Refining Imprecise Models and Their Behaviors

by

Herbert Kay, B.S., M.A.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 1996

To Smilin' Jack

Acknowledgments

I would like to thank my advisor, Ben Kuipers, for his support throughout my graduate student career here at UT. He helped me keep my standards high and provided me with the opportunity to explore lots of interesting paths. I would also like to thank My Lovely Wife, Meg, for putting up with the time I spent exploring all of those interesting paths, and my Mother for reminding me that one shouldn't forget to get on with one's life just because there are so many interesting paths to explore.

I would also like to acknowledge the support provided to me and the rest of the Qualitative Reasoning Group by the NSF, NASA, and the Texas Advanced Research Program.

Finally, I'd like to thank the members of the Qualitative Reasoning Group, both past and present, for their help and advice.

After all is said and done, I had a blast doing this research.

Herbert Kay

The University of Texas at Austin
December 1996

Refining Imprecise Models and Their Behaviors

Publication No. ______________________

Herbert Kay, Ph.D.
The University of Texas at Austin, 1996

Supervisor: Benjamin J. Kuipers

This dissertation describes methods for simulating and refining imprecisely-defined Ordinary Differential Equation (ODE) systems. When constructing a model of a physical process, a modeler must cope with uncertainty due to incomplete knowledge of the process. For tasks such as design and diagnosis, the effects of this uncertainty must be considered. However, predicting the behavior of an imprecisely-defined model is not easy since the model covers a space of many precise instances, each of which behaves differently.

While model uncertainty cannot be completely eliminated, it is possible to reduce it. Model refinement uses observations of a physical process to rule out portions of the model space that could not have produced the observations. As more experience with the physical process is gained, the imprecision in the model is further reduced.

This dissertation describes three methods for reasoning with imprecise ODE models. SQsim is a simulator that produces a guaranteed bound on the behavior of an imprecise ODE model. By using a multiple-level representation and inference methods that span the qualitative-to-quantitative spectrum, SQsim produces predictions whose uncertainty is consistent with model imprecision. We demonstrate SQsim on a complex, nonlinear chemical process and compare it to other methods for simulating imprecise ODE models.

MSQUID is a function estimator for fitting (and bounding) noisy data that is known to be monotonic. It uses a neural-network inspired model and nonlinear constrained optimization to search a space of monotonic functions. We prove that MSQUID can estimate any monotonic function and show that it produces better estimates than does unconstrained optimization.

SQUID, which uses SQsim and MSQUID as components, is a system identification method that refines an imprecise model using a stream of observations from a physical process. SQUID uses refutation to rule out portions of the model space that are inconsistent

with the observations. We show that this approach to refinement is significantly more efficient than parameter estimation for models with functional uncertainty and that it provides greater robustness in the face of uninformative observations.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation is concerned with representing and reasoning with imprecise models of dynamic systems. In this chapter, we motivate the need for reasoning about imprecise systems for both prediction and model refinement tasks.

1.1 The Need for Reasoning with Imprecision

Much work in engineering and science involves the construction of models. For continuous physical systems, Ordinary Differential Equations (ODE) models are a common representation. Such models consist of a system of differential equations that describe the trajectory of state variables over time. Making predictions from such models is quite straightforward and efficient given the availability of numerical ODE solvers such as LSODE and Runge-Kutta. However, this approach to modeling and simulation requires that the modeler determine the precise ODE for the physical system of interest. For many problems, a precise ODE may be difficult to find.

Consider the task of designing a physical device (e.g., a chemical plant or electronic circuit). The designer will typically construct a model of the device with precise values for its components (determined by detailed analysis), but real components have manufacturing tolerances, so it is unlikely that the real device will behave exactly the same as the model.

Consider the task of modeling an existing physical device, perhaps for the purpose of monitoring or diagnosis. While the modeler may have a good understanding of the physical principles involved in the device, he will not have precise knowledge of the values of every parameter and function that describes the system. If he creates a precise model, its predictions may not agree with the behavior of the real system.

In each of these cases, there is an inherent imprecision in the modeling task. This imprecision defines a *model space* that covers many different precise models. Models are typically used to make predictions about the behavior of the modeled system. If we view a model space $\mathcal{M}$ as a set of ODE models, we can define a *trajectory space* $\mathcal{T}(\mathcal{M})(t)$ as the set of trajectories starting from some initial state. We can then describe two different tasks associated with imprecise models:

- The *prediction task* determines $\mathcal{T}(\mathcal{M})(t)$ from $\mathcal{M}$. Ideally, we would like to use models that explicitly represent model imprecision and can produce a *useful* behavioral prediction – i.e., a prediction that is precise enough to distinguish desired from undesired behavior.

- The *refinement task* determines $\mathcal{M}_R \subseteq \mathcal{M}$ from $\mathcal{T}(\mathcal{M})(t)$ and a stream of (possibly noisy) measurements. Ideally, we would like $\mathcal{M}_R$ to be as small as possible for a given set of measurements.

1.2 Existing Methods for Dealing With Imprecision and Why They are Insufficient

1.2.1 Prediction

Considering the importance of reasoning about imprecise models, it is not surprising that there has been much work in this area. Some approaches to imprecise modeling are:

- Simply ignore the problem. For some classes of systems (for example, systems with tight feedback), small amounts of parametric uncertainty do not affect the results. For systems with larger amounts of uncertainty or nonlinearities, however, the model–device mismatch may be large.

- Use Monte-Carlo analysis [Kahaner *et al.*, 1989]. By running repeated simulations of the ODE system using different combinations of parameter values, it is possible to get some idea of the behavior of the models in the model space. Unfortunately, in addition to being slow, this approach may miss certain key combinations of parameters that produce a behavior not captured by the simulations that were run. Furthermore, it is difficult to represent functional uncertainty with this method.

- Represent uncertainty using probability distributions for imprecisely known parameters and use variance propagation methods to make predictions [Gelb, 1974, Reckhow, 1987]. This approach fails when the uncertainty cannot be modeled as probabilities (for example, if we have functional uncertainty). Furthermore, linearization is required to solve the variance equations which leads to a mismatch between the behavior of the true model and its linearization.

- Use a specialized model structure that separates the uncertainty from the ODE system [Lunze, 1989]. The ODE model and the uncertainty model can be solved separately and then combined to bound the model space. This method yields very precise predictions, but requires very specific model structures and is thus not generally applicable.

- Use intervals to bound parameters and use an interval ODE solver to compute predictions [Corliss, 1995a]. This method is fast and general, but produces very weak bounds.

In this dissertation, we present a *general* method that can represent and simulate model spaces of ODE systems such that the predictions maintain precision consistent with the size of the model space.

1.2.2 Refinement

In contrast to prediction, refinement is a learning task given prior information in the form of a model space $\mathcal{M}$. Traditional system identification methods define $\mathcal{M}$ using a parametric model and then search the parameter space for a point that minimizes some criterion (e.g., sum of squared error between model and measurements). Traditional system identification methods make the following problem assumptions:

- There exists a precise parameterization of the model space. If the desired model space cannot be spanned by a single parameterization then multiple models must be considered.
- The model space is easily searchable. This may not be the case if $\mathcal{M}$ has high dimensionality.
- The data is informative. Because search methods must converge to a precise point in the parameter space, measurements which do not exhibit the dynamic behavior to be modeled can cause the identifier to converge on the wrong model.

These assumptions require that adequate information about the model space and control over the measurement set is possible. For some cases, such as monitoring an uncertain system, such detailed prior knowledge and an informative measurement stream may not be available. In this dissertation, we present a model refinement method that is robust in the face of uncertain model structure and uninformative data.

1.3 Goals of this Research

This dissertation investigates the use of so-called *semiquantitative methods*, i.e., the combination of qualitative and quantitative representations and inference methods, for simulation and data analysis. The driving viewpoint behind this work is that the tasks of building and using models most naturally draws on information across a spectrum of imprecision from the purely qualitative to the purely quantitative. One goal of this research is to show that this viewpoint can be successfully used to better analyze engineered systems. Another goal is to extend existing semiquantitative methods so that they are more useful and to assess the limitations of such techniques. This research has been exploratory in nature and the actual methods developed in the course of investigation are *not* meant to be efficient or complete solutions. Rather, they were designed to illustrate fruitful directions for future research.

1.4 Summary of Key Contributions of this Research

This dissertation presents four new methods for predicting and refining imprecisely-defined ODE systems:

1.4.1 NSIM – A Numerical Simulator for Imprecise Models

NSIM (for Numerical Simulator using Interval Methods) is an extension to the QSIM modeling and simulation system that generates a numerical bounding envelope for each variable in a QSIM model [Kuipers, 1994] augmented with partial quantitative information. NSIM's envelopes are guaranteed to bound all trajectories in $\mathcal{T}(\mathcal{M})(t)$.

1.4.2 SQSIM – A Semiquantitative Simulator for Imprecise Models

SQSIM (for SemiQuantitative SIMulator) is a functional superset of the QSIM modeling and simulation system that utilizes QSIM, Q2 [Kuipers and Berleant, 1988], and NSIM to produce better predictions of semiquantitative models by ensuring that predictions from each inference method are consistent with each other. SQSIM produces predictions that are guaranteed to cover all possible behaviors of the ODE systems spanned by the model space.

1.4.3 MSQUID – A Monotonic Function Estimator with Confidence Bands

MSQUID (for Monotonic SQUID) is a non-parametric function estimator that fits a monotonic function to noisy data using a neural network-inspired estimation function. In contrast to most non-parametric estimators, MSQUID can also compute a confidence band on the resulting fit.

1.4.4 SQUID – A System Identification Method Using Semiquantitative Models

SQUID (for SemiQUantitative IDentifier) is an identification method that uses semiquantitative inference (rather than search) to identify a model space $\mathcal{M}_R \subseteq \mathcal{M}$ given a set of noisy measurements. Because it does not directly search the model space, it can be faster and more robust in the face of model uncertainty and uninformative data than can search-based methods.

1.5 Guide to this Dissertation

This dissertation has 4 chapters. Chapters 2, 3, and 4 are self-contained and describe methods, examples, and related work for their respective topics. Chapter 2 presents the representation and simulation of imprecisely-defined models – in particular, the NSIM and SQSIM systems. Chapter 3 discusses the refinement of memoryless functions in the spirit of regression and describes the MSQUID system. Chapter 4 discusses refinement of imprecisely-defined ODE systems using SQUID. Chapter 5 discusses future directions for this research as a whole.

Chapter 2

Representing and Simulating Imprecise Models

Making predictions from imprecise models requires the selection of a model representation as well as a simulation method tailored to that representation. In this chapter, we describe the SQDE (for SemiQuantitative Differential Equation) representation for imprecise models and show that it is expressive enough to capture a wide range of engineering uncertainty. Next, we describe the SQsim (for SemiQuantitative SIMulator) simulator which produces *semiquantitative behaviors* from an SQDE. Finally, we discuss related work.

2.1 Representing Imprecise Models Using SQDEs

2.1.1 Requirements

Our goal is to represent a space of of ODE models. To be useful, the representation must have the following properties:

- It must span a space of ODE models.
  We want to capture a space of ODE models in a manner that is precise enough to represent a set of models with a minimum of unwanted ones.

- It must tractably represent the model space.
  The representation must be tractable both in the size of the description and in the degree to which it can be simulated. For example, one possible representation of an imprecise ODE model is as a set of ODEs. This is impractical since it requires the enumeration of an infinite set and an infinite amount of work to simulate each ODE.

- It must captures common forms of engineering uncertainty.
  We define two classes of model uncertainty:

- Parametric imprecision models the uncertainty in the value of a scalar model parameter. This is the most common type of uncertainty and can represent the tolerance on a component or a bound on a reaction rate.
- Functional imprecision models the uncertainty in the functional relationship between two or more variables. This type of uncertainty can be used to represent a family of functions which may not have a common parametric form yet share some qualitative property (such as monotonicity). Functional imprecision is a very useful feature for a model representation since it reduces the number of models needed to reason about a particular process, however, because it is often difficult to reason with, it is often ignored in practice.

2.1.2 The SQDE Representation

When simulating an uncertain system, it is helpful to separate the information that *is* precisely known from that which *is not*, so that unambiguous inferences can be made from the precise information. Often, there will be precise information about the structural and qualitative properties of the model, while the numerical information will be imprecise. We make use of this distinction by using a multi-level representation based on the QSIM [Kuipers, 1986] representation for ODE systems. The levels are:

- The structural level (SDE).
  At this level, we describe the form of the ODE system in terms of the state variables and the constraints that link them. Constraints are described as arithmetic operators and functional relationships. The structural level provides the backbone of the modeling process. We call this representation the SDE (for Structural Differential Equation).

- The qualitative level (QDE).
  The qualitative level adds information about the domains of each model variable as well as the nature of the functional constraints. Each model variable is partitioned into a set of "landmark" values which represent important (to the modeler) values in the domain of the variable. For example, if A models the amount of fluid in a tank, it may be modeled by the *quantity space* $(0, FULL)$ which states that empty $(A = 0)$ and full $(A = FULL)$ are two interesting values in the space of the variable. The landmarks in a quantity space define an ordinal relationship on the values in the domain of the variable. Furthermore, sets of corresponding landmarks restrict the possible effects of constraints in the model. We call this representation the QDE (for Qualitative Differential Equation).

  The QDE also records information about the shape of functional forms in the model. Functional shapes can be defined as being monotonic over a region, S-shaped, Ushaped, etc. As we shall see, by making this information explicit, simulation methods that work at the qualitative level can infer facts about the behavior of a model that are unavailable using purely numerical methods.

- The quantitative level (SQDE).
  At this level, we record the uncertainty in the model. We represent parametric imprecision with numerical intervals that bound the uncertainty. We represent functional imprecision by defining *static envelopes* within which the functional constraint must lie. This type of imprecise knowledge has several advantages:

  - It is a form of knowledge readily available in engineering domains.
    Often, parametric and functional tolerances are described using intervals and bounding curves in the engineering literature.
  - There is an existing mathematics for computing with intervals and static envelopes.
  - We can represent precise models by allowing all interval widths and static envelope widths to go to zero. Thus, the more we know about a model, the more precise we can make it.

  We call this representation the SQDE (for Semiquantitative Differential Equation) because it contains both the QDE and imprecise quantitative information.

Figure 2.1 shows the representation of a simple one-tank system with a constant inflow. At each level, we further restrict the model space so that it eventually contains only one ODE. By using simulation techniques targeted for particular levels, we can thus utilize this information in a variety of ways.

2.2 Simulating SQDEs Using SQsim

By using a multi-level model, the modeler can represent both precise qualitative knowledge explicitly at the QDE level and imprecise numerical information at the SQDE level. We can then use inference methods on the QDE that are independent of the imprecise quantitative information. For example, if we are describing a single tank system with a model that states that outflow is a monotonic function of amount then regardless of the precise functional relationship, we can absolutely conclude that an increase in amount will produce an increase in outflow.

As we shall see, inference at the qualitative level can improve the quality of predictions at the numerical level and vice versa. Our goal is thus to use a set of inference techniques at various points along the qualitative-to-quantitative spectrum and combine

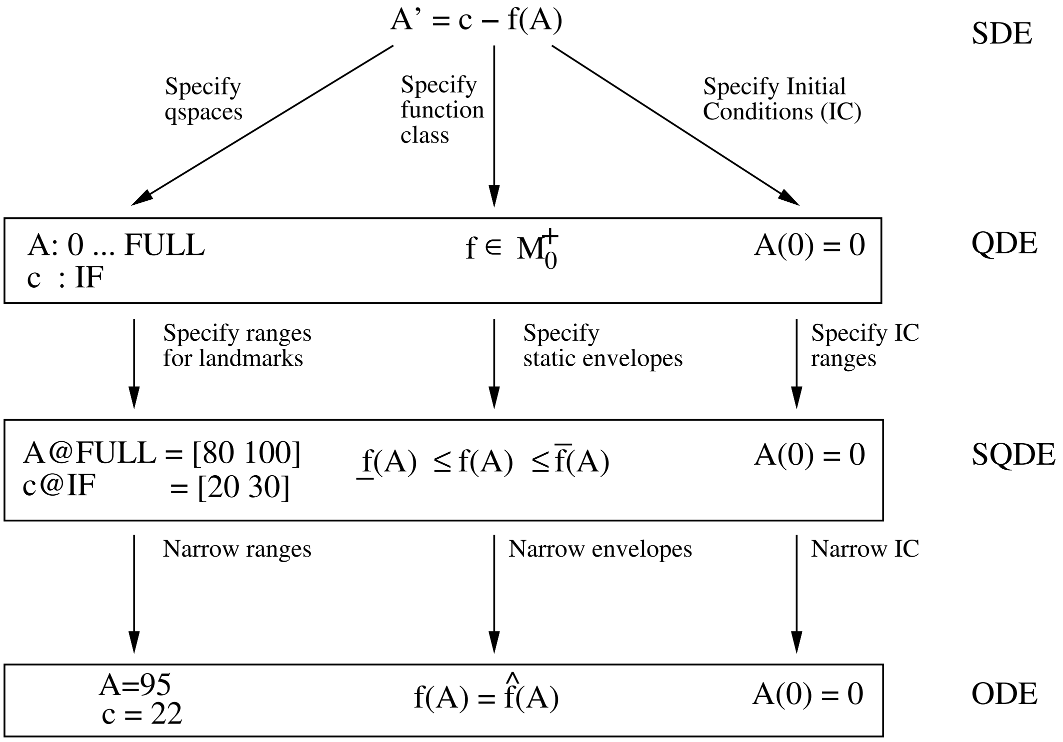


Figure 2.1: Multi-level model of an imprecisely-known system. At each level, the representation entails a space of ODEs. At the top is the purely structural SDE. The QDE adds qualitative information to the SDE, and the SQDE adds imprecise numerical information to the QDE. As we move downward we specify more information, thus reducing the size of the model space.

their results to yield tighter behavioral predictions. The next sections describe these methods.

2.2.1 The Semiquantitative Behavior Representation

In traditional numerical simulation, the representation of a behavior is simply a list of time and value pairs for each variable in the system. In interval methods, the representation is a triple of time, lower bound, upper bound. The semiquantitative behavior representation extends these descriptions by adding additional qualitative and semiquantitative information. This additional information provides a richer description of the trajectory family which can be exploited to better distinguish between models. The semiquantitative description consists of three components:

The Qualitative Trajectory

The qualitative trajectory describes the shape of the behavior in terms of regions of monotonic behavior with respect to time. It consists of a series of alternating time-point and time-interval descriptions for each model variable. The value of a variable in each state consists of its *qvalue* which is either a landmark from the qspace of the variable or a pair of landmarks which describes a region between two landmarks, and its *qdir* which describes the sign of the time derivative of the variable as $\uparrow$ = increasing, $\ominus$ = steady (0), $\downarrow$ = decreasing, and $*$ = unknown. The description also contains a special landmark called $TIME$ whose landmarks are generated during simulation such that each time-point state gets a unique landmark.

