, G.A. Agha, and J. Inman, editors. *Towards Open Information Systems Science.* MIT Press, Cambridge, Massachusetts, 1992. To Appear.

[99] N.J. Higham. A survey of condition number estimation for triangular matrices. *SIAM Review*, 29(4):575–596, December 1987.

[100] W. D. Hillis. Intelligence as an emergent behavior, or, the songs of Eden. *Daedalus*, 117(1):175–189, Winter 1988.

[101] W. Horwat. Concurrent smalltalk on the message-driven processor. Technical Report 1321, Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts, September 1991.

[102] A. Howe, P. Cohen, J. Dixon, and M. Simmons. Dominic : A domain-independent program for mechanical engineering design. In Sriram and Adey [233], pages 289–299.

[103] T.C. Hu and M.T. Shing. Some theorems about matrix multiplication. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, pages 28–35. IEEE Computer Society, 1980.

[104] B. A. Huberman, editor. *The Ecology of Computation*, The Netherlands, 1988. Elsevier Science Publishers.

[105] B. A. Huberman and T. Hogg. The behavior of computational ecologies. In Huberman [104], pages 77–115.

[106] T.E. Hull. The use of controlled precision. In J.K. Reid, editor, *Proceedings of the IFIP TC2 Working Conference on the Relationship between Numerical Computation and Programming Languages held in Boulder, Colorado in August 1981*, pages 71–84. North-Holland Publishing Company, 1982.

[107] T.E. Hull and A. Abrham. Variable precision exponential function. *ACM Transactions on Mathematical Software*, 12(2):79–91, June 1986.

[108] M. Iri. Simultaneous computation of functions, partial derivatives and estimates of rounding errors —complexity and practicality—. *Japan Journal of Applied Mathematics*, 1:223–252, 1984.

[109] P. Jain and A.M..Agogino. Global optimization using the multistart method. In Ravani [196], pages 39–44.

[110] P. Jansen and P. Weidner. High-accuracy arithmetic software – some tests of the acrith problem-solving routines. *ACM Transactions on Mathematical Software*, 12(1):62–70, March 1986.

[111] C. Jansson. A geometric approach for computing a posteriori error bounds for the solution of a linear system. *Computing*, 47:1–9, 1991.

[112] C. Jansson. Interval linear systems with symmetric matrices, skew-symmetric matrices and dependencies in the right hand side. *Computing*, 46:265–274, 1991.

[113] V.Y. Jin and A.H. Levis. Effects of organizational structure on performance: Experimental results. Technical Report LIDS-P-1978, Laboratory for Information and Decision Systems, MIT, May 1990.

[114] R. C. Johnson and R. C. Benson. A multistage decomposition strategy for design optimization. *Journal of Mechanisms, Transmissions, and Automation in Design*, 106:387–393, September 1984.

[115] C.B. Jones. A significance rule for multiple-precision arithmetic. *ACM Transactions on Mathematical Software*, 10(1):97–107, March 1984.

[116] W.M. Kahan. A more complete interval arithmetic. Lecture notes for an engineering summer course in numerical analysis, University of Michigan, 1968.

[117] K. M. Kahn and M. S. Miller. Language design and open systems. In Huberman [104], pages 291–313.

[118] R.B. Kearfott. Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, September 1987.

[119] R.B. Kearfott. Decomposition of arithmetic expressions to improve the behavior of interval iteration of nonlinear systems. *Computing*, 47:169–191, 1990.

[120] F. Kinoglu, D. Riley, and M. Donath. Knowledge-based system model of the design process. In Gupta [81], pages 181–191.

[121] S. Kirkpatrick, C. D. Gellat, and M. P. Jr. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[122] D.E. Knuth. *The Art of Computer Programming: Volume 2, Seminumerical Algorithms.* Addison-Wesley Publishing Company, 1969.

[123] W. A. Kornfeld. Ether - a parallel problem solving system. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 490–492, Tokyo, Japan, August 1979.

[124] E.V. Korngold, M.S. Shephard, R. Wentorf, and D.L. Spooner. Architecture of a design system for engineering idealizations. In B. Ravani, editor, *Advances in Design Automation*, volume 1, pages 259–265, Montreal, Canada, September 1989.

[125] A.S. Kott and J.H. May. Decomposition vs. transformation: Case studies of two models of the design process. In D.R. Riley and T.J. Cokonis, editors, *Proceedings of the ASME Computers in Engineering Conference*, pages 1–8, Anaheim, California, 1989.

[126] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1969.

[127] R. Krawczyk. Intervallsteigungen für rationale Funktionen and zugeordnete zentrische Formen. *Freiburger Intervall-Berichte*, February 1983.

[128] R. Krawczyk. A remark about the convergence of interval sequences. *Computing*, 31:255–259, 1983.

[129] R. Krawczyk. Interval iterations for including a set of solutions. *Computing*, 32:13–31, 1984.

[130] R. Krawczyk. A class of interval-Newton-operators. *Computing*, 37:179–183, 1986.

[131] R. Krawczyk. A Lipschitz operator for function strips. *Computing*, 36:169–174, 1986.

[132] R. Krawczyk. Properties of interval operators. *Computing*, 37:227–245, 1986.

[133] R. Krawczyk. Optimal enclosure of a generalized zero set of a function strip. *SIAM Journal of Numerical Analysis*, 24(5):1202–1211, October 1987.

[134] R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM Journal of Numerical Analysis*, 22:604–616, 1985.

[135] R. Krawczyk and A. Neumaier. An improved interval Newton operator. *Journal of Mathematical Analysis and Applications*, 118:194–207, 1986.

[136] R. Krawczyk and A. Neumaier. Interval Newton operators for function strips. *Journal of Mathematical Analysis and Applications*, 124:52–72, 1987.

[137] R. Krawczyk and K. Nickel. Die zentrische Form in der Intervallarithmetik, ihre quadratische Konvergenz und ihre Inklusionsisotonie. *Computing*, 28:117–132, 1982.

[138] G.A. Kriezis. *Algorithms for Rational Spline Surface Intersections*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, March 1990.

[139] F. Krückeberg and R. Leisen. Arbitrarily accurate boundaries for solutions of ODEs with initial values using variable precision arithmetic. In R. Vichenevetsky and J. Vignes, editors, *Numerical Mathematics and Applications*, pages 47–53. Elsevier Science Publishers, 1986.

[140] U.W. Kulisch. A new arithmetic for scientific computation. In Kulisch and Miranker [142], pages 1–26.

[141] U.W. Kulisch and W.L. Miranker. *Computer Arithmetic in Theory and Practice.* Academic Press, New York, 1981.

[142] U.W. Kulisch and W.L. Miranker, editors. *A New Approach to Scientific Computation.* Academic Press, 1983.

[143] J.E. Laird, A. Newell, and P.S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[144] L. Lamport and N. Lynch. Chapter on distributed computing. *Preprint*, February 1988.

[145] V. R. Lesser and D. D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):81–96, January 1981.

[146] V. R. Lesser and D. D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *The AI Magazine*, 4(3):15–33, Fall 1983.

[147] S. H. Lucas and S. J. Scotti. The sizing and optimization language, SOL: A computer language for design problems. Technical Memorandum 100565, NASA, April 1988.

[148] D. G. Luenberger. *Introduction to Dynamic Systems : Theory, Models and Applications.* John Wiley and Sons, 1979.

[149] P. Maes. Computational reflection. Technical Report 87-2, Artificial Intelligence Laboratory, VUB, Brussels, Belgium, 1987.

[150] P. Maes. How to do the right thing. *Connection Science*, 1(3), March 1990.

[151] P. Mandel and C. Chryssostomidis. A design methodology for ships and other complex systems. *Philosophical Transactions of the Royal Society A.*, 273:85–98, 1972.

[152] M.L. Manheim. *Highway Route Location as a Hierarchically Structured Sequential Decision Process : An Experiment in the use of Bayesian Decision Theory for Guiding an Engineering Process.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1964.