Due to imprecision in both the simulation method and the model space spanned by the SQDE, a single SQDE can generate several different qualitative trajectory components. These are typically grouped into a *behavior tree* (see Figure 2.3).

Semiquantitative Events

Each time-point state in the qualitative trajectory represents a time instant in the prediction. As such, the value of a variable at a time-point state is a precise value. However, since each landmark in the SQDE representation is described by a range (including the $TIME$ landmark) this precise value is represented by a rectangle in the trajectory (see Figure 2.2). The semantics of this rectangle is thus that all trajectories *must* pass through it.

Dynamic Envelopes

We also describe the overall bounds on the qualitative trajectory of a variable by a pair of functions of time called the *dynamic envelope* of the variable. The semantics of this description is that the behavior of all ODEs spanned by the SQDE that match this qualitative

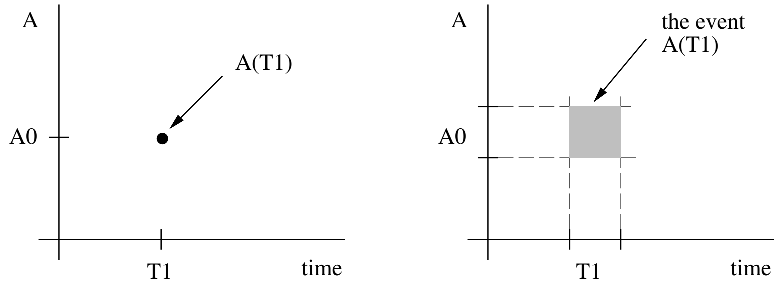


Figure 2.2: An event corresponds to the value of a variable at a time-point. As such, it represents a point in the state space (left graph). Due to uncertainty about the exact coordinates of the point, however, the event appears as a rectangle when viewed quantitatively.

trajectory must remain inside the dynamic envelope. Note that this means that all SQ event rectangles must fall inside the dynamic envelope for the variable.

2.2.2 SQSIM – A Semiquantitative Simulator

SQSIM takes an SQDE and produces a SQ behavior description. It does this by using multiple simulation methods that span the qualitative-to-quantitative spectrum and then combining the results to ensure that the different predictions are consistent with one another. Each simulation method corresponds to a component of the SQ behavior description. They are described in the next sections.

QSIM – Qualitative Simulator

Qualitative inference produces a qualitative trajectory behavior tree from the QDE portion of the model. We use Kuipers' QSIM algorithm [Kuipers, 1984, Kuipers, 1986] to perform this inference.

The simulation alternates between simulating time-point and time-interval states. By applying the rules of qualitative arithmetic [Kuipers, 1994], QSIM generates a set of variable descriptions consistent with the model constraints at the initial state. For instance, if x and y are related by a monotonic increasing function, $qdir(x) = \uparrow$ implies $qdir(y) = \uparrow$. Once a consistent assignment of qmags and qdirs is defined for the initial state, qualitative transition rules determine what will happen in the following time-interval. These transition rules enforce continuity constraints on the behavior trajectory. There are two different sets of transitions – those that hold when the simulation is at a time-point and those that occur when the simulation is in a time-interval. For example, if at time-point $T0$, $qmag(A) = 0$ and $qdir(A) = \uparrow$ then over the time-interval $(T0\ T1)$, $qmag(A) = (0\ FULL)$. Then at time-point $T1$, $qmag(A) = (0\ FULL)$[1] or $FULL$ since A may remain in the interval or increase to the next landmark.

By repeating this process, QSIM produces the set of alternating time-point and time-interval descriptions that comprise the qualitative trajectory portion of the SQ behavior description.

Figure 2.3 shows the result of simulating the QDE in Figure 2.1. It produces three behaviors corresponding to the cases where A overflows (first behavior), reaches a steady-state below the $FULL$ landmark (second behavior), or reaches a steady-state at $FULL$ (third behavior). These behaviors can be organized into a *behavior tree* which highlights where qualitative distinctions between behaviors occur.

One nice property of QSIM is that all real behaviors of the QDE are predicted. This *guaranteed coverage property* is of great importance to design (where we must be sure that all consequences of the design are explored) as well as diagnosis (where we want to be sure that all possible outcomes of our diagnostic hypotheses are considered) and will be preserved in all stages of the SQSIM algorithm.

One weakness of QSIM is that some spurious behaviors may also be predicted. By utilizing a variety of additional qualitative information (for instance, information about the higher-order derivatives of the system) QSIM can often refute branches of the behavior tree, thus reducing the number of spurious behaviors. As we shall see, numerical information can also be used to prune spurious qualitative behaviors thereby reducing prediction ambiguity.

Q2 – Semiquantitative Event Prediction

Since QSIM ignores information at the quantitative level, it generates the set of all possible qualitative behaviors of ODEs spanned by the QDE, but it may also include behaviors of systems that are not part of the family described by the SQDE. Semiquantitative inference, however, does use the quantitative knowledge in the SQDE to attach range information to the qualitative behavior tree. In the process, some branches of the tree may be refuted because their qualitative descriptions are inconsistent with the quantitative information, while on other branches quantitative information refines the prediction.

The Q2 algorithm [Kuipers and Berleant, 1988] produces semiquantitative event descriptions from the SQDE. Since event descriptions represent the instantaneous snapshot of a particular variable at a particular time, the SDE portion of the SQDE can be viewed as a set of equations relating the model variables, and the events must satisfy these equations. Using the range information associated with an event, we end up with a set of interval equations that must hold at each time-point.

[1] If the qmag of a variable is an interval between two landmarks at a time-point state and the qdir is steady, QSIM will create a new landmark in the qspace of the variable between the existing landmarks. Such landmarks are created because points where the derivative reaches zero are often interesting points in the trajectory of a variable and thus should be considered landmarks.

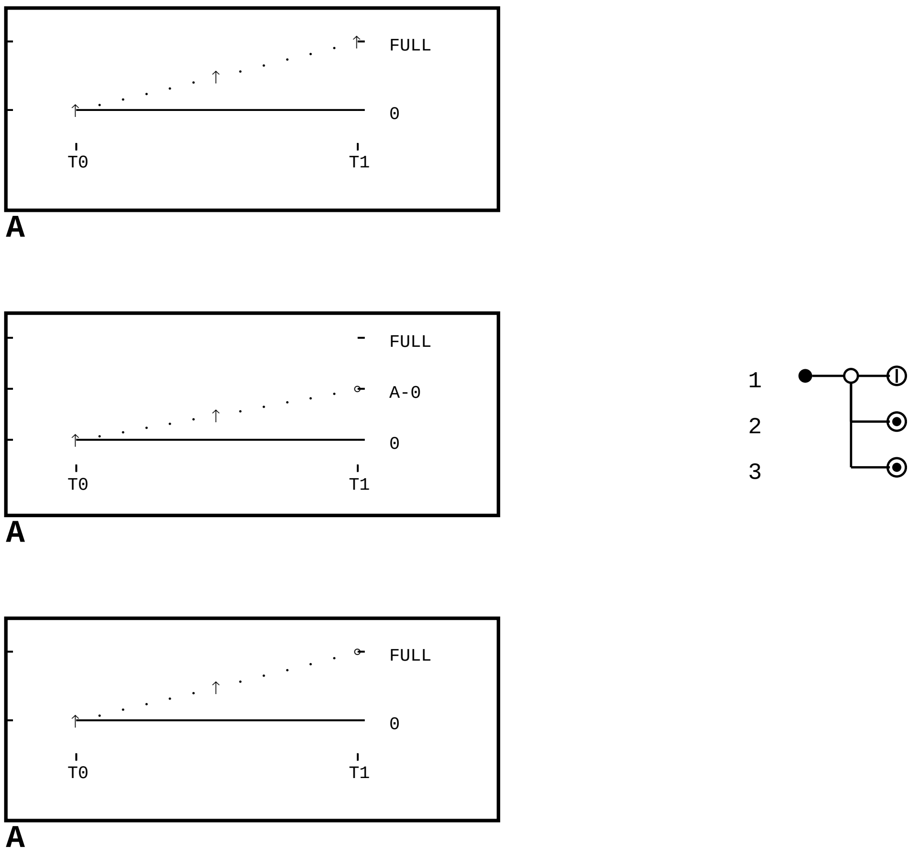


Figure 2.3: Simulation of the single-tank QDE. Three behaviors are produced from one initial state. Only the final state differs in each behavior (see graph at right). Each plot shows one possible behavior of the system. The qmag is determined by the height on the vertical axis and the qdir is determined by the plotted symbols ($\uparrow$ for increasing, $\downarrow$ for decreasing, and $\circ$ for steady). Each symbol is plotted either at a landmark, or midway between two landmarks. The connecting dots are purely visual aids and do not represent a specific trajectory taken by the prediction. In the first behavior, the tank overflows (A reaches $FULL$ and is increasing). In the second behavior, the amount of water reaches some equilibrium value between 0 and $FULL$ to which QSIM automatically assigns the landmark $A0$. In the third behavior, the amount of water reaches equilibrium just as the tank becomes full. All three behaviors are possible given the information in the QDE.

Given an initial assignment of ranges to certain landmarks, we can solve the system of equations using propagation of intervals through model constraints using interval mathematics [Moore, 1979][2]. Whenever we find an equation that computes the value of a model variable, we intersect its existing range with the computed one. If their intersection reduces the range, we assert the updated range and follow the consequences of this new, smaller range. If the intersection is null, we refute the behavior since the interval equations are inconsistent with the qualitative behavior. We continue this process until either the state is refuted or we reach a fixed-point with respect to the ranges of the variables[3].

For example, in the final state of the first behavior in Figure 2.3, we have the event $A(T1) = FULL$. The SDE gives us the equation $A' = c - f(A)$ which must hold for this event. Annotating $FULL$ and c with the ranges given in the SQDE and using $\underline{f}(A) = 8\sqrt{A}$ and $\overline{f}(A) = 10\sqrt{A}$ leads to the following inferences:

$$\begin{aligned} A'(T1) &= [\underline{c}, \overline{c}] - [\underline{f}(\underline{FULL}), \overline{f}(\overline{FULL})] \\ &= [20, 30] - [\underline{f}(80), \overline{f}(100)] \\ &= [20, 30] - [71.5, 100] \\ &= [-80, -41.5] \end{aligned}$$

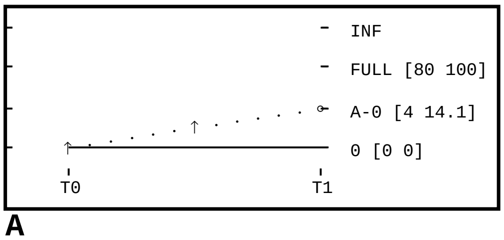
Since the qualitative prediction determines that the qdir of A is $\uparrow$, we know that A' must be greater than zero at $T1$ or $A'(T1) = [0, \infty]$. Intersecting this with the value computed above leads to a null interval, and so we may refute this behavior on the basis of the quantitative information.

The propagation algorithm can be extended across time-intervals by using the Mean Value Theorem to relate the derivative of each state variable to the bounds of the state variable and the bounds on the difference in time between the two time-points. This type of inference reduces the width of the interval associated with the qmag of the time value at each time-point. Q2 thus refines the behavior description produced by QSIM by generating numeric bounds for both events and time-intervals such that:

- Each event determines a rectangle in the trajectory space that all behaviors must pass through.
- Between events, each time-interval description encloses all trajectories of the variable in a *dynamic envelope*, i.e., a pair of functions of time $\underline{A}(t)$ and $\overline{A}(t)$ such that $\underline{A}(t) \leq A(t) \leq \overline{A}(t)$ over the interval.

[2]We propagate intervals through the unknown functional relationships by using the static envelopes defined in the SQDE.

[3]We ensure that a fixed-point will be found in finite time by defining a minimum change ϵ below which a change in a range is considered insignificant.



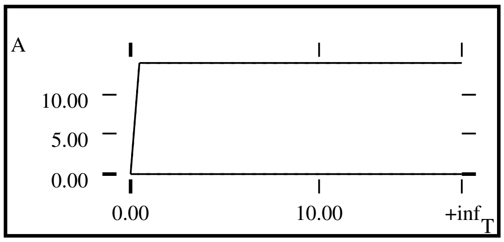


Figure 2.4: Q2 simulation of the single-tank SQDE with static envelope $\underline{f}(A) = 8\sqrt{A}$, $\overline{f}(A) = 10\sqrt{A}$. On the left, we display the same result as in Figure 2.3 but annotated with ranges. On the right, we display the same behavior as a dynamic envelope on a quantitative plot of A versus time.

Figure 2.4 shows the results of running Q2 on the QDE version of the single-tank system. Given the numerical information, Q2 is able to eliminate both behaviors where the amount of water reaches $FULL$.

While Q2 is a powerful inference method for reducing the ambiguity in the qualitative behavior description, it may still permit spurious behaviors and overly wide dynamic trajectory envelopes. There are two reasons for this:

- Interval arithmetic is inherently ambiguous because correlations between variables are ignored. Consider solving the equation $z = x - x$ where x is in the interval $[1, 2]$. Applying the rules of interval mathematics $z = [1, 2] - [1, 2] = [-1, 1]$ whereas the true solution is clearly $z = 0$. The problem is that the interval operator assumes that x can be 2 for one operand and 1 for the other, which is not possible since they must be the same.

- Q2 uses the Mean Value Theorem to describe changes between time-points in a behavior. Since the distance between time-points can be arbitrarily large, this means that Q2's prediction over intervals can be very coarse for the same reason that a numerical simulator with too large a time-step produces poor results.

NSIM – Dynamic Envelope Prediction

Q2 bridges the gap between the qualitative and quantitative representations in the SQDE by using the qualitative model as a framework for generating quantitative interval equations at each qualitative time-point. While interval mathematics can infer numerical bounds on the behavior, there are other purely numerical inference methods that can tighten these bounds even further. We have developed such a method that uses a step-size that fits the numerical properties of the SQDE rather than relying on the distance between qualitative time-points. Our method, called NSIM [Kay and Kuipers, 1993], does this by constructing an ODE system that is guaranteed to bound the behaviors of the SQDE. This system is then simulated to generate a dynamic envelope for all behaviors of the SQDE.

NSIM constructs an *extremal* ODE system from a QDE by minimizing and maximizing each derivative equation in the SQDE. To compute an extremal equation, NSIM rewrites the SQDE using the translations in Table 2.1 to substitute terms. For example, the ODE system

$$\begin{aligned} A' &= c - f(A) \\ B' &= f(A) - f(B) \end{aligned}$$

becomes

$$\begin{aligned} \underline{A}' &= \underline{c} - \overline{f}(\underline{A}) \\ \underline{B}' &= \underline{f}(\underline{A}) - \overline{f}(\underline{B}) \\ \overline{A}' &= \overline{c} - \underline{f}(\overline{A}) \\ \overline{B}' &= \overline{f}(\overline{A}) - \underline{f}(\overline{B}) \end{aligned}$$

Note that while this system can be decomposed into two second-order systems, neither of these systems is a member of the ODE family spanned by the SQDE. This leads to the question of why we don't simply select members of the family that represent "extreme" behavior and simulate them? The answer is that for many systems, there is no single member of the family that can bound the behavior over all time. As a particular example, consider the member of the SQDE system above where $c = 10$, $f(A) = kA$, $k \in [1, 10]$, and $g(B) = 5B$. Figure 2.5 shows the result of simulating this system for different values of k starting from the initial condition $A(0) = 1000, B(0) = 0$. While the behaviors for $k = 1$ and $k = 10$ bound A, no single behavior bounds B over all time. Thus, this simple approach fails.

By applying both a lower and an upper translation for each equation, NSIM generates a new ODE system whose state variables are the upper and lower bounds for the state variables of the SQDE. Thus, we replace the original imprecisely-defined system with a precise ODE system of twice the order that is guaranteed to bound the original SQDE[4]. Since this new system is an ODE system, we can simulate it using conventional simulation methods to obtain a bound on the system[5]. Figure 2.6 shows the result of applying NSIM to the SQDE of the single-tank system. Since the simulation time-step is controlled by the numerical properties of the extremal system, NSIM's bounds are much tighter than those of Q2.

[4]See Appendix A for a proof.

[5]In our work we use the LSODA subroutine from ODEPACK.

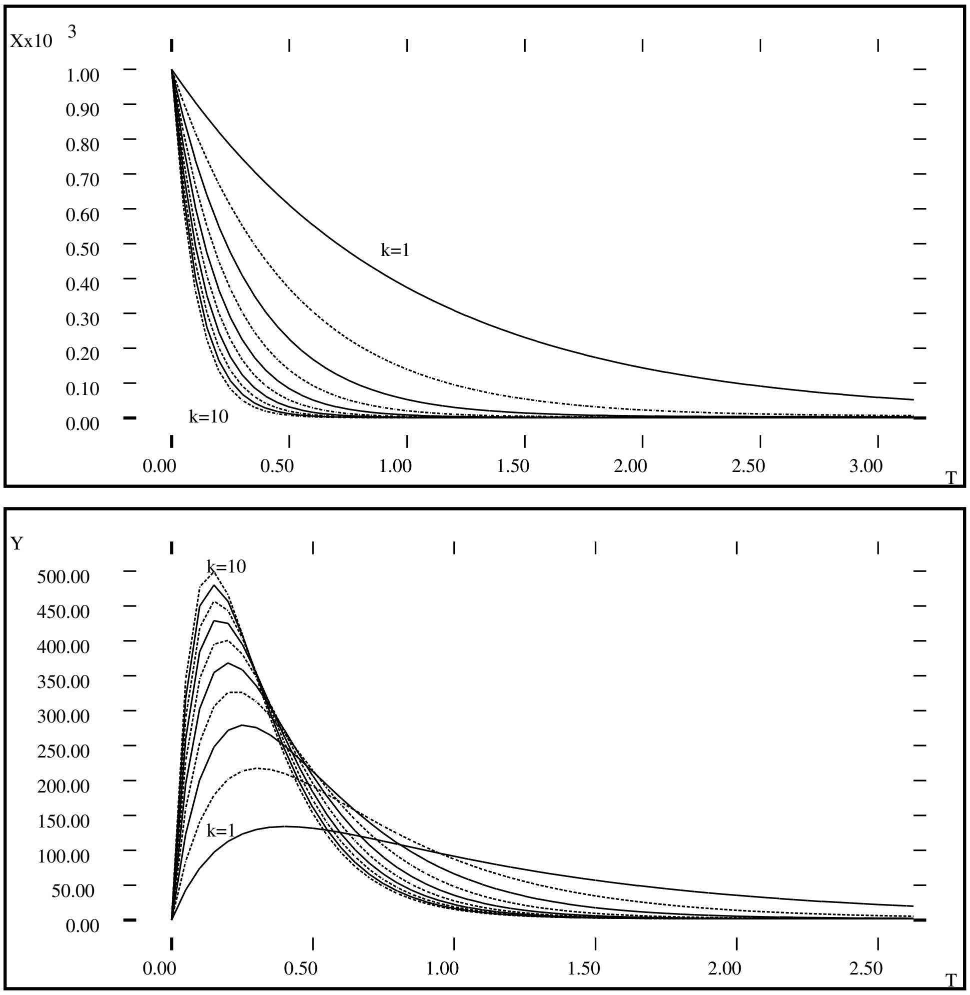


Figure 2.5: A plot of the system $A' = 10 - kA$, $B' = kA - 5B$, $A(0) = 1000$, $B(0) = 0$ for $k = 1, \ldots, 10$. While the behaviors with $k = 1$ and $k = 10$ bound A, no single behavior bounds B.

e	$L(e)$	$U(e)$
c	$\underline{c}$	$\overline{c}$
x_j	$\underline{x}_j$	$\overline{x}_j$
x_i	$\beta(x_i)$	$\beta(x_i)$
$A+B$	$L(A)+L(B)$	$U(A)+U(B)$
$A-B$	$L(A)-U(B)$	$U(A)-L(B)$
$A\times B$	$min(C_{Mult}(A,B))$	$max(C_{Mult}(A,B))$
$A\div B$	$min(C_{Div}(A,B))$	$max(C_{Div}(A,B))$
$-A$	$-U(A)$	$-L(A)$
$M^{+}(A)$	$\underline{M}^{+}(L(A))$	$\overline{M}^{+}(U(A))$
$M^{-}(A)$	$\underline{M}^{-}(U(A))$	$\overline{M}^{-}(L(A))$

$$\begin{array}{rrcl} \text{where} & C_{Mult}(A,B) & = & \{L(A)\cdot L(B), U(A)\cdot L(B), L(A)\cdot U(B), U(A)\cdot U(B)\} \\ \text{and} & C_{Div}(A,B) & = & \{L(A)/L(B), U(A)/L(B), L(A)/U(B), U(A)/U(B)\}. \end{array}$$

Table 2.1: Translation table for extremal expressions of the equation $x_i' = f_i(\mathbf{x}_i)$. Let $\beta(f_i)$ be the desired bound on x_i' ($\beta = L$ or $\beta = U$). The table is applied recursively to the subexpressions of f_i. The symbol x_j is any state variable other that x_i, c is a constant, M^+ and M^- are monotonic functions, $\underline{c}$ and $\overline{c}$ return the lower or upper range values of c, $\underline{M}^*$ and $\overline{M}^*$ return the lower or upper functional envelope of the monotonic function. For the state variable x_i, the bound is the same as that for f_i.

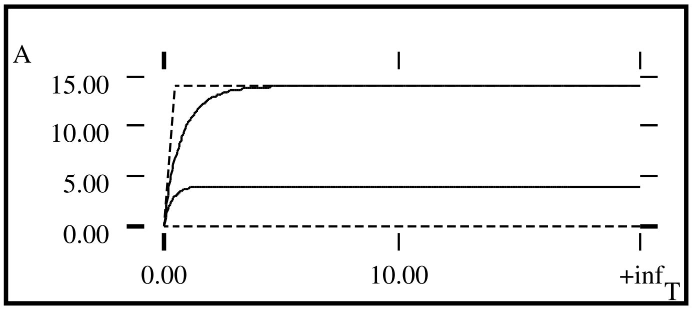


Figure 2.6: Simulation of the single-tank SQDE with NSIM. The solid curve is the NSIM dynamic envelope. The dotted box is the Q2 dynamic envelope defined by the initial state and equilibrium state events. The NSIM envelope is much tighter than in Figure 2.4 because the ODE solver determines the simulation step-size based on the numerical properties of the extremal system.

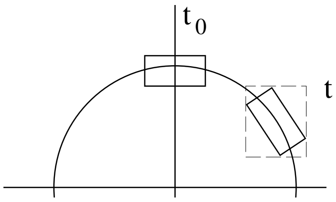


Figure 2.7: The wrapping effect. As we travel through the phase space from t_0 to t_1, the state hypercube orients itself with respect to the trajectory. Since intervals are described with respect to the coordinate axes, we must "wrap" the new bound with a new hypercube oriented with respect to the original coordinate system.

While NSIM produces tighter bounds than Q2 in many cases, it can also generate spurious behaviors in the form of overly-wide dynamic envelopes. These spurious behaviors stem from:

- Representing uncertainty with intervals.
  One can precisely describe the state-space uncertainty of a system at a given time by a set of points. Unfortunately, this type of description is intractable with respect to representation and inference since the set of points is normally infinite. A useful description must represent such sets finitely. NSIM uses hypercubes as an uncertainty representation. Such a simplification makes description and inference tractable, however it also admits spurious behaviors. Given that the hypercube $\mathbf{X}(t)$ describes the state of uncertainty at t, the set $\mathbf{S}$ which contains all states reachable from $\mathbf{X}(t)$ at time $t+h$ is not in general a hypercube. Thus, we must enclose the reachable states at $t+h$ in a hypercube so that they are representable. We thus include extra states in $\mathbf{X}(t+h)$ that are not reachable from $\mathbf{X}(t)$. As simulation proceeds, this set may include more and more states that are not reachable from the initial state, thus leading to bounds that diverge with time. This is known as the "wrapping problem" in the interval simulation literature [Moore, 1979] (see Figure 2.7). Note that wrapping is the property of any finite representation of a set of states. Depending on the system, is sometimes possible to reduce the effect of wrapping by using other representations such as ellipsoids [Moore, 1979] or by using coordinate transformations that orient the hypercube with respect to the trajectory of the system (see Section 2.4).

- Ignoring correlations in the SQDE.
  A model may contain multiple references to a function, however, the extremization process may make these references refer to different functions. For example, if we have a model

$$x' = c - f(x) \qquad y' = f(x) - g(y)$$

our intention is that the two occurrences of $f(x)$ refer to the same function. The extremal system for the lower bounds of x and y, however, will look like

$$\underline{x}' = \underline{c} - \overline{f}(\underline{x}) \qquad \underline{y}' = \underline{f}(\underline{x}) - \overline{g}(\underline{y}).$$

Unlike the original system, the second term in the first equation is no longer identical to the first term in the second one. NSIM is thus making predictions about models that are not part of the original model space and so these predictions admit spurious states. In some cases, model correlation can be eliminated by rewriting the SQDE so that only one reference to a function is made (see [Hüllermeier, 1995]). In the above example, we can introduce a new variable $z = x + y$ thus transforming the system into

$$x' = c - f(x) \quad z' = x' + y' = c - g(z - x)$$

which makes only one reference to f. The lower bound is now

$$\underline{x}' = \underline{c} - \overline{f}(\underline{x}) \qquad \underline{z}' = \underline{c} - \overline{g}(\underline{z} - \overline{x})$$

- Ignoring correlations between variable bounds.
  The extremization process, coupled with the use of the interval representation for uncertainty, causes NSIM to produce an ODE system that can, in some cases, ignore the correlation between upper and lower bounds of a variable in the original system. While the SQDE exists in an order-n state-space, the extremal system generated by NSIM is of order-$2n$. If the trajectories generated by each ODE and initial condition entailed by the SQDE and state hypercube at t_0 are mapped into $2n$-space, they will occupy an order-n subspace since, they are subject to the added constraint $\forall_{i,t}\ \underline{x}_i(t) = \overline{x}_i(t)$. The extremal system, however, is not restricted to considering points in this subspace. Thus the extremal system may compute solutions based on points in $2n$-space where $\underline{x}_i(t) \neq \overline{x}_i(t)$ which cannot happen in any of the ODEs. For example, consider the harmonic oscillator

$$x' = v \qquad v' = -x$$

which has an extremal system defined as

$$\underline{x}' = \underline{v} \qquad \underline{v}' = -\overline{x} \qquad \overline{x}' = \overline{v} \qquad \overline{v}' = -\underline{x}. \tag{2.1}$$

For this system, the derivative of the extremal system state vector $[\underline{x}, \overline{x}, \underline{v}, \overline{v}]^T$ depends simultaneously on both upper and lower bounds of x and v. Thus, the correlations $\underline{x} = \overline{x}$ and $\underline{v} = \overline{v}$ are explicitly ignored (assuming that the bounds on x and v are of non-zero width). In contrast, consider the system

$$x' = c - x \qquad y' = x - y$$

which has the extremal system

$$\underline{x}' = \underline{c} - \underline{x} \qquad \underline{y}' = \underline{x} - \underline{y} \qquad \overline{x}' = \overline{c} - \overline{x} \qquad \overline{y}' = \overline{c} - \overline{y}$$

In this case, the extremal system separates into two second-order ODEs. The derivative of $[\underline{x}, \underline{y}]$ depends only on the lower bounds of x and y and the derivative of $[\overline{x}, \overline{y}]$ depends only on their upper bounds. Because neither system requires the upper and lower bounds of a system simultaneously, the correlations $\underline{x} = \overline{x}$ and $\underline{y} = \overline{y}$ are not ignored. The practical result is that the extremal system for this system produces tighter bounds than extremal system (2.1).

SQDEs that lead to separable extremal systems have the property that their bounds are determined by simulating "corners" of the hypercube. This class is identical to the class of quasi-monotonic systems [Walter, 1970] for which it can be shown that simulating the corners of the hypercube leads to an optimal bound [Corliss, 1995a].

These sources of spurious behavior suggest that we can improve the NSIM prediction by using different uncertainty representations or simulation methods. In Section 2.4 we will examine some of these possibilities. While such methods can reduce the width of dynamic envelopes in some cases, they require specialized simulation engines. It is worth noting that by casting the bounding problem as the solution to an ODE system, NSIM can take advantage of the speed and efficiency of existing numerical ODE simulators as well as exploit future advances in ODE simulation technology.

Combining Inference at Different Levels

The predictions produced by QSIM, Q2, and NSIM use information in the SQDE in different ways. QSIM provides the qualitative shape of the trajectories. Q2 augments the qualitative events with interval ranges. NSIM produces dynamic envelopes directly from the imprecise numerical information. While each method guarantees that all real behaviors entailed by the original SQDE are predicted, they each admit spurious behaviors as well. QSIM generates them due to uncertainty in qualitative arithmetic. Q2 fails to refute them due to uncertainty in interval mathematics and the uncertainty in the distance between qualitative time-points. NSIM generates them because the process which generates the extremal system can ignore correlations between variables. By combining these inference methods, SQSIM is able to reduce the imprecision in predictions by ensuring consistency across the predictions.

Figure 2.8 shows the flow of information through SQSIM. SQSIM uses QSIM as the basis for simulation. At every qualitative time-point Q2 and NSIM are run to produce events and dynamic envelopes that describe the qualitative behavior up to the time-point. Then, each of the combination methods described below are run until a fixed-point with respect to Q2 events is found. If no contradiction is detected, SQSIM generates the successor states of the current state and repeats the process.

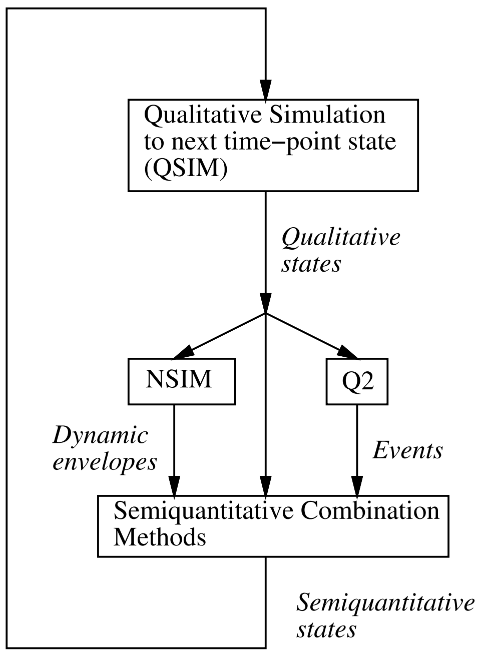


Figure 2.8: Information flow through SQsim. QSIM generates a time-point state which Q2 and Nsim annotate with events and dynamic envelopes. This information is then passed through the semiquantitative combination methods. If the state is consistent, the cycle repeats with QSIM generating all successors to the state.

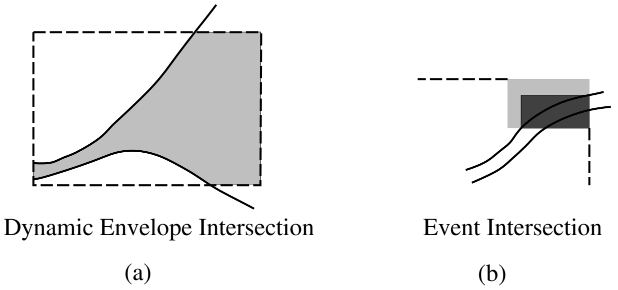


Figure 2.9: Q2 and NSIM intersection methods. We can intersect both Q2 events and dynamic envelopes with NSIM dynamic envelopes. For dynamic envelope intersection, the Q2 dynamic envelope (dashed box) and NSIM envelope (solid line) produce the shaded envelope. For event intersection a Q2 event (light rectangle) and the NSIM envelope (solid line) produce a reduced event description (darker rectangle).

SQSIM uses four methods for combining QSIM, Q2, and NSIM inferences:

Dynamic envelope intersection We can intersect the Q2 dynamic envelopes between two events with the NSIM prediction to yield a tighter bound. Figure 2.9(a) demonstrates this process. In this figure, the NSIM prediction starts out tight and diverges as time increases. In contrast, the Q2 dynamic envelope is a constant band over the interval. By intersecting both predictions, we obtain a behavior that is better than Q2 alone at the beginning of the interval and better than NSIM for later values of time. If there is a time at which the NSIM and Q2 dynamic envelopes do not intersect, the entire behavior can be refuted since their predictions must be consistent over the entire behavior.

Event intersection NSIM's dynamic envelopes can also be intersected with Q2 event descriptions. In this case, a behavior can be refuted if the NSIM envelope does not pass through any point in the event rectangle. If the event and dynamic envelope do overlap, it is possible to refine the event based on the intersection. In Figure 2.9(b), for example, the NSIM envelope restricts the starting time for the event as well as the maximum value for the event variable. By running Q2 on this reduced event description, SQSIM may further refine other events in the behavior.

Extremum detection For some models, the NSIM dynamic envelope will exhibit clear qualitative properties. For example, the envelope may contain a “bump” where all trajectories reach a local minimum or maximum (see Figure 2.10). In such cases, the QSIM description should also contain a bump in the corresponding region of the behavior. The extremum detector locates minima and maxima within the dynamic envelope of a variable and

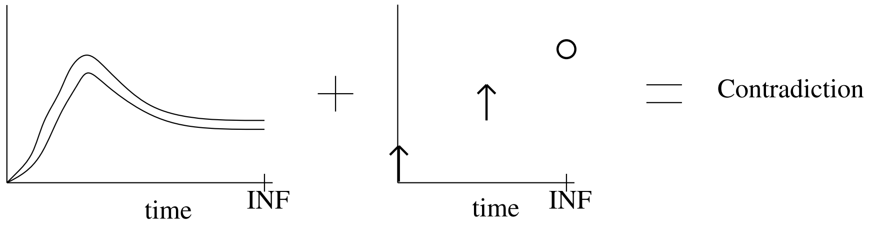


Figure 2.10: Extremum detection ensures that an NSIM-detected extremum has a corresponding qualitative extremum. The dynamic envelope in the left graph shows that all behaviors must pass through a maximum before reaching quiescence. The qualitative plot on the right has no such maximum and therefore there is a contradiction.

ensures that there is a corresponding critical point in the QSIM description. Note that this test can only be used when the NSIM envelopes contain a recognizable extremum. This is because a dynamic envelope of non-zero width does not normally provide information about the shape of the underlying trajectories. Therefore, every behavior that passes through the envelope *may* pass through a local extremum. It is only when there is an extremum in the envelope that we are guaranteed that every behavior *must* pass through an extremum.

Order reduction The Q2 dynamic envelope is a coarse trajectory envelope of the behavior. NSIM can produce a more precise bound, however its envelope can also diverge with time. By using the Q2 dynamic envelope, we can control NSIM so that it does not simulate an upper (or lower) bound when that bound is already at the maximum (or minimum) determined by Q2. Instead, we can set this bound's derivative to zero, thus reducing the order of the extremal system. Such an operation can greatly reduce the width of the dynamic envelope. This method can also be applied to the sign of the derivative – i.e., if the derivative of an upper (or lower) bound takes on a sign inconsistent with the Q2 prediction, the corresponding bound in the extremal system can be set to zero. Figures 2.19 and 2.20 in Section 2.3 demonstrate the use of order reduction to improve a dynamic envelope where bounds on the derivative of a state variable are known.

Resimulation SQSIM operates on a state-by-state basis, using all events from the current and previous states to refine the behavior. Sometimes, however, the bounds on events may be shrunk as a result of events generated at later time-points. This is especially true for equilibrium states (which occur at the end of behaviors) since in these states Q2 can produce tighter event bounds because the derivatives of all the state variables are known to be precisely 0. Furthermore, it is sometimes possible to bound the time over which derivatives of state variables have constant qdir and use this information to guide order-reduction.

Resimulation is the process of analyzing a completed behavior for these types of non-local properties and then running all combination methods described above. Figures 2.19 and 2.20 in Section 2.3 illustrate the use of resimulation.

2.3 A Complex Example – The CSTR

In this section, we present an example of using SQSIM to determine the behavior of a chemical system. The intent is to demonstrate the behavior, as well as the strengths and limitations of SQSIM.