[153] C. R. Manning. Acore : The design of a core actor language and its compiler. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1987.

[154] C.R. Manning. Introduction to programming actors in acore. In Hewitt and Agha [97].

[155] C.R. Manning. Traveler: The actor observatory. In Hewitt and Agha [97].

[156] H.M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, 1957.

[157] O. Mayer. Über die Bestimmung von Einschließungsmengen für die Lösungen linearer Gleichungssysteme mit fehlerbehafteten Koeffizienten. *Angewante Informatik (Elektronische Datenverarbeitung)*, 12:164–167, 1970.

[158] M.D. Mesarovic, D. Macko, and Y. Takahara. *Theory of Hierarchical Multilevel Systems*. Academic Press, 1970.

[159] M. Minsky. *The Society of Mind*. Simon and Schuster, 1986.

[160] R.E. Moore. Automatic error analysis in digital computation. Technical Report LMSD-48421, Lockheed Missiles and Space Division, Sunnyvale, CA, 1959.

[161] R.E. Moore. *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Stanford University, Stanford, CA, 1962.

[162] R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[163] R.E. Moore. A test for existence of solutions to nonlinear systems. *SIAM Journal of Numerical Analysis*, 14(4):611–615, September 1977.

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

[165] R.E. Moore, editor. *Reliability in Computing: The Role of Interval Methods in Scientific Computing*. Academic Press, September 1988.

[166] R.E. Moore and S.T. Jones. Safe starting regions for iterative methods. *SIAM Journal of Numerical Analysis*, 14(6):1051–1065, December 1977.

[167] R.E. Moore and L. Qi. A successive interval test for nonlinear systems. *SIAM Journal of Numerical Analysis*, 19(4):845–850, August 1982.

[168] A. Neumaier. New techniques for the analysis of linear interval equations. *Linear Algebra Applications*, 58:273–325, 1984.

[169] A. Neumaier. Interval iteration for zeros of systems of equations. *BIT*, 25:256–273, 1985.

[170] A. Neumaier. Existence of solutions of piecewise differentiable systems of equations. *Arch. Math.*, 47:443–447, 1986.

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

[172] K. Nickel. On the Newton method in interval analysis. MRC Technical Summary Report 1136, University of Wisconsin, Madison, WI, 1971.

[173] *Proceedings of NSF Design and Manufacturing Systems Conference*, Tempe, Arizona, January 1990.

[174] *Proceedings of NSF Design and Manufacturing Systems Conference*, Atlanta, Georgia, January 1992.

[175] D.P. O'Leary. Estimating matrix condition numbers. *SIAM Journal of Scientific and Statistical Computation*, 1(2):205–209, June 1980.

[176] O. Østerby and Z. Zlatev. *Direct Methods for Sparse Matrices*, volume 157 of *Lecture Notes in Computer Science*. Springer-Verlag, 1983.

[177] A.M. Ostrowski. Über die Determinanten mit überwiegender Hauptdiagonale. *Comment. Math. Helv.*, 10:69–96, 1937.

[178] H. M. Painter. *Analysis and Design of Engineering Systems*. MIT Press, Cambridge, Massachusetts, 1960.

[179] J.J. Palmucci. ORGs: An architecture for constructing large scale actor systems. Bachelor's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1989.

[180] J.Y.C. Pan and J.M. Tenenbaum. An intelligent agent framework for enterprise integration. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1391–1408, November 1991.

[181] P.Y. Papalambros and D.J. Wilde. *Principles of Optimal Design : Modeling and Computation*. Cambridge University Press, 1987.

[182] J.D. Papastavrou and M. Athans. On optimal distributed decision architectures in a hypothesis testing environment. Technical Report LIDS-P-1928, Laboratory for Information and Decision Systems, MIT, January 1990.

[183] S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3(2):119–130, April 1961.

[184] N.M. Patrikalakis. Interrogation of surface intersections. In Barnhill [14]. To appear.

[185] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[186] M. Petkovic. *Iterative Methods for Simultaneous Inclusion of Polynomial Zeros*. Springer-Verlag, 1989.