The system that we simulate is an adiabatic Continuously-Stirred Tank Reactor (CSTR) with an irreversible exothermic first-order reaction $A \rightarrow B$. The equations of the system are :

$$\begin{aligned} \frac{dC_A}{dt} &= \frac{C_{Ai} - C_A}{\tau} - k_0 e^{-E/T} C_A \\ \frac{dT}{dt} &= \frac{T_i - T}{\tau} - h_r k_0 e^{-E/T} C_A \end{aligned} \tag{2.2}$$

where C_A is the concentration of reactant A in the tank, C_{Ai} is the inlet concentration of A, T is the tank temperature, T_i is the temperature of the inlet stream, τ is the residence time, h_r is the heat of reaction, k_0 is the rate constant, and E is the activation energy. This system is highly nonlinear due to the exponential containing T.

In the process of design or monitoring such a system, we may be interested in answering the following questions:

- What are the locations of equilibrium points in the state space?
- What are the qualitative trajectories of the system?
- What are the final values of the state variables and how do they compare to the initial values?
- Do the trajectories remain inside an acceptable region of the state space?
- Where and when do extrema occur in the trajectories of the state variables?
- How sensitive is the behavior to uncertainty in model parameters?

Our example follows the work of Dalle Molle [Dalle Molle, 1989] who made a detailed study of the adiabatic CSTR for the case where the system is in equilibrium and then C_{Ai} is negatively perturbed. He analyzed the numerical and qualitative properties of this system and determined that there are three equilibrium states for an ODE instance with the values described in Table 2.2. He then examined the effect of different perturbations from these

State number	C_A	T	Comments
1	0.9332	353.4	Low conversion
2	0.6090	418.2	Unstable
3	0.1566	508.7	High conversion

Table 2.2: The three possible equilibrium states of the CSTR given the parameters $C_{Ai} = 1.0$, $T_i = 340$, $\tau = 10$, $k_0 = 10000$, $E = 5000$, $h_r = -200$.

equilibrium states. In all cases T decreases over time, however there are four possible qualitative behaviors of C_A. For our analysis, we focus on a decrease on C_{Ai} from 1.0 to 0.9. In this case, there are two different behaviors depending on which steady-state we start from (see Figure 2.11).

In the next section, we analyze the behavior of the precise model using SQsim. The only uncertainty that we allow is in the initial state description. Then, in the following section, we will analyze the behaviors produced when we allow parametric uncertainty in the model.

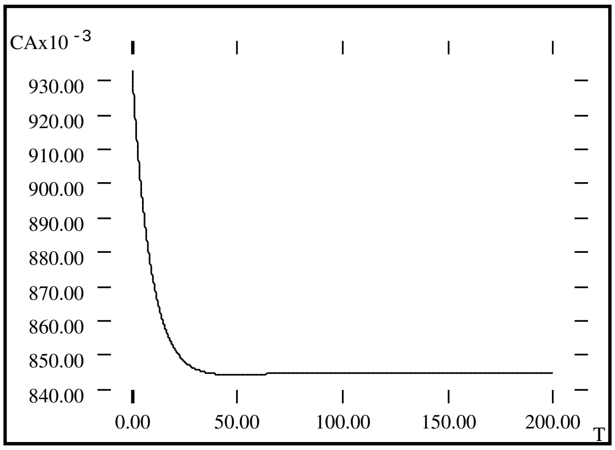
2.3.1 The Precise CSTR under SQsim

In this section, we consider the precise model of the CSTR simulated from an imprecise initial state.

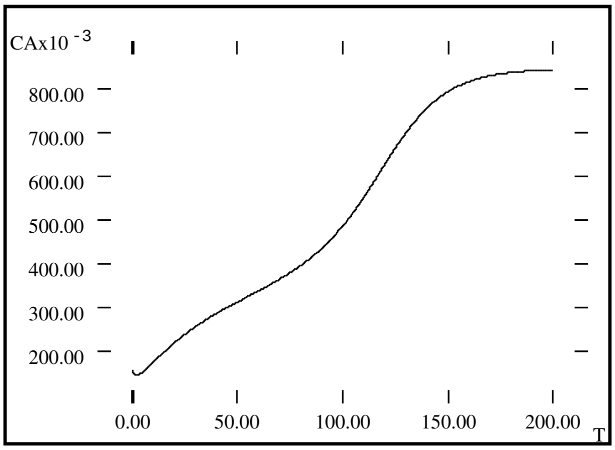
Qualitative Inference

Figure 2.12 gives an overview of the results in this section and is intended as a road map for following the analysis of the example.

We begin with the SDE described by Equation 2.2 which defines the model variables and the relations between them. From this, we form a QDE by defining the quantity spaces for the model variables and then use QSIM to generate the qualitative equilibrium states. The quantity space for C_A contains the landmarks 0, CAI (which represents the input stream concentration), and INF (which represents the landmark at ∞, a value larger than any finite real number) while T contains the landmarks 0, TI (which represents the input stream temperature) and INF. These quantity spaces restrict C_A and T to positive values. By asserting that the qdirs of C_A and T are steady and that all constants are at fixed landmark values, QSIM is able to determine the equilibrium values of the qmags and qdirs for all model variables. QSIM determines that there is a single qualitative state consistent with the three equilibrium states in Table 2.2 (see Figure 2.13). This is because all three quantitative equilibria are located such that C_A is between 0 and $C_{Ai}(= 1.0)$ and T is between $T_i(= 340)$ and ∞.



(a)



(b)

Figure 2.11: The two behaviors of C_A when simulated numerically from (a) steady-state 1 and (b) steady-state 3. Note the local minimum in (a) at $t \approx 50$ and in (b) at $t \approx 6$.

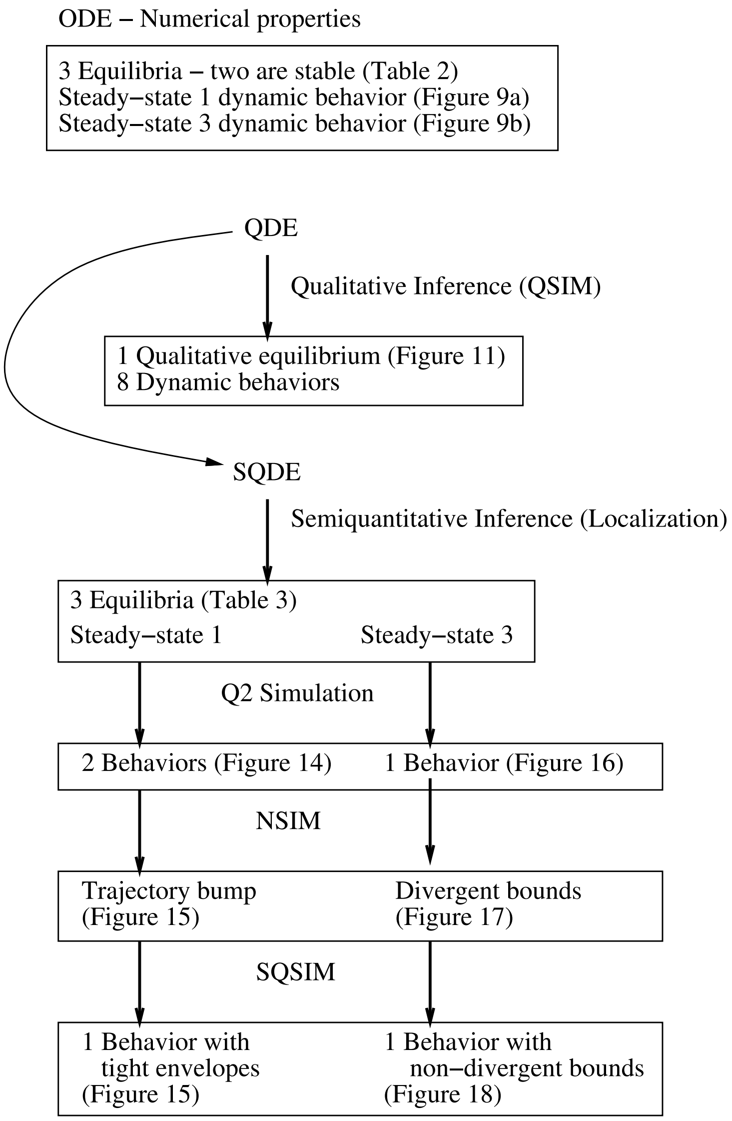


Figure 2.12: An overview of the analysis of the precise CSTR model using SQSIM. The user provides the SQDE (which contains the QDE) together with a (possibly-imprecise) initial condition. The prediction of equilibria and dynamic behavior is produced automatically by SQSIM.

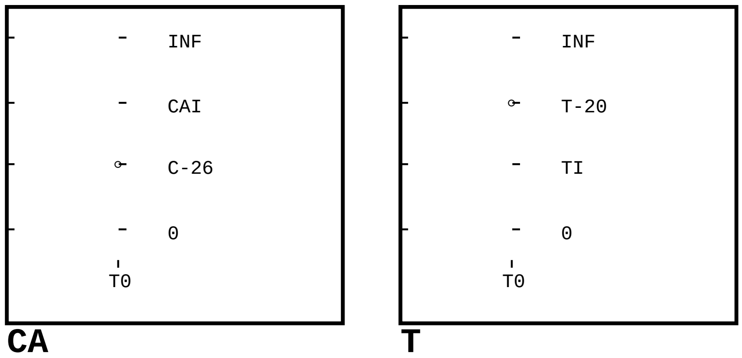


Figure 2.13: The qualitative equilibrium of the CSTR. Because the the three equilibria in Table 2.2 are qualitatively identical, QSIM generates a single equilibrium state.

Next, we perturb C_{Ai} by asserting that its qmag is less than its qmag in the equilibrium state. QSIM generates 8 behaviors from three possible initial states (which are due to uncertainty in the the value of $C_{Ai} - C_A$ in the first equation which may be positive, zero, or negative). In all behaviors T decreases monotonically over time and C_A has one of four qualitative shapes (see Figure 2.14) which include the two numerically produced behaviors in Figure 2.11. The other two behaviors occur when C_{Ai} is perturbed to values other than 0.9.

Semiquantitative Inference

Next, we add quantitative information to produce the SQDE. Since we are modeling the precise ODE, the semiquantitative information consists of zero-width intervals for each model parameter. Our first goal is to identify the equilibria of the system. We define the bounds on C_A and T to be $[0, +\infty]$ which encodes the knowledge that the equilibria lie in the positive portion of the state space of the system. SQSIM is then able to isolate the equilibria of the system by using a technique called Localization on the initial state. Localization addresses the "gap" problem with interval representations. For example, the narrowest interval representing $\sqrt{4}$ is $[-2, 2]$ even though it is really only the endpoints of the interval that are true solutions. There is thus a gap in the interval. Localization attempts to find gaps in the domain of a variable. Given a variable to localize and an initial range, the Localizer selects a point within the range of the variable and using Q2 to derive a contradiction, indicating that the initial range contains a gap. When a gap is found, it splits the state description in two – one part where the range of the variable falls below the gap and one part where the range falls above the gap. It then applies Q2 to each of these more precise states which may result in further convergence of ranges. If not, it attempts

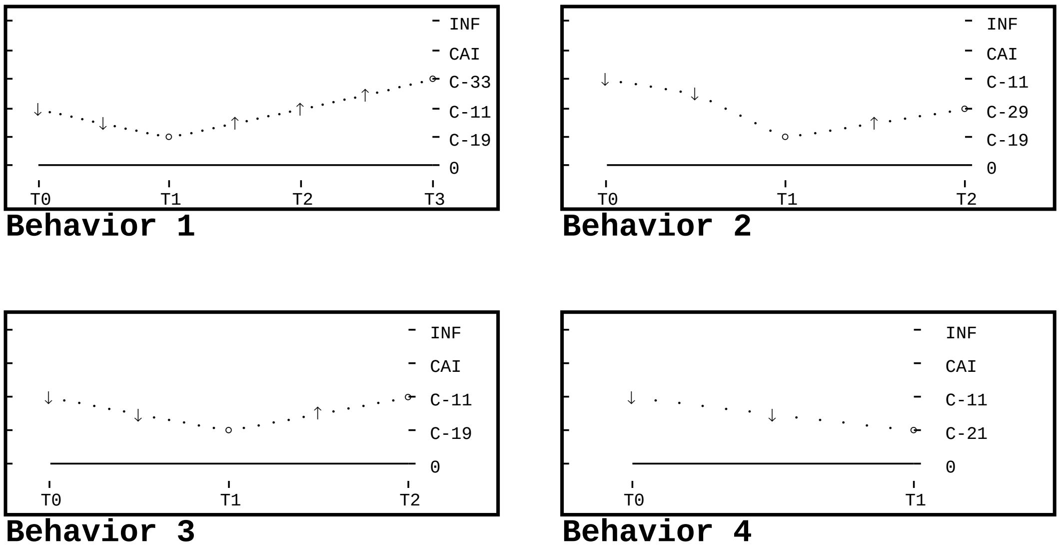


Figure 2.14: The four qualitative behaviors of C_A that are possible for the CSTR when decreasing C_{Ai}. Only behaviors 1 and 2 are possible when C_{Ai} decreases to 0.9.

to localize each subrange (see Figure 2.15). The process terminates when either the final subranges are points or after three attempts to find a gap fails[6].

For the CSTR model starting with ranges of $[0, \infty]$ for C_A and T, Localization identifies the three numerically-determined equilibria from Table 2.2 (see Table 2.3). Note that the equilibrium state descriptions have non-zero width. This is because the Q2 portion of SQsim uses a parameter ϵ to determine when the ranges associated with landmarks reach a fixed-point so as to prevent excessive interval propagation. By decreasing ϵ, we can increase the precision with which SQsim localizes the equilibria. Because Localization maintains the Guaranteed Coverage Property (see Section 2.2.2), for any non-zero ϵ the equilibrium description will contain some uncertainty. In contrast, the numerically-determined equilibria in Table 2.2, while precise, are not guaranteed to be correct[7]. This is because the representation (precise floating-point numbers) cannot represent the imprecision inherent in the numerical simulator. For some systems, this error could lead to incorrect results (for example, if the system is chaotic and thus sensitive to initial conditions). By simulating from an imprecise initial condition, we maintain the Guaranteed Coverage Property of

[6] Localization is akin to Target Interval Splitting [Berleant and Kuipers, 1992] which attempts to refute the *ends* of intervals.

[7] In fact, the values in the table are incorrect since they do not fall inside the guaranteed Q2 envelope. In this case, this is because of round-off in the table.

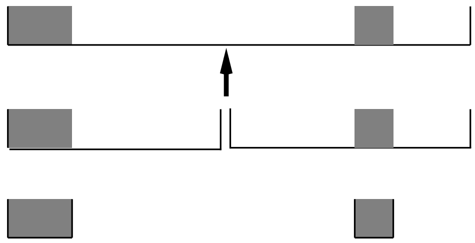


Figure 2.15: Localization splits an interval with a gap in it into smaller intervals by locating a small gap and then forming two intervals on either side of it. These new intervals can then be refined using Q2 to further reduce their range (gray squares). Since finding a small gap is less subject to correlation error than is examining the entire interval, Localization can quickly shrink overly-wide bounds.

State number	C_A	T
1	$[0.933, 0.934]$	$[353.358, 353.360]$
2	$[0.609, 0.610]$	$[418.194, 418.196]$
3	$[0.156, 0.157]$	$[508.667, 508.673]$

Table 2.3: The equilibria of the CSTR computed using SQSIM Localization with a Q2 ϵ of 10^{-6}. By decreasing ϵ, Q2 will produce tighter bounds. The true equilibria are guaranteed to lie within the bounds.

SQSIM and do not "miss" potentially relevant behaviors[8].

Decreasing C_{Ai} starting near steady-state 1 Starting from the Localization-determined equilibrium interval $T = [353.358, 353.360]$ and $C_A = [0.933, 0.934]$, the dynamic simulation leads to the two behaviors in Figure 2.16. Q2 refutes the other 6 branches because they are not consistent with phase space location of steady-state 1. Dalle Molle determined that only the first behavior is real for the precise model. Q2 is unable to determine this because its prediction for the upper bound of the local minimum of C_A is the same as that for its final value and so the two behaviors look equivalent.

Applying NSIM to the SQDE results in the envelopes in Figure 2.17. Note that they fall completely inside the Q2 envelope and that there is a trajectory minimum near $t = 50$. By using the SQSIM extremum detector, we can eliminate the spurious behavior. Thus SQSIM predicts a single qualitative behavior and a tight dynamic envelope which suffices to closely bound any dynamic properties of interest.

[8]If we *did* choose a precise initial condition, we would get a single precise behavior since the NSIM prediction of a precise SQDE and initial condition is a zero-width trajectory.

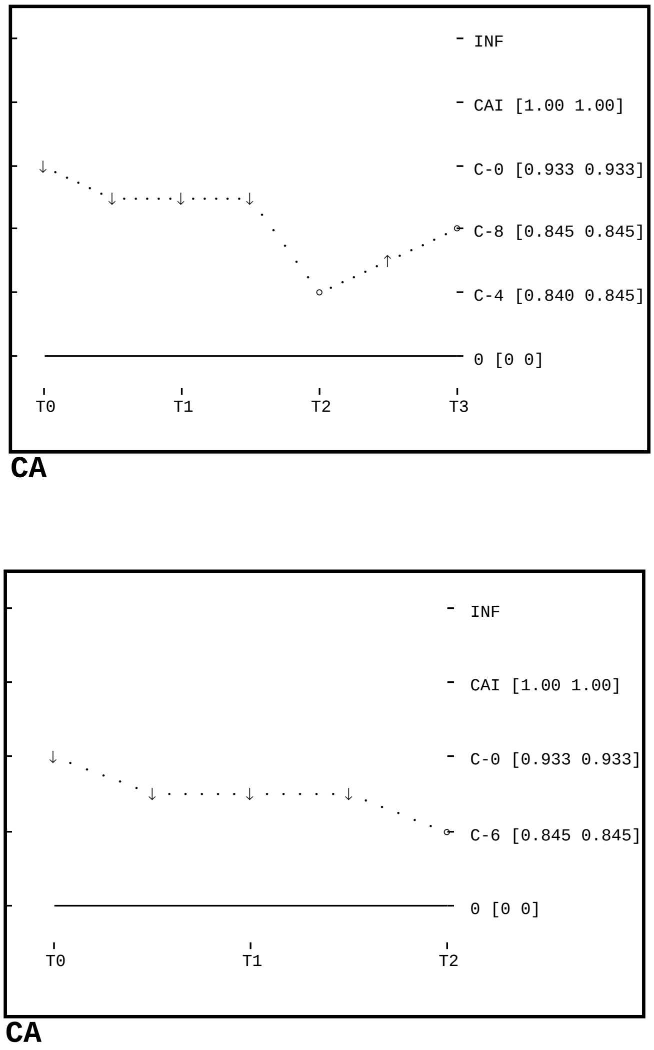
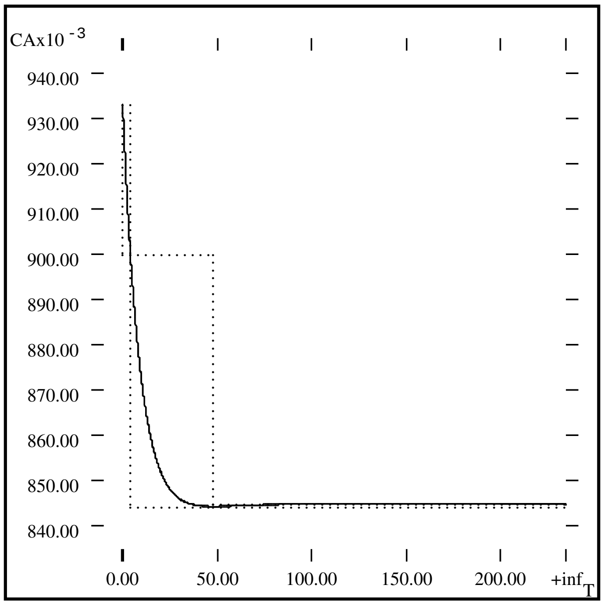


Figure 2.16: Q2 prediction of C_A from steady-state 1 when $C_{Ai} = 0.9$. Two behaviors are produced. The ranges of all time-points after $T0$ are $[0, \infty]$. Q2 is unable to distinguish these two cases because, in the first behavior, the ranges associated with the events $C_A(T2)$ and $C_A(T3)$ are such that the depth of the minimum could possibly be zero, thus making the first and second behaviors identical.



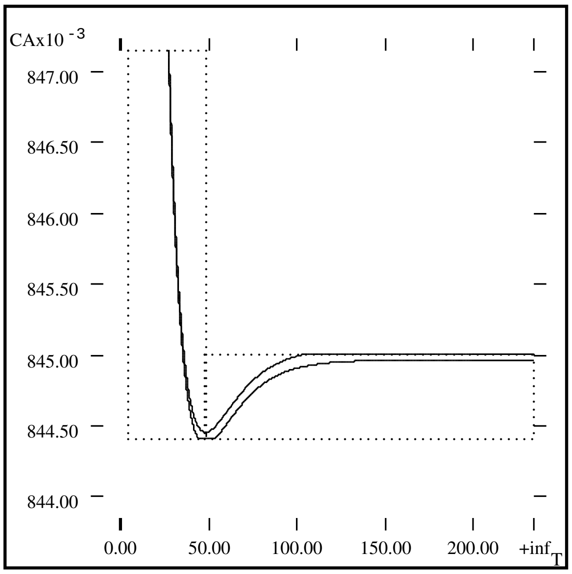


Figure 2.17: NSIM simulation of the example in Figure 2.16 together with the Q2 prediction (dashed boxes). The spurious behavior is eliminated by SQSIM due to the detection of the minimum in the envelope (see blowup in lower graph). The dynamic envelopes remain tight over the entire prediction.

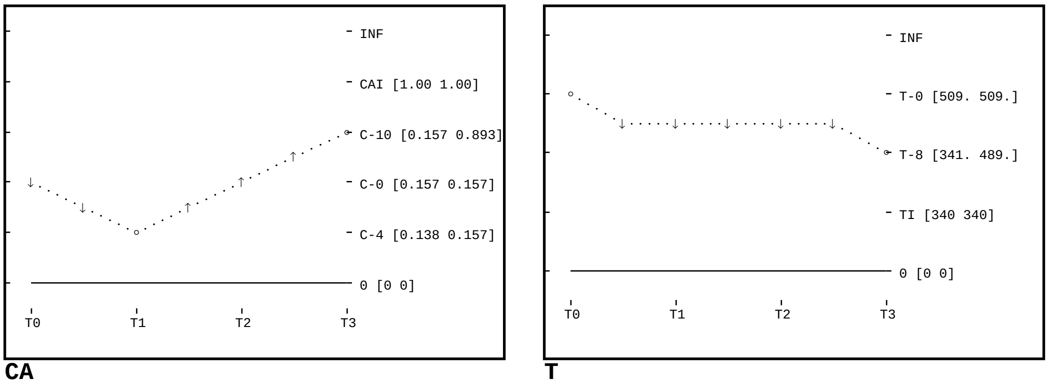


Figure 2.18: Q2 prediction of C_A from steady-state 3 when $C_{Ai} = 0.9$. The ranges of all time-points after $T0$ are $[0, \infty]$. One behavior is produced. Although concentration clearly increases, the final value is still rather large. By using Target Interval Splitting we can reduce this range, although dynamic properties, like the time when the minimum concentration is reached are still poorly predicted.

Decreasing C_{Ai} starting near steady-state 3 Starting from the Localization-determined equilibrium interval $T = [508.667, 508.673]$ and $C_A = [0.156, 0.157]$, Q2 produces the single behavior shown in Figure 2.18. While it has refuted the other 7 behaviors, the final range of C_A ($[0.157, 0.893]$) is rather large and is bounded below by the initial concentration, so we cannot predict that the final concentration increases. By manually applying Target Interval Splitting to the final state, we find that the final value of C_A is $[0.845, 0.846]$ so the concentration has definitely increased. We still have very weak information about the time of the inverse response.

Applying NSIM only, we obtain the envelopes in Figure 2.19. For $C_{Ai} = 0.9$, the extremal system is quite stable up until $t \approx 20$ where it exhibits a high degree of divergence. This is due to ignoring correlations in the extremal system which turns the negative feedback in the original system into positive feedback. For other values of C_{Ai}, this divergence is not so sharp and NSIM alone produces tighter envelopes[9]. Once we reach the final state, however, it is clear from the qualitative behavior that C'_A is always positive after reaching the minimum at $T1$. Therefore, SQSIM can resimulate using order-reduction to prevent C'_A from becoming negative. This significantly shrinks the final envelopes as shown in Figure 2.20.

Referring back to the questions that we are interested in for this model, if we start with a SQDE of the CSTR model, SQSIM can determine the following:

[9]Experimentation reveals that $C_{Ai} \leq 0.7$ leads to tight predictions.

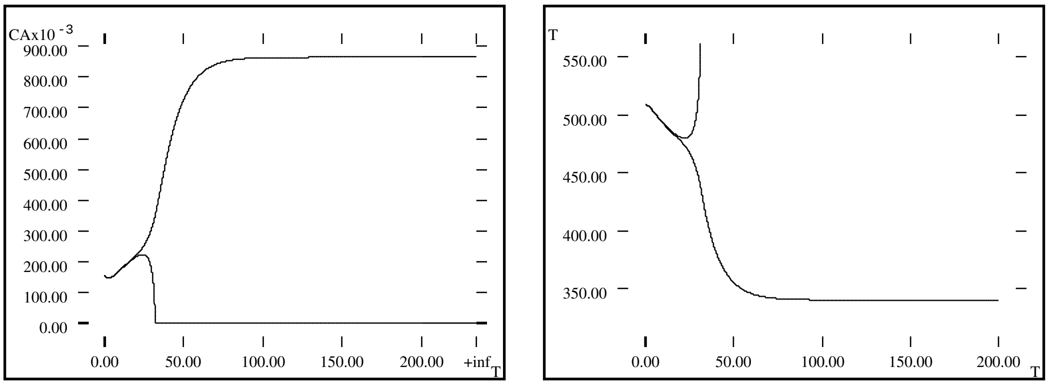


Figure 2.19: NSIM envelopes for the example in Figure 2.18. These bounds diverge after $t \approx 20$ due to the extremal system ignoring correlations between upper and lower bounds of the state variables.

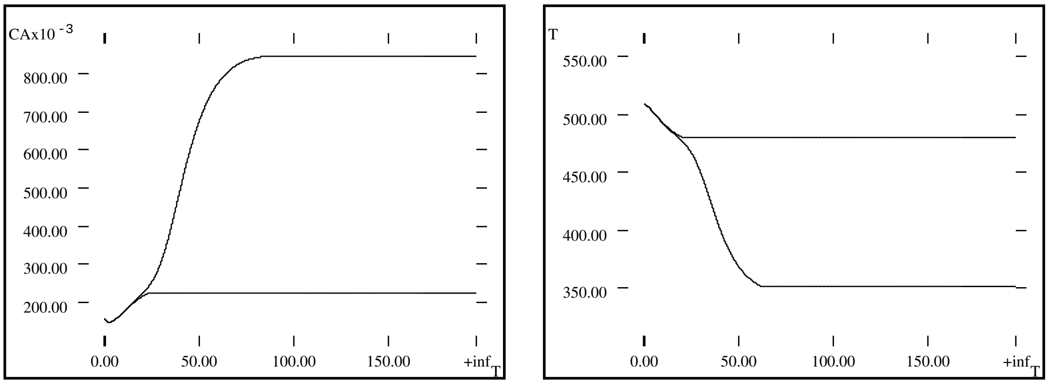


Figure 2.20: SQSIM envelopes for the example in Figure 2.18. By using order-reduction and resimulation based on and the inference at qualitative time-point $T3$ that C'_A is never negative after C_A reaches its minimum value, SQSIM can produce a tighter dynamic envelope.

- It automatically identifies the location of the equilibria.
- It produces single qualitative trajectories from the stable equilibria.
- When input concentration is perturbed from 1.0 to 0.9, SQSIM determines that the final concentration of C_A increases from steady-state 1 and decreases from steady-state 3.
- From steady-state 1 it finds a tight envelope over the the entire trajectory. From steady-state 3, the trajectory widens as time increases, but remains bounded.
- The location and deviation of the inverse responses are tightly bounded.

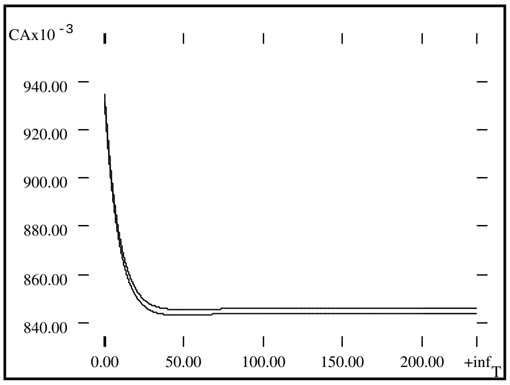
SQSIM is thus able to automatically infer the relevant qualitative and quantitative properties of this system, together with behavioral bounds over time, even in the presence of uncertainty in the initial condition.

2.3.2 The Imprecise CSTR

So far, we have considered only initial state uncertainty. In this section, we examine the more interesting case when there is uncertainty in the model by varying the values of the model parameters k_0, h_r, and E_a by $\pm 1\%$ and $\pm 5\%$. Varying these quantities examines cases where there is uncertainty about the reaction rate, heat of reaction, and activation energy. While SQSIM can handle the more general case of functional imprecision, for this model, parametric imprecision is sufficient to examine the properties of our method.

Decreasing C_{Ai} near steady-state 1 For 1% error in k_0 we obtain the results in Figure 2.21. Table 2.3.2 summarizes the results from varying all three variables independently by ± 1 and ± 5 percent. Even with model uncertainty, the behavior of the system is still tightly bounded by the dynamic envelopes and that prediction uncertainty increases with model imprecision. Target Interval Splitting yields tight initial state and final state equilibria. The increase in uncertainty prevents us from eliminating the "non-bump" behavior[10], so we have two possible qualitative behaviors. However, because the bounds are tight, the possible deviation of the inverse response is still tightly bounded. For similar amounts of uncertainty, k_0 and h_r have a similar effect on prediction uncertainty. Varying E_a has a greater effect due to its position in the exponential of the differential equations.

[10] We have not done an analysis to determine if the behavior without inverse response is spurious for the uncertain model.



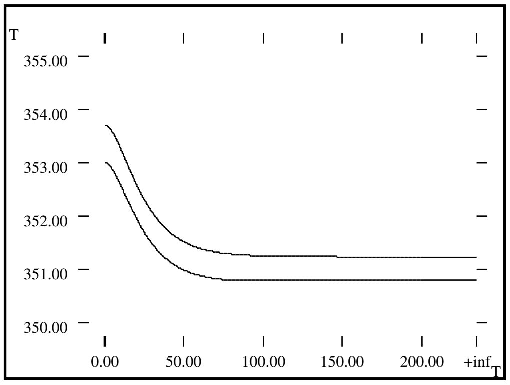


Figure 2.21: SQSIM simulation of the CSTR from steady-state 1 and $C_{Ai} = 0.9$ with 1% error in k_0. There are two qualitative behaviors (one with a local minimum in C_A and one without), however the dynamic envelopes are the same in each case.

Case		Initial State		Final State	
var	error	C_A	T	C_A	T
–	0%	[0.933, 0.934]	[353.358, 353.360]	[0.845, 0.846]	[350.998, 350.999]
k_o	±1%	[0.931, 0.935]	[353.025, 353.706]	[0.843, 0.847]	[350.795, 351.225]
k_o	±5%	[0.923, 0.942]	[351.797, 353.253]	[0.839, 0.851]	[349.934, 352.201]
h_r	±1%	[0.932, 0.935]	[353.048, 353.681]	[0.844, 0.846]	[350.794, 351.557]
h_r	±5%	[0.928, 0.938]	[351.904, 355.105]	[0.842, 0.848]	[350.000, 352.117]
E_a	±1%	[0.909, 0.948]	[350.449, 358.008]	[0.823, 0.859]	[348.326, 355.254]
E_a	±5%	[0.899, 0.979]	[344.294, 360.001]	[0.823, 0.879]	[344.166, 355.251]

Case		Max Envelope Width	
var	error	C_A	T
–	0%	0.001	0.284
k_o	±1%	0.004	0.446
k_o	±5%	0.011	2.265
h_r	±1%	0.002	0.434
h_r	±5%	0.006	2.126
E_a	±1%	0.044	9.626
E_a	±5%	0.078	15.835

Table 2.4: Decreasing C_{Ai} to 0.9 from steady-state 1 for the imprecise CSTR for 1 and 5% error in k_0, h_r and E_a. In all cases, two qualitative behaviors are generated, however the dynamic envelopes for each behavior are identical. For each parameter, increasing the uncertainty increases the width of the initial state, final state, and envelope width.

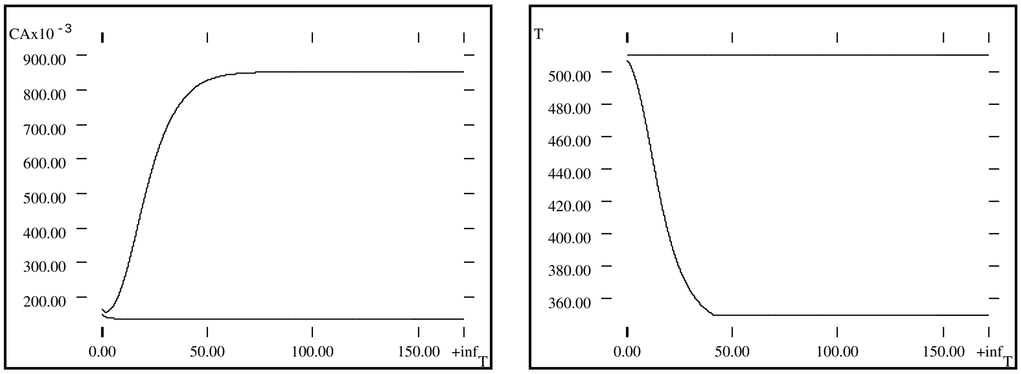


Figure 2.22: SQSIM envelopes for the CSTR from steady-state 3 and $C_{Ai-} = 0.9$ with 1% error in k_0. There is one qualitative behavior. The dynamic envelopes over time are very wide, however.

Decreasing C_{Ai} near steady-state 3 The results for a 1% error in k_0 are shown in Figure 2.22. As was demonstrated in the precise case, this system exhibits divergence about this equilibrium when $C_{Ai} = 0.9$. The increase in uncertainty prevents SQSIM from precisely determining the location and time of the minimum in C_A, however, the envelopes still restrict the trajectory so we can determine the minimum possible value of this event. Table 2.5 summarizes the results from varying all three parameters by 1 and 5%. As was the case near steady-state 1, prediction uncertainty increases with increasing model imprecision.

We conclude this section by summarizing the strengths and limitations of our method. By combining qualitative and quantitative inference, SQSIM produces predictions in terms of both qualitative trajectories and envelopes. The qualitative prediction serves two purposes:

- It provides a basis for distinguishing trajectories that fall within the envelopes. The predictions are thus more detailed than are predictions based on envelopes alone.
- It provides information that the NSIM numerical simulator can use to further refine the dynamic envelopes.

This synergy between representations can often produce very tight bounds on the behavior of an imprecisely-defined ODE system. The degree of uncertainty in prediction is directly related to the degree of imprecision in the ODE model and reduces to zero (and gives a single, zero-width trajectory) in the case of a precise ODE model.

Under some circumstances, however, the properties of the imprecisely-defined system are such that our inference methods produce dynamic envelopes that are overly wide given

Case		Initial State		Final State	
var	error	C_A	T	C_A	T
–	0%	$[0.156, 0.157]$	$[508.667, 508.673]$	$[0.845, 0.846]$	$[350.998, 350.999]$
k_0	$\pm1\%$	$[0.149, 0.165]$	$[507.087, 510.116]$	$[0.843, 0.847]$	$[350.775, 351.227]$
k_0	$\pm5\%$	$[0.143, 0.176]$	$[505.000, 511.207]$	$[0.299, 0.856]$	$[348.883, 460.074]$
h_r	$\pm1\%$	$[0.139, 0.172]$	$[505.000, 513.859]$	$[0.844, 0.846]$	$[350.789, 351.212]$
h_r	$\pm5\%$	$[0.120, 0.235]$	$[488.750, 524.542]$	$[0.201, 0.854]$	$[348.856, 486.789]$
E_a	$\pm1\%$	$[0.133, 0.189]$	$[502.239, 513.269]$	$[0.224, 0.861]$	$[347.938, 513.269]$
E_a	$\pm5\%$	$[0.074, 0.900]$	$[360.100, 525.000]$	$[0.108, 0.868]$	$[346.571, 498.245]$

Case		Max Envelope Width	
var	error	C_A	T
–	0%	0.690	148.386
k_0	$\pm1\%$	0.697	154.847
k_0	$\pm5\%$	0.713	161.186
h_r	$\pm1\%$	0.707	158.612
h_r	$\pm5\%$	0.734	169.298
E_a	$\pm1\%$	0.728	165.331
E_a	$\pm5\%$	0.794	178.430

Table 2.5: Decreasing C_{Ai} to 0.9 from steady-state 3 for the imprecise CSTR for 1 and 5% error in k_0, h_r and E_a. For each parameter, increasing the uncertainty increases the width of the initial state, final state, and envelope width.

the amount of model imprecision. In Section 2.4, we examine related methods for clues on how to further improve bounds.