[187] O. Pironneau. *Optimal Shape Design for Elliptic Systems*. Springer-Verlag, 1984.

[188] P. V. Prakash. *Computation of Surface-Surface Intersections for Geometric Modeling*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1988.

[189] L. Qi. A note on the Moore test for nonlinear systems. *SIAM Journal of Numerical Analysis*, 19(4):851–857, August 1982.

[190] L.B. Rall. *Automatic Differentiation: Techniques and Applications.* Number 120 in Springer Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1981.

[191] L.B. Rall. Differentiation and generation of Taylor coefficients in PASCAL-SC. In Kulisch and Miranker [142], pages 291–309.

[192] L.B. Rall. Mean value and Taylor forms in interval analysis. *SIAM Journal of Mathematical Analysis*, 14(2), March 1983.

[193] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Ellis Horwood Limited, Chichester, 1984.

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

[195] H. Ratschek and J. Rokne. The transistor modeling problem again. Unpublished Manuscript, April 1991.

[196] B. Ravani, editor. *Advances in Design Automation*, volume 2, Montreal, Canada, September 1989.

[197] J. Regh, A. Elfes, S. Talukdar, R. Woodbury, M. Eisenberger, and R. Edahl. Case : Computer-aided simultaneous engineering. Technical Report EDRC 05-22-88, Engineering Design Research Center, CMU, Pittsburg, Pennsylvania, 1988.

[198] G.V. Reklaitis, A. Ravindran, and K.M. Ragsdell. *Engineering Optimization : Methods and Applications.* John Wiley and Sons, 1983.

[199] J.E. Renaud and G.A. Gabriele. Using the generalized reduced gradient method to handle equality constraints directly in a multilevel optimization. In Ravani [196], pages 7–14.

[200] J.R. Rinderle. Implications of function-form-fabrication relations on design decomposition strategies. In Gupta [81], pages 193–198.

[201] F.N. Ris. *Interval analysis and applications to linear algebra.* PhD thesis, Oxford University, Oxford, U.K., 1972.

[202] J. Rohn. New condition numbers for matrices and linear systems. *Computing*, 41:167–169, 1988.

[203] J. Rohn. Solving systems of linear interval equations. In Moore [165], pages 171–182.

[204] J. Rohn. Some tests of generalized bisection. *Linear Algebra Applications*, 1990. To appear.

[205] D.J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[206] D.J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[207] D.J. Rose and R.E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal of Applied Mathematics*, 34(1):176–197, January 1978.

[208] J. R. Rossignac and M. A. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In Wozny et al. [270], pages 145–180.

[209] S. Rowley and C. Rockland. The design of simulation languages for systems with multiple modularities. Technical Report LIDS-P-1961, Laboratory for Information and Decision Systems, MIT, April 1990.

[210] D. E. Rumelhart, J. L. McClelland, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, Massachusetts, 1986.

[211] T.L. Saaty. *The Analytic Hierarchy Process : Planning, Priority Setting, Resource Allocation.* McGraw-Hill Book Company, 1980.

[212] S.E. Sarma and J.R. Rinderle. Quiescence in interval propagation. In Stauffer [236], pages 257–263.

[213] G. Scheu. Schrankenkonstruktion für die Lösung der elliptischen Randwertaufgabe mit konstanten Koeffizienten. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 55:T221–223, 1975.

[214] D. A. Schmidt. *Denotational Semantics : A Methodology for Language Development.* Wm. C. Brown Publishers, Dubuque, Iowa, 1986.

[215] J.L. Schonfelder and J.T. Thomason. Arbitrary precision arithmetic in Algol 68. *Software – Practice and Experience*, 9:173–182, 1979.

[216] J. Schröder. Numerical algorithms for existence proofs and error estimates for two-point boundary value problems. In Ullrich [253], pages 247–268.

[217] H. Schwandt. *Schnelle fast global konvergente Verfahren für die fünf punkt Diskretisierung der Poisson Gleichung mit Dirichletschen Randbedingungen auf Rechteckgebieten.* PhD thesis, Technical University of Berlin, 1981.