2.3.3 Other Examples

In addition to its application in predicting the behavior of the CSTR, SQSIM has also been applied to a variety of other systems, including a model of a semiconductor vacuum system during pumpdown that was used in a monitoring system [Kay, 1991, Kay and Kuipers, 1993]. Because of the highly nonlinear nature of the model, a small error in modeling can have a large effect on the predicted behavior, and so it is very difficult to accurately predict the dynamic behavior of the system with a precise model. Thus, the typical method for diagnosing faults in these systems requires waiting until the system reaches equilibrium (which can take considerable time). By explicitly representing the modeling error in an SQDE and using SQSIM, it was possible to generate a robust dynamic prediction and monitor for faults while the pressure of the vacuum system was still changing, thus reducing fault detection time.

2.4 Related Work

There are very few general methods for producing bounded predictions from imprecise ODE models. In this section we examine one technique – interval ODE simulation – and describe how it compares with SQSIM.

Interval ODE simulation has been an active area of research for over thirty years. Interval ODE simulators focus on the problems of arithmetic and round-off error caused by using computer arithmetic to solve *precise* ODE systems. Interval ODE methods consider initial state and parameter uncertainty, but they normally assume that they are due primarily to machine precision and thus are small. An interval ODE simulation problem can thus be described as

> Given a precise, parameterized order-n ODE system of the form $x' = f(x, c)$ with state vector x and constant vector c and an initial bound on each component of x and c, determine a hypercube $X(t)$ such that $\forall_{i=1,n}\ x_i(t) \in X_i(t)$ for all t.

Most interval simulation methods rely on the concept of inclusion monotonicity to determine enclosures on the trajectories of state variables over time. Inclusion monotonicity is the property that if X is an interval vector and $F(X)$ is an interval function[11] then if $Y \subseteq X$ then $F(Y) \subseteq F(X)$. Typical algorithms [Moore, 1979, Corliss, 1995b] work in two stages:

[11]We say that F is an interval extension of f if $F(X_1, \ldots, X_n) = f(x_1, \ldots x_n)$ as long as the interval $X_i = [x_i, x_i]$ (i.e., it is of zero width).

1. Compute a coarse bound on $x(t)$ over some time interval $I = [t_j, t_j + h]$.
   Assume that at t_j, $X(t_j)$ bounds the trajectory. If we can find an A such that $X(t) \subseteq A$ for $t \in I$, then A will be a coarse bound for the trajectory over I. The trick is to find A (and a suitable h) so that the bound is tight and h is large. One method for finding such a bound is to use a first order Taylor expansion:

   $$A = X(t_j) + [0, h]F(t_j, X(t_j)) + \alpha$$

   where h and α are selected based on the problem.

   Because A is selected using only information at t_j, we must verify that it does indeed bound the trajectory over all of I. Inclusion monotonicity ensures that we have such a bound if, for example

   $$X(t_j) + [0, h]F([t_j, t_j + h], A)) \subseteq A$$

   If the inclusion is not met, we may either shrink h or widen A so as to maintain the inclusion.

2. Compute a tighter bound based on A.
   We can compute a tighter bound over I by expanding the Taylor Series for X and bounding the remainder with A. One such method is to use[12]

   $$X(t) = X(t_j) + \sum_{i=1}^{p} \frac{X^{(i)}(t_j)}{i!}(t - t_j)^i + \frac{A^{(p+1)}}{(p+1)!}(t - t_j)^{p+1}$$

   for $t \in I$ where the Taylor coefficients are computed recursively starting with $X(t_j)$ and A (see [Moore, 1979] for details).

Interval ODE simulators are also susceptible to the effects of ignoring correlations between variables, particularly the wrapping effect. Modern simulators such as AWA [Lohner, 1988] minimize this effect by using a coordinate system that that follows the trajectory of the system. AWA does this by producing an approximate solution to the ODE system and using the angle between this solution and the coordinate axes to rotate the frame of reference for computing intervals. This can greatly reduce the wrapping effect in some cases.

2.4.1 Comparing SQsim and Interval Simulation

In this section, we compare interval ODE methods to both the Nsim subsystem as well as the entire SQsim system. Both Nsim and interval ODE methods produce dynamic envelopes

[12]The symbol $Y^{(i)}$ represents the ith time derivative of Y (i.e., $\frac{d^iY}{dt^i}$).

for an SQDE model. The key difference is in the focus of each method. Interval simulation methods focus on accounting for error in machine arithmetic and approximation error caused by using floating point numbers to represent real numbers in the prediction. The models and initial conditions are assumed to be precise (or very nearly so). NSIM concentrates on representing and reasoning with models that have significant imprecision and is less concerned with arithmetic error, relying on the underlying ODE solver to intelligently deal with this problem[13].

One important advantage that NSIM has is that it can simulate models that have functional uncertainty in the form of monotonic static envelopes. Functional uncertainty cannot be allowed in a method that uses Taylor expansions since computing Taylor coefficients requires a precise mathematical form.

Another NSIM advantage is that since the extremization process produces an ODE system, conventional ODE methods can be used to produce bounds. This permits the use of state-of-the-art numerical ODE solvers that may behave better with stiff systems as well as Differential Algebraic simulators that can include algebraic constraints in addition to the differential equations.

One weakness of NSIM is that extremization removes the original structure of the SQDE and replaces it with a new ODE structure. This new structure will often have very different dynamics from the original system so it is difficult to determine what correlations are being ignored in the extremal system. In particular, using coordinate transformations to combat wrapping becomes more difficult.

The main advantage of SQSIM over interval ODE simulation is the integration of qualitative and quantitative inference to improve predictions. By using qualitative cues, SQSIM is able to better guide NSIM into producing tighter bounds. The precise CSTR model is a good example of this since interval ODE simulators produce divergent predictions for this model (as does NSIM alone). By using qualitative information, however, SQSIM is able to eliminate the divergence in the predictions. This suggests that existing interval ODE simulators could benefit from using an SQSIM-like approach to simulation.

2.4.2 Other Methods

Interval simulation methods were examined in detail in the previous section because many of the problems with reasoning about SQDEs also affect interval ODE simulation and solutions have been investigated and implemented by researchers in this field. There are, however, several other methods that provide useful perspectives on the problems of SQDE simulation. In the interest of completeness, we list some of these below.

[13] Of course, since the extremal system is a precise ODE system, there is no reason that NSIM could not use an interval ODE solver to ensure that this source of error is accounted for.

- The NIS system [Vescovi *et al.*, 1995] is similar to NSIM in that it produces dynamic envelopes for systems describable by SQDEs. The method calculates the derivative of the SQDE system at each time point using interval arithmetic and then uses this interval bound in a custom-designed simulator to compute envelopes on the solution (current methods include Euler and Runge-Kutta). While it admits a slightly wider class of functions than NSIM, NIS is also subject to the same limitations caused by using an interval uncertainty representation, wrapping, and ignoring model correlation. Since it is a purely numerical method, it cannot exploit qualitative information about the model to prevent envelope divergence as SQSIM does.

- The system of Danès [Danès *et al.*, 1993] simulates precise models whose inputs are imprecisely-defined and satisfy certain controllability and observability requirements (which means that it can simulate only a subset of the SQDEs that SQSIM can). It is noteworthy because it casts the bounding problem as a set of optimization problems in the state space of the system. This allows it to avoid manipulating state-space uncertainty representations, thus sidestepping the correlation problems of most other methods.

- The paper by Corliss [Corliss, 1995a] describes a variety of other bounding methods for ODE systems and analyzes conditions under which they are applicable.

There is also a large body of theoretical work that provides a formal basis for many bounding methods:

- The field of differential inequalities [Walter, 1970] has relevance to bounding behaviors of imprecisely-defined ODE systems. This field analyzes conditions under which bounds exist for an ODE system. Several algorithms (e.g., [Markov and Angelov, 1986]) follow directly from this work.

- The field of differential inclusions [Aubin and Cellina, 1984] views the SQDE prediction problem from a framework where state and model uncertainty are directly represented by sets of state-space points and ODE models, respectively, and analyzes conditions under which bounds for the system exists. In [Hüllermeier, 1995], Hüllermeier examines SQDE simulation as done by NSIM in this framework and suggests methods for using alternate representations that can sometimes produce better bounds at greater computational expense.

There has also been applied work in behavior bounding in the area of VLSI simulation by Zukowski [Zukowski, 1986]. When treated as a resistive and capacitive switching network, digital VLSI circuit ODE systems are quasi-monotonic (or nearly so) and so simplified bounding methods can be used to predict circuit behavior by bounding the true circuit equations by linear ones and simulating these.

All of these methods treat the bounding problem in the quantitative domain and thus are similar in spirit to NSIM. This suggests that these methods could be improved by integrating them into a qualitative-to-quantitative framework with SQSIM. Since each method has its strengths and weaknesses, it would then be possible to select the appropriate method given the nature of the SQDE itself.

Chapter 3

Refining Monotonic Model Spaces

The next two chapters deal with refining model spaces by using observations. This process, known as *system identification* typically involves using the observations to search the model space for a "best fit" to the data. In this chapter, we present MSQUID, a method for defining and searching a model space composed of memoryless monotonic functions using a neural network-inspired model space. In addition to providing a general method for fitting functions using weak prior knowledge, MSQUID is also a key component of the more general SQUID system (described in Chapter 4) that can refine ODE model spaces.

3.1 Introduction

Techniques for estimating a functional relationship between an output variable y and the input vector $\mathbf{x}$ typically assume that there is some deterministic function g and some random variate ϵ such that $y = g(\mathbf{x}) + \epsilon$, where ϵ is a normally distributed, mean zero random variable with variance σ^2 that represents measurement error and stochastic variations. The estimate is computed by using a parameterized fitting function $f(\mathbf{x}; \boldsymbol{\theta})$ (which defines a model space in terms of the unknown $\boldsymbol{\theta}$) and then using regression analysis to determine the values $\widehat{\boldsymbol{\theta}}$ such that $f(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \approx g(\mathbf{x})$.

Traditional regression methods require knowledge of the form of the estimation function f. For instance, we may know that body weight is linearly related to amount of body fat. We may then determine that $f(weight; \theta) = weight \cdot \theta$ is an appropriate model. Neural network methods have been developed for cases where no information about f is known. This may be the case if f models a complex process whose physics is poorly understood. Such networks are known to be capable of representing any functional relationship given a large enough network. In this chapter, we consider the case where some intermediate level of knowledge about f is known. In particular, we are interested in cases where we have knowledge of the monotonicity of f in terms of the signs of $\frac{\partial f}{\partial x_k}$ where $x_k \in \mathbf{x}$. For example, we might know that outflow from a tank monotonically increases with tank pressure. This type of knowledge is prevalent in qualitative descriptions of systems, so it makes sense to take advantage of it.

Since the estimate is based on a finite set of data, it is not possible for it to be exact. We therefore require our estimate to have an associated confidence measure which takes into account the uncertainty introduced by the finite sample size. For our semiquantitative representation, we are therefore interested in deriving an envelope that bounds all possible functions that could have generated the data-stream with some probability p.

This chapter describes MSQUID, a method for estimating and computing bounding envelopes for multivariate functions based on a set of data and knowledge of the monotonicity of the functional relationship. First, we describe our method for computing an estimate $f(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \approx g(\mathbf{x})$ based on a neural network that is constrained to produce only monotonic functions. Second, we describe our method for computing a bounding envelope, which is based on linearizing the estimation function and then using F-statistics to compute a simultaneous confidence band. Third, we present several examples of function fitting and its use in semiquantitative simulation using QSIM. Next, we analyze the performance of MSQUID with respect to the number of units in the hidden layer, noise level, and complexity of the monotonic function. Finally, we discuss related work in neural networks and non-parametric analysis and describe some applications of this method.

3.2 Computing the Estimate

Computing the estimate of g requires that we make some assumptions about the nature of deterministic and stochastic portions of the model. We assume that the relationship between y and $\mathbf{x}$ is $y = g(\mathbf{x}) + \epsilon$ where ϵ is a normally distributed random variable with mean 0 and variance σ^2. Other assumptions, such as different noise probability distributions or multiplicative rather than additive noise coupling could be made. The above model, however, is fairly general and it permits us to use powerful regression techniques for the computation of the estimate and its envelope. For situations where variance is not uniform, we can use variance stabilization techniques to transform the problem so that it has a constant variance.

In traditional regression analysis, the modeler supplies a function $f(\mathbf{x}; \boldsymbol{\theta})$ together with a dataset to a least-squares algorithm which determines the optimal values for $\boldsymbol{\theta}$ so that $f(\mathbf{x}; \widehat{\boldsymbol{\theta}}) \approx g(\mathbf{x})$. The estimated value of y is then $\hat{y} = f(\mathbf{x}; \widehat{\boldsymbol{\theta}})$. In our case, however, the only information available about f is the signs of its n partial derivatives $\frac{\partial f}{\partial x_k}$ where $x_k \in \mathbf{x}$, so no explicit equation for f can be assumed. One way to work without an explicit form for f is to use a neural net function estimator. Figure 3.1 illustrates a network for determining $\hat{y}$ given a set of inputs $\mathbf{x}$. The network has three layers. The input layer contains one node

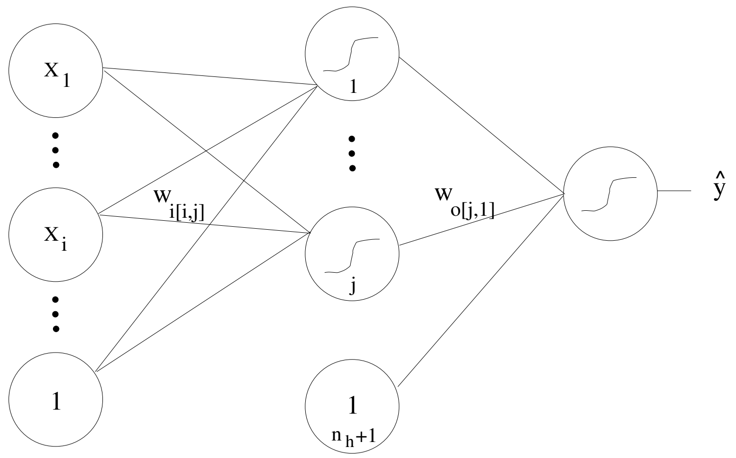


Figure 3.1: A neural net-based function estimator. This three-layer net computes the function $\hat{y} = \sigma\left(\sum_{j=1}^{n_h}\left[w_{o[j,1]}\sigma\left(\sum_{i=1}^{n} w_{i[i,j]}x_i + w_{i[n+1,j]}\right)\right] + w_{o[n_h+1,1]}\right)$.

for each element x_k and one bias node set to a constant value of 1. The hidden layer consists of a set of n_h nodes which are connected to each input variable as well as to the bias input. The output layer consists of a single node which is connected to all the hidden nodes as well as to another bias value which is fixed at 1[1]. All nodes use sigmoidal basis functions and all connections are weighted. In our notation, $w_{i[i,j]}$ represents the connection from input x_i to hidden node j and $w_{o[j,1]}$ represents the connection from hidden node j to the output layer[2]. This network represents the function

$$\begin{aligned} \hat{y} &= \sigma(s + w_{o[n_h+1,1]}) \\ s &= \sum_{j=1}^{n_h}\left[w_{o[j,1]}\sigma\left(\sum_{i=1}^{n} w_{i[i,j]}x_i + w_{i[n+1,j]}\right)\right] \end{aligned}$$

where $\sigma(x)$ is the sigmoidal function $\frac{1-e^{-x}}{1+e^{-x}}$. We can compute the weights by solving the nonlinear least squares problem

$$\min_{\mathbf{w}} \sum_i (y_i - \hat{y}_i)^2$$

where $\hat{y}_i = f(\mathbf{x}_i; \mathbf{w})$ and $\mathbf{w}$ is a vector of all weights. Cybenko [Cybenko, 1989] and others have shown that with a large enough number of hidden units, any continuous function may be approximated by a network of this form.

[1]The bias terms permit the estimation function to shift the center of the sigmoid.

[2]The weight $w_{i[n+1,j]}$ represents the connection to the input bias and the weight $w_{o[n_h+1,1]}$ represents the connection from the hidden layer bias node to the output node.

One drawback of using this estimation function is that it can over-fit the given data. This results in the estimate following the random variate ϵ as well as the deterministic part of the model which means that we get a poor approximation of g[3]. We therefore reduce the scope of possible functions to include only monotonic functions. To do this, note that if f is monotonically increasing in x_k, then $\frac{\partial f}{\partial x_k}$ must be positive. By constraining the weights, we can force this derivative to positive for all $\mathbf{x}$, insuring that the resulting function is monotonic. The derivative in question is

$$\begin{aligned}
\frac{\partial f}{\partial x_k} &= \sigma'(s + w_{o[n_h+1,1]}) \cdot \frac{\partial s}{\partial x_k} \\
\frac{\partial s}{\partial x_k} &= \sum_{j=1}^{n_h} w_{o[j,1]} \sigma' \left(\sum_{i=1}^{n} (w_{i[i,j]} x_i) + w_{i[k,j]} \right) w_{i[k,j]}
\end{aligned}$$

Since σ' is positive for all values of its domain, $\frac{\partial f}{\partial x_k}$ will be positive if $\frac{\partial s}{\partial x_k}$ is positive. This will be the case if, for each k,

$$\forall_{1 \leq j \leq n} w_{o[j,1]} \cdot w_{i[k,j]} \geq 0. \tag{3.1}$$

If the partial derivative is negative, then the inequality is reversed. While this result holds for any k, for the remainder of this dissertation we restrict our interest to single input models (i.e., $k = 1$). Appendix B contains a proof that a network constrained in this manner is capable of representing any monotonic function.

This set of additional constraints on weights causes the network to produce only monotonic functions. We may determine an estimate $\hat{\mathbf{w}}$ for the weights by solving the problem

$$\min_{\mathbf{w}} \sum_i (y_i - \hat{y}_i)^2 \tag{3.2}$$

$$\text{subject to } \forall_{1 \leq j \leq n_h} w_{o[j,1]} \cdot w_{i[1,j]} \geq 0.$$

These n_h constraints transform the least squares problem into a constrained nonlinear optimization problem. While more difficult to solve than the unconstrained problem, there are still a number of methods based on numerical methods. In our work we use the OPT algorithm [Biegler, 1985, Biegler and Cuthrell, 1985].

To compute the estimate, we need to find the best value for n_h. We determine n_h by repeatedly solving the optimization problem for increasing values of n_h starting at 1 and continuing until there is no improvement in the sample standard error

$$\frac{\sum_i (y_i - \hat{y}_i)^2}{n - p - n_h}$$

[3] Visually, the estimate will try to pass through every data-point.

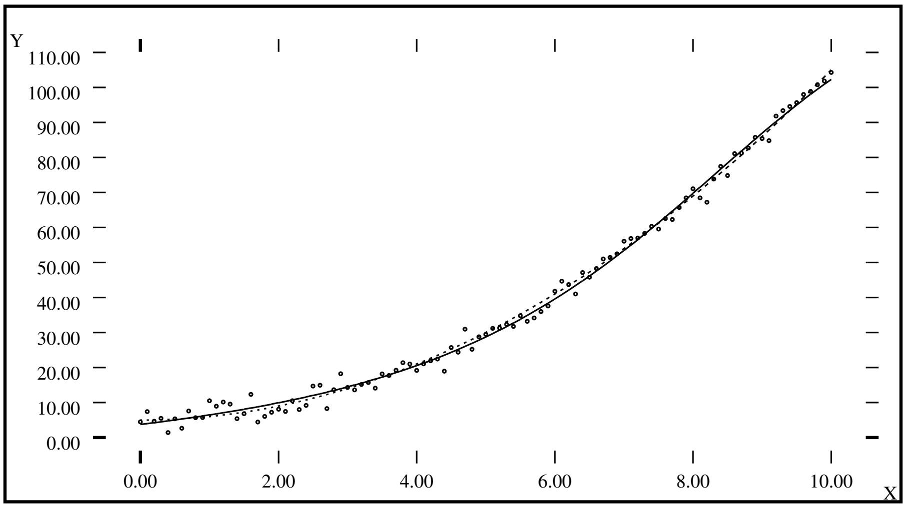


Figure 3.2: Fitting a neural net estimator to a data-stream of 100 points. The data was generated by adding noise with a variance of 4 to the curve $y = x^2 + 5$ (shown as a dashed line). The estimated function is shown as a solid line. There are two hidden nodes.

where the presence of n_h in the denominator accounts for reducing the degrees of freedom by one for each hidden unit since each hidden unit adds a constraint to the optimization problem.

Since these type of regression methods are strongly susceptible to scaling problems, we prescale the data so that each independent variable falls in the range $[0, 1]$ and the dependent variable falls within the range $[0, 0.8]$.[4] For monotonically decreasing functions, we perform a further transformation that maps each y value to $0.8 - y$ thus making it a monotonically increasing function.

Figure 3.2 shows the result of using the above method to estimate a fit to a data-stream derived from a quadratic function with noise from a normal distribution with $\sigma^2 = 4$. Note that the estimate is in fact monotonic and does not follow the noise.

3.3 Computing the Envelope

The estimate computed in the previous section is affected by the sample dataset. Since this dataset is of finite size, the estimate generated from it will not be precisely correct. In this section, we describe our method for bounding our estimate with an envelope that captures the uncertainty introduced by using a finite set of data.

[4]The upper value is less than 1.0 so that the output sigmoid does not saturate.

The typical confidence estimate used in regression analysis is the confidence interval, which is defined at a point $\mathbf{x}$ as $P(f(\mathbf{x})-b_{\mathbf{x}} \leq g(\mathbf{x}) \leq f(\mathbf{x})+b_{\mathbf{x}}) = 1-\alpha$ where $b_{\mathbf{x}}$ depends on $\mathbf{x}$. The confidence interval is a point probability measure since it expresses the uncertainty of the estimated value $f(\mathbf{x})$ at a single point $\mathbf{x}$ in the domain. Since we wish our envelope to bound all possible functions, we require that our confidence interval holds simultaneously at all points in the domain. This measure (called a *confidence band*) can be easily computed for linear regression models. Since our model is nonlinear, we use a linearization of f to form an approximate confidence band.

Assume that we have a linear model $f(\mathbf{x};\boldsymbol{\beta}) = \mathbf{x}^{\mathbf{T}}\boldsymbol{\beta}$ where $\boldsymbol{\beta}$ is a parameter vector of length p and that $\widehat{\boldsymbol{\beta}}$ is our estimate of the parameters. If we represent the data-stream as a matrix $[\mathbf{Y} \mid \mathbf{X}]$ where the ith row represents a single sample data-point $(y_i, \mathbf{x}_i)$, it can be shown [Bates and Watts, 1988] that the $1-\alpha$ confidence band for f is

$$\mathbf{x}^{\mathbf{T}}\widehat{\boldsymbol{\beta}} \pm s\sqrt{pF(\alpha;p,n-p)}\|\mathbf{x}^{\mathbf{T}}\mathbf{R}^{-1}\|$$

where s^2 is the sample standard error $\frac{\|\mathbf{Y}-\mathbf{X}\widehat{\boldsymbol{\beta}}\|^2}{n-p}$, F is the α quantile of the F-statistic with p and $n-p$ degrees of freedom, and $\mathbf{R}$ is the square portion of the QR decomposition of the array of sample inputs $\mathbf{X}$ ($\mathbf{X} = \mathbf{Q}\begin{bmatrix}\mathbf{R}\\ 0\end{bmatrix}$). Geometrically, the columns of $\mathbf{X}$ form a p-dimensional linear subspace called the *expectation surface* in which the solution must lie. The least squares computation finds the point on the expectation surface that is closest to $\mathbf{Y}$.

To use this result, we must linearize the nonlinear problem as follows. Note that the entry x_{np} of $\mathbf{X}$ is $\frac{\partial \mathbf{x}_n^{\mathbf{T}}}{\partial \beta_p}$. Using this, we linearize our estimator by defining a matrix $\mathbf{V}$ such that $v_{np} = \frac{\partial f(\mathbf{x}_n;\hat{\mathbf{w}})}{\partial w_p}$ and a vector $\mathbf{v}$ such that $v_p = \frac{\partial f(\mathbf{x};\hat{\mathbf{w}})}{\partial w_p}$. The envelope is then defined by

$$f(\mathbf{x};\hat{\mathbf{w}}) \pm s\sqrt{pF(\alpha;p,n-p)}\|\mathbf{v}^{\mathbf{T}}\mathbf{R}_{\mathbf{v}}{}^{-1}\|$$

where $\mathbf{V} = \mathbf{Q}_{\mathbf{v}}\begin{bmatrix}\mathbf{R}_{\mathbf{v}}\\ 0\end{bmatrix}$. The linearization can be viewed as defining a plane which is tangent to the true expectation surface (which is not a linear subspace) at $\hat{\mathbf{w}}$. Assuming that $\hat{\mathbf{w}}$ is close to the exact value for $\mathbf{w}$, the linearization will hold near this point on the plane.

Because the linearization is only an approximation, this estimate does not provide an exact $1-\alpha$ confidence band for f. However, the result is approximately correct and depends on the degree of nonlinearity of the expectation surface [Bates and Watts, 1988].

We use the LINPACK subroutine `DQRDC` to compute the QR decomposition of $\mathbf{V}$. This method performs the decomposition using pivoting so that $\mathbf{R}_{\mathbf{v}}$ is both upper triangular and has diagonal elements whose magnitude decreases going down the diagonal. With this form, we can easily recognize the case where the linearized matrix $\mathbf{V}$ may not have full

Figure 3.3: Reducing the size of a confidence band using monotonicity information. If we know that the underlying function is monotonically increasing, we can rule out the shaded areas of the figure as part of the envelope.

rank (i.e., only $q < p$ columns are linearly independent). This in turn means that $\mathbf{R_v}$ has zeros in all diagonals past column q. In such cases, the product $\mathbf{v^T}\mathbf{R_v}^{-1}$ may not have a solution. To handle this problem, we simply reduce the size of $\mathbf{R_v}$ by using only the upper left $q \times q$ corner. We must then also reduce the size of $\mathbf{v}$ so that it does not contain partials with respect to w_i where $i > q$. This is justified, since if $\mathbf{V}$ forms a p-dimensional basis for the linearized expectation surface, and its rank is only q, then the w_i where $i > q$ may be ignored since their value is arbitrary.

The confidence band is designed to cover all possible curves that fit the data with a probability of $1 - \alpha$. With traditional regression, this confidence band is the most precise description we can achieve given the general form of f. Using monotonicity information, however, we can further refine our prediction to derive a tighter envelope by ruling out portions of the curves that could not be monotonic. Consider the confidence band shown in Figure 3.3. If we know that the underlying function is monotonic we may remove the shaded regions from the prediction since no monotonic function within the confidence band could pass through these regions. Note that this final step could not be performed if we used point confidence intervals.

3.4 Examples

This section consists of several simple examples of function fitting together with two more complex examples – a real-world application of using MSQUID to estimate stream-flow given level and another real application of using MSQUID in the context of learning SQDEs from data.

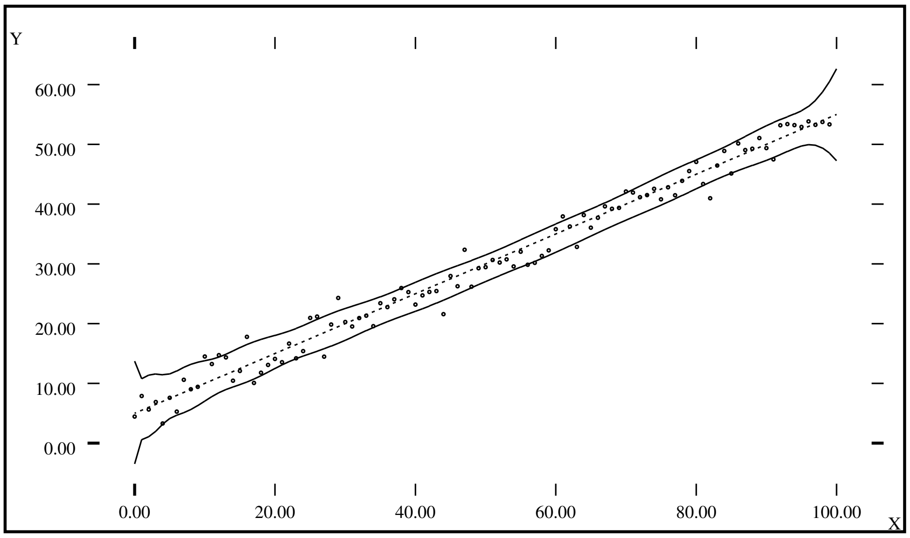


Figure 3.4: 95% envelope (solid lines) for the linear function $y = 0.5x + 5$ (dashed line). The estimator has 3 hidden nodes. Sample variance is 4.520.

3.4.1 Simple Examples

We have applied our envelope method to noisy datasets whose underlying functions are linear, quadratic, square root, exponential, and have sign changes in the second derivative. In order to give a feeling for the form of the envelopes generated, we present several of these test cases. A complete summary is provided in the next section. Each result is for a univariate monotonically increasing model function $f(x)$. Figure 3.4 shows the estimate and envelope for a set of 100 samples drawn from the function $y = 0.5x + 5$ with additive noise with variance 4. Note that the envelope expands at the ends since there is less data there to constrain the estimate.

The second example is that of a quadratic function with noise of variance 4. This dataset demonstrates the effect of nonuniform sampling. In areas where there is little data, the envelope bulges outward to compensate. Note also that the upper portion of the envelope is narrowest at the far end of the plot. This is due to bias in the estimation function which assumes that the curve will "heel over" just past the end of the dataset. For this reason, the computed envelope is valid only over the range of the given data.

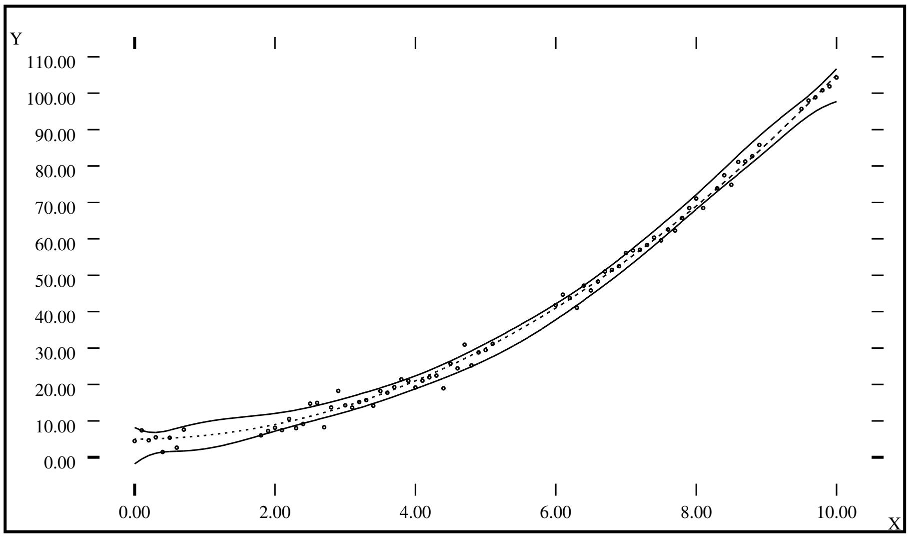


Figure 3.5: 95% envelope (solid lines) for the quadratic function $y = x^2 + 5$ (dashed line). The estimator has two hidden nodes. Sample variance is 4.227.

3.4.2 Application: Determining Stream-flow versus Level Curves

The control of reservoir levels is of great importance to water management authorities. Part of this task is determining the amount of inflow into a watershed from the different streams that feed it. These measurements are particularly important during flood conditions so that potential reservoir overflow can be predicted (and prevented) well in advance of its actual occurrence.

Unfortunately, measuring the flow of a stream is a manual and time-consuming task involving the lowering of a flow gauge into the stream at various points on a cross-section of the stream. On the other hand, the measurement of level is quite easy, and is often automatically sampled by data acquisition equipment that immediately relays the information back to a central office. Thus, an experimentally determined curve that relates level to flow would provide a means for quickly ascertaining stream-flow given only level.

One difficulty with producing such a curve is that finding a parametric model for a particular stream can be difficult since the shape of the stream-bed cross-section (which depends on effects such as silting, vegetation, and physical obstructions) must be accounted for. Typical solutions to this problem are (1) to use a hand-drawn curve (which is time-consuming and requires expertise) or (2) to use a logarithmic approximation under the belief that "everything looks like a line on log-log paper" (which often results in too simplistic a model).

While it is difficult to find a good parametric model, the knowledge that flow is a monotonic function of level makes constructing the stream-flow versus level curve an ideal application for MSQUID. To test MSQUID on this task, we collected stream-flow versus level datasets compiled by the Lower Colorado River Authority (LCRA) for three streams in the Austin area and fit them using MSQUID with three hidden units. Figure 3.6 shows the results of fitting the data for the Bay City gaging station. The standard error is 907, and an analysis of the residuals shows that 95% of the data falls within two standard errors of the estimate, indicating a reasonable fit.

On more careful examination, however, we see that the fit could be improved especially at levels below 10 (where the curve passes above much of the data) and near $level = 20$ (where the curve passes below the data). The reason why MSQUID has produced a curve with a poor fit in these regions is because the data is not sampled uniformly in the dependent variable. Instead, the data is clustered about particular values of level. Clustering can lead to a bad fit because local properties of the data may be swamped by the effect of far-away clusters. Clustering is particularly common in stream-flow data since most measurements are made at "normal" water levels whereas it takes floods and droughts to produce data for high and low levels.

One way to overcome this fitting error is to treat each cluster as a separate fitting problem and to run MSQUID on each region separately. One can then interpolate between the clusters using a function that maintains monotonicity[5]. Figure 3.7 demonstrates the resulting fit using such a method. The Bay City measurements can be split into three clusters representing the large mass of points in $[4, 15]$ together with two "arms" which contain points to the left and right of the main cluster. Fitting each of these clusters separately[6] yields a standard deviation of 690 resulting in a 25% better fit.

3.4.3 Application: SQDE Learning

In this example, the envelope method is used to create a semiquantitative model which is then simulated using QSIM to predict the behavior of a real physical system. To develop this example, we took a small plastic tank and drilled a hole in its bottom. Our goal was to construct a semiquantitative model that could be used to predict the level of fluid in the tank over time. The standard approach to constructing such a model in QSIM is to construct a qualitative model and then augment it with numerical information about the monotonic functions and parameters. Normally, these numerical envelopes are *ad hoc* in that they are hand-derived by the modeler. By applying the envelope method to a data

[5]This approach is in essence what someone using a French Curve would do – fit each cluster and then interpolate smoothly between them.

[6]To ensure a minimum number of points in a cluster, we "borrow" points from the main cluster as needed. This also improves the interpolation between clusters.