[218] D. Serrano. *Constraint Management in Conceptual Design.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, October 1987.

[219] D. Serrano and D. Gossard. Constraint management in MCAE. In J.S. Gero, editor, *Artificial Intelligence in Engineering: Design*, pages 217–240. Elsevier Science Publishers, 1988.

[220] V. Shapiro and H. Voelcker. On the role of geometry in mechanical design (technical note). *Research in Engineering Design*, 1(1):69–73, 1989.

[221] J.M. Shearer and M.A. Wolfe. An improved form of the Krawczyk-Moore algorithm. *Applied Mathematics and Computation*, 17:229–239, 1985.

[222] J.M. Shearer and M.A. Wolfe. Some algorithms for the solution of a class of nonlinear algebraic equations. *Computing*, 35:63–72, 1985.

[223] J.M. Shearer and M.A. Wolfe. Some computable existence, uniqueness, and convergence tests for nonlinear systems. *SIAM Journal of Numerical Analysis*, 22(6):1200–1207, December 1985.

[224] J.M. Shearer and M.A. Wolfe. A note on the algorithm of Alefeld and Platzöder for systems of nonlinear equations. *SIAM Journal of Scientific and Statistical Computation*, 7(2):362–369, April 1986.

[225] J.N. Siddall. *Probabilistic Engineering Design.* Marcel Dekker, New York, 1983.

[226] V.G. Sigillito. *Explicit a priori inequalities with applications to boundary value problems.* Pitman Publishing, 1977.

[227] H.A. Simon. *Administrative Behavior : A Study of Decision-Making Processes in Administrative Organization.* The Free Press, New York, 1957.

[228] H.A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, volume 106, pages 467–482, December 1962.

[229] H.A. Simon. *The Shape of Automation.* Harper and Row, Publishers, 1965.

[230] H.A. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, Massachusetts, 1969.

[231] R. D. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):61–70, January 1981.

[232] B. Speelpenning. *Compiling fast Partial Derivatives of Functions given by Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1980.

[233] D. Sriram and R. Adey, editors. *Applications of Artificial Intelligence in Engineering Problems: 1st International Conference*, volume 1, Southampton, U.K., April 1986.

[234] D. Sriram and R. Adey, editors. *Applications of Artificial Intelligence in Engineering Problems: 1st International Conference*, volume 2, Southampton, U.K., April 1986.

[235] D. Sriram, R. D. Logcher, N. Groleau, and J. Cherneff. Dice : An object oriented programming environment for cooperative engineering design. Technical Report IESL-89-03, Intelligent Engineering System Laboratory, MIT, Cambridge, Massachusetts, March 1989.

[236] L.A. Stauffer, editor. *Design Theory and Methodology - DTM '91*, Miami, Florida, September 1991.

[237] V. L. Stefanuk. Collective behavior of automata and the problems of stable local control of a large-scale system. In Walker [258], pages 51–58.

[238] H.J. Stetter. Validated solution of initial problems for ODE. In Ullrich [253], pages 171–187.

[239] J.E. Stoy. *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, Massachusetts, 1977.

[240] N. P. Suh. *The Principles of Design*. Oxford University Press, 1990.

[241] R. Suri and M. Shimizu. Achieving competitiveness in engineering design through 'design for analysis'. In NSF DMSC '90 [173], pages 47–50.

[242] G. J. Sussman and G. L. Jr. Steele. Constraints : A language for expressing almost-hierarchical descriptions. *Artificial Intelligence*, 14:1–39, 1980.

[243] B.A. Szabo. Control of errors of idealization in design computations. In NSF DMSC '90 [173], pages 105–107.

[244] G. Taguchi and M. S. Phadke. Quality engineering through design optimization. In *Proceedings of IEEE Globecom Conference*, pages 1106–1113, Atlanta, Georgia, November 1984.

[245] K. S. Tam and E. D. Bloodworth. Formation and exploration of feasible design space. In NSF DMSC '90 [173], pages 1–7.

[246] R.E. Tarjan. Graph theory and Gaussian elimination. In Bunch and Rose [34], pages 3–22.

[247] R.H. Thomason. Fahlman's NETL and inheritance theory. Preprint, 1990.

[248] W.F. Tinney and J.W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of IEEE*, 55:1801–1809, 1967.

[249] T. Tomiyama, T. Kiriyama, H. Takeda, D. Xue, and H. Yoshikawa. Metamodel : A key to intelligent CAD systems. *Research in Engineering Design*, 1(1):19–34, 1989.

[250] T. Tomiyama and H. Yoshikawa. Extended general design theory. In T. Sata and E. Warman, editors, *Proceedings of IFIP WG 5.2 Working Conference, Design Theory for CAD*, pages 95–130, Tokyo, Japan, October 1985.

[251] S.S. Tong. Coupling symbolic manipulation and numerical simulation for complex engineering designs. In *Conference on Expert Systems for Numerical Computing*, Purdue University, December 1988. International Association of Mathematics and Computers in Simulation.

[252] J.F. Traub. *Iterative Methods for the Solution of Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1964.

[253] C. Ullrich, editor. *Computer Arithmetic and Self-Validating Numerical Methods*. Academic Press, 1989.

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

[255] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Massachusetts, 1989.

[256] G.N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design.* McGraw-Hill Book Company, 1984.

[257] R.J. Verrilli, K.L. Meunier, J.R. Dixon, and M. K. Simmons. Iterative respecification management: A model for problem-solving networks in mechanical design. In *Proceedings of the ASME Computers in Engineering Conference*, pages 103–112, New York, 1987.

[258] D.E. Walker, editor. *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, London, U.K., September 1971.

[259] A. C. Ward. *A Theory of Quantitative Inference for Artifact Sets Applied to a Mechanical Design Compiler.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, November 1988.

[260] A.C. Ward. Foundations of quantitative inference about sets of design possibities: 1991 progress report. In NSF DMSC '92 [174], pages 441–447.

[261] K. J. Weiler. *Topological Structures for Geometric Modeling.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, August 1986.

[262] A. W. Westerberg, P. C. Piela, E. Subrahmanian, G. W. Podnar, and W. Elm. A future computer environment for preliminary design. Technical Report EDRC 05-42-89, Engineering Design Research Center, CMU, Pittsburg, Pennsylvania, 1989.

[263] J.H. Wilkinson. *Rounding Errors in Algebraic Processes.* Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

[264] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.

[265] M.A. Wolfe. A modification of Krawczyk's algorithm. *SIAM Journal of Numerical Analysis*, 17:376–379, 1980.

[266] M.A. Wolfe. Interval methods for algebraic equations. In Moore [165], pages 229–248.

[267] K.L. Wood, E.K. Antonsson, and J.L. Beck. Comparing fuzzy and probability calculus for representing imprecision in preliminary engineering design. In W.H. Elmaraghy, W.P. Seering, and D.G. Ullman, editors, *Design Theory and Methodology – DTM '89*, pages 99–105, Montreal, Canada, September 1989.

[268] R. Woodbury and I. Oppenheim. An approach to geometric reasoning. Technical Report EDRC-48-06-87, Engineering Design Research Center, CMU, Pittsburg, Pennsylvania, 1987.

[269] J. Woodwark, editor. *Geometric Reasoning.* Oxford University Press, 1989.

[270] M. J. Wozny, J. U. Turner, and K. Preiss, editors. *Geometric Modelling for Product Engineering*, The Netherlands, 1990. Elsevier Science Publishers.

[271] A. Yonezawa. *ABCL - An Object-Oriented Concurrent System.* MIT Press, Cambridge, Massachusetts, 1990.

[272] H. Yoshikawa. General design theory and a CAD system. In T. Sata and E. Warman, editors, *Proceedings of IFIP WG 5.2-5.3 Working Conference, Man-Machine Communication in CAD/CAM*, pages 35–58, Tokyo, Japan, October 1980.

[273] R.C. Young. The algebra of many-valued quantities. *Mathematical Annals*, 104:260–290, 1931.