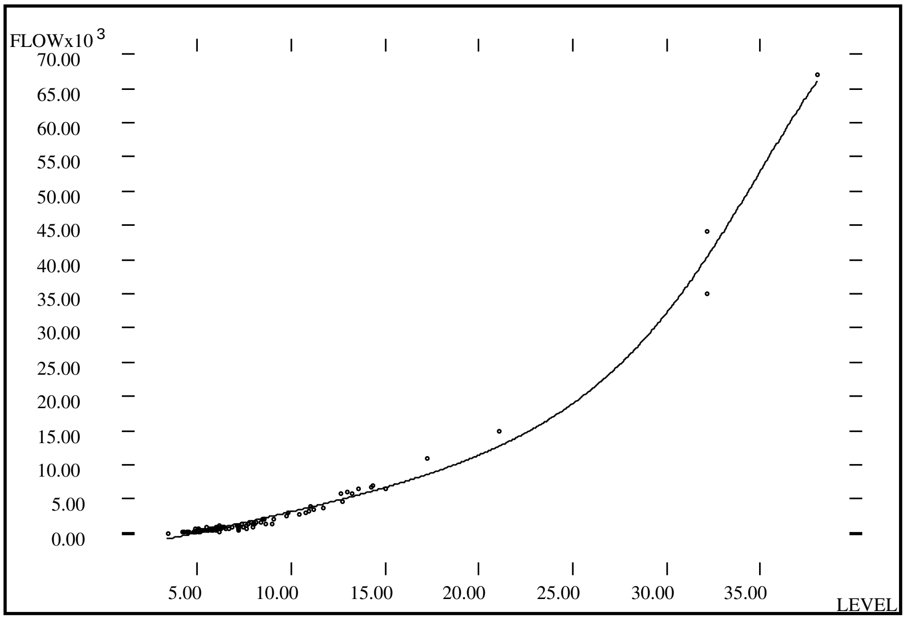


Figure 3.6: Using MSQUID to fit the flow vs level data for the Bay City gaging station. The standard error is 907. While the fit is reasonably good, it appears to miss the center of the main cluster between levels 4 and 15 and falls below the points near $level = 20$.

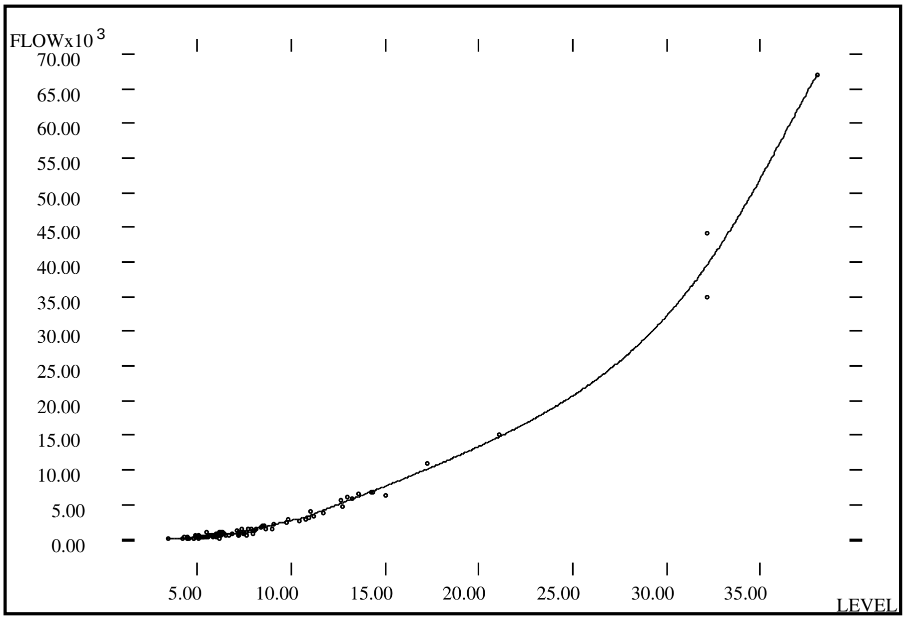


Figure 3.7: Using a clustering version of MSQUID to fit the same data as in Figure 3.6. By using different MSQUID functions for the central cluster and its arms, the fit is improved. The standard error in this case is 690.

stream from the system, we can construct these envelopes in a more principled way.

We require that the model prediction bound all real behaviors of the system (i.e., that the true behavior is within the predicted bounds) with probability $1 - \alpha$. Semiquantitative models are particularly good at this because they produce a bounded solution. We begin by asserting the qualitative part of the model – $A' = -f(A)$, $f \in M^+$ which asserts that the derivative of amount (A) is a monotonic function of amount. Since we are interested in level rather than amount, we also assert that $g(L) = A$, $g \in M^+$, i.e., that level (L) and amount are monotonically related. The complete qualitative model is thus

$$A' = -f(g(L)) \qquad f, g \in M^+$$

This qualitative description holds for a wide class of tanks. While it is unquestionably true for our tank, it is not specific enough to produce useful predictions. To increase the precision of the prediction, we must further define $f(A)$ and $g(L)$. To determine these functions, we performed the following experiment. We filled the tank with water and allowed it to drain while measuring flow (cc/sec), level (cm), and amount (cc). Figures 3.8 and 3.9 show the experimental data together with the 95% confidence envelopes. With this information, we used the QSIM semiquantitative simulation techniques Q2 and NSIM to produce a prediction. Since QSIM already reasons with monotonic envelopes, there are no special techniques required to handle the composed function $f(g(L))$.

Figure 3.10 shows the results of the simulation together with experimental data for time vs level. Since the width of the monotonic function envelopes are a function of the data-stream length, by observing the physical system for a longer time period, we would expect the envelopes to tighten further, thus yielding tighter predictions.

3.5 Analysis

The previous sections of this chapter have demonstrated the capabilities of MSQUID on a variety of function-learning tasks. This section examines the performance of MSQUID. We will see that MSQUID outperforms unconstrained optimization with less cost while robustly handling noise. Guided by this observation, we will suggest certain enhancements that will enable us to reliably use MSQUID in the context of semiquantitative identification.

3.5.1 Quality of the Estimation Function

To determine the quality of the estimate, we ran both MSQUID and UNCMND [Kahaner *et al.*, 1989], a fast, readily available, unconstrained optimizer, using the same neural network structure on the following series of test functions:

- linear ($y = 0.5x + 5$) on $x \in [0, 100]$

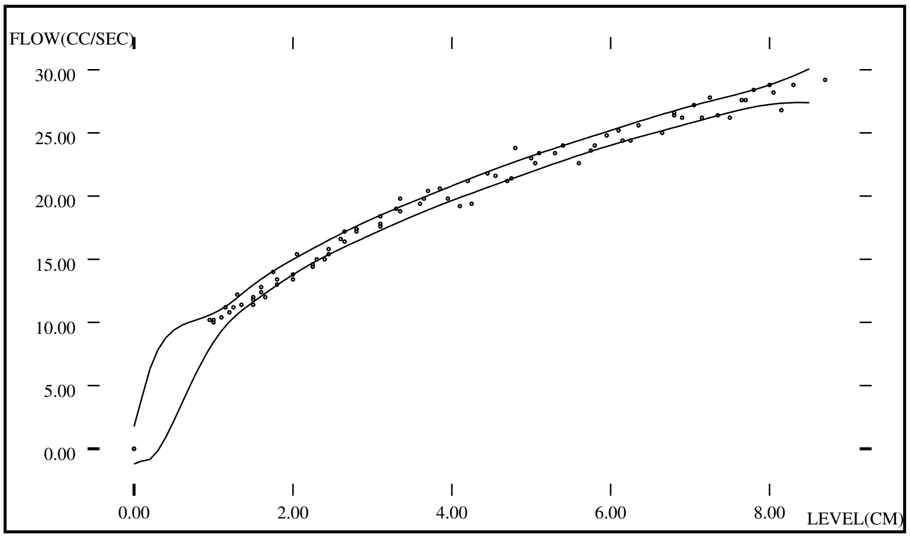


Figure 3.8: 95% envelope for level vs flow data from a tank. The estimator has two hidden nodes. Sample variance is 0.441.

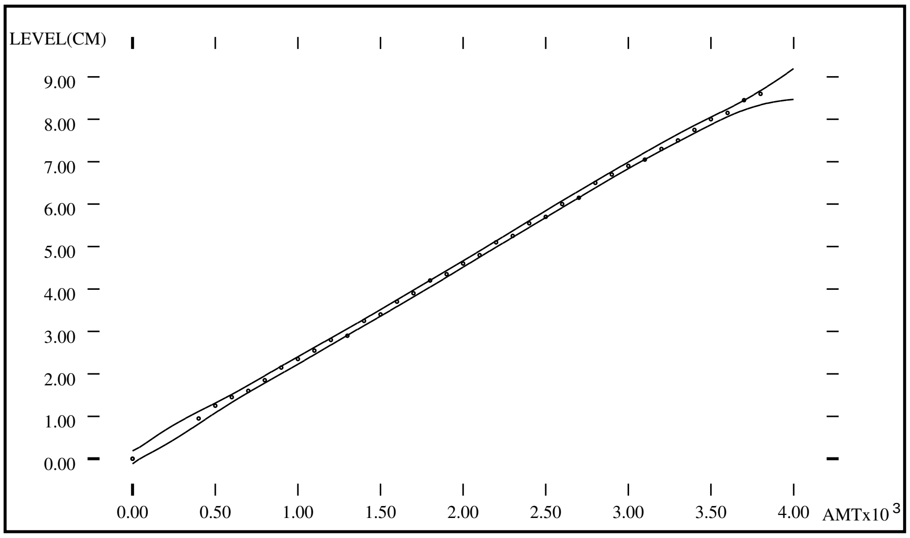


Figure 3.9: 95% envelope for the amount vs level data. The measurement accuracy for this data was greater than for the level vs flow data, hence the estimate is narrower. The estimator has two hidden nodes. Sample variance is 0.003.

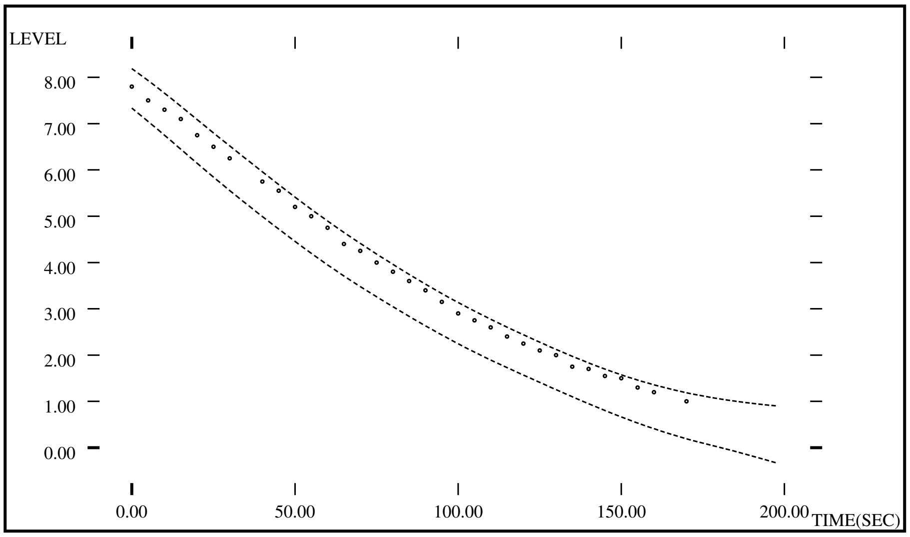


Figure 3.10: Prediction envelope for the tank model using the previous monotonic envelopes for f and g. Note that sample measurements taken from the system are all contained within the envelope. The predictions of this model could be strengthened with increased measurements which would serve to reduce the monotonic envelopes.

- short linear ($y = 0.5x + 5$) on $x \in [0, 10]$
- square root ($y = \sqrt{x}$) on $x \in [0, 2500]$
- cubic ($y = (x - 1.8)^3$) on $x \in [0, 3.68]$
- exponential ($y = e^x$) on $x \in [0, 3.9]$
- composed with sign change in second derivative ($y = 25(\sigma(x - 10) + \sigma(x - 25)))$ on $x \in [0, 40]$

These x ranges were selected to give comparable signal-to-noise ratios for each model (other than the short linear which is included to examine smaller signal-to-noise ratios). Test runs were generated by:

- selecting a test function,
- creating a sampling function that adds Gaussian noise of standard deviation 1, 5, 10, or 20 to the test function (which give signal-to-noise ratios of 50, 10, 5, and 2.5 for each function[7]),
- generating a fitting dataset by selecting a sample of 5, 10, 15, 20, 25, 50, 70, or 100 points uniformly sampled across the domain of the sampling function[8],
- selecting a value for n_h (either a fixed value of 3 or 8, or allowing MSQUID to select the best value),
- selecting an optimizer (MSQUID or unconstrained),
- fitting the data five times (to minimize the problem of falling into a local minimum in the parameter space),
- computing the average standard deviation of the fit function to an independently generated sampling of 1000 points from the same test function + noise.[9]

The effect of noise and sampling Figure 3.11 illustrates the effects of noise level and sampling frequency on the linear model. Not surprisingly, we found that for a fixed sample size, an increase in noise leads to an increase in error across all models. This result is consistent with the intuition that noise has a positive effect on uncertainty. Similarly, we found that error rate decreases with respect to increasing sample size across all models.

[7] Or 5, 1, 0.5, 0.25, and 0.05 for the short linear model. Signal-to-noise ratio was computed by dividing the maximum y range over the test interval by the noise standard deviation.

[8] These sizes were selected to examine behavior of MSQUID in both under-constrained and over-constrained cases for both 3 and 8 units.

[9] To further reduce experimental variance, we also eliminated the best and worst fits before computing the average.

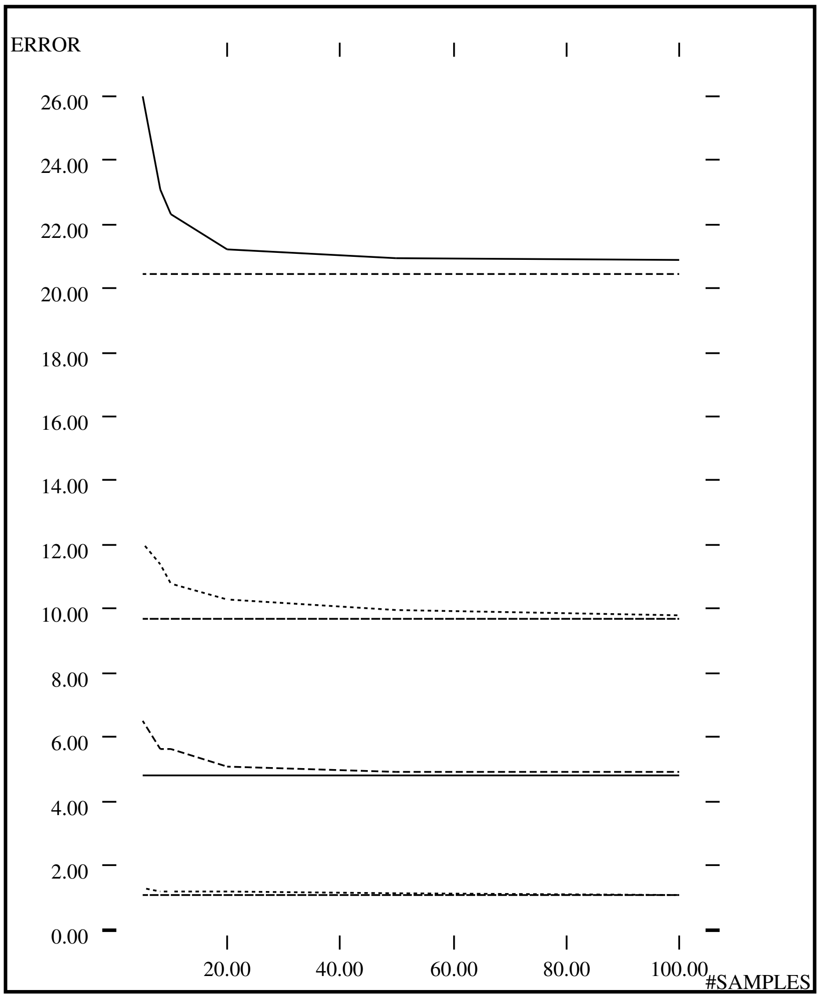


Figure 3.11: The effect of varying the noise and sample size on MSQUID fits. Each plot shows error as a function of the number of samples for noise with standard deviation 1, 5, 10, and 20 for the linear model (the baseline standard deviations for each curve are also plotted). Decreasing noise and increasing sample size decreases the fit error.

The effect of the number of hidden units and model complexity For all but the composed model, the number of units in the hidden layer had little effect on the error when comparing $n_h = 3$ with $n_h = 8$ (see Table 3.1). For the composed model, however, the effect of a small n_h resulted in significantly worse predictions for larger sample sizes, especially at low noise levels. This is presumably because 3 hidden units are not enough to model the sign changes in the second derivative of the composed function[10]. When allowed to compute n_h on its own, MSQUID normally selected a value between 1 and 3, except for the composed case where it selected $n_h = 4$.

The close agreement between the fits at $n_h = 3$ and $n_h = 8$ at low sample sizes seems somewhat surprising. Normally, one would expect that for samples of less than 25 points, eight hidden units (which translates to 25 weights) would over-fit the data and lead to a worse error than would fitting with 3 hidden units (which translates to 10 weights). It appears, however, that the monotonicity constraints on the weights (see Equation 3.1) narrow the search space sufficiently to reduce over-fitting even in the under-constrained cases.

Table 3.2 shows the CPU time required to compute fits with $n_h = 3$. Times range from slightly less than two seconds up to almost fourteen seconds and, in general, increase with the number of samples but are constant with respect to noise. This result is reasonable since the time complexity of the objective function[11] is linear in the number of samples.

Table 3.3 demonstrates that the cost of MSQUID increases with the number of hidden units with $n_h = 8$ being between roughly two and thirteen times as slow as $n_h = 3$. It is instructive to understand the reasons for this decrease in speed. MSQUID uses OPT [Biegler and Cuthrell, 1985] to solve the constrained optimization problem. OPT breaks the optimization problem into the tasks of computing a search direction in the parameter space defined by $\mathbf{w}$ and then selecting a distance along this direction that reduces the objective function. This pair of operations constitutes one *step* of the optimizer and requires repeated evaluation of the objective function. The cost of evaluating the objective function is linear in n_w, the number of weights in the model. Since $n_w = 3n_h + 1$, the computation is 2.5 times as expensive for 8 units as it is for 3 units. Furthermore, since the search space is of dimension n_w, the increased dimensionality when $n_h = 8$ makes each step of the search more difficult even disregarding the increased cost of the objective function.

Overall, this cost analysis suggests that while using a model that is too flexible will not impact the quality of the fit, it will have a cost impact. Thus, knowledge of the proper number of units required to model a given function can save compute time.

[10] This result indicates that using too few a number of hidden units to the model results in a bias towards an overly-smooth approximation.

[11] The *objective function* computes the sum-of-squared errors between each sample point and the estimate (see Equation 3.2).

Model	Noise	*error difference/s* per sample size							
	stddev	5	10	15	20	25	50	70	100
Linear	1	–	–	–	–	–	–	–	–
	5	-0.22	–	–	–	–	–	–	–
	10	–	–	–	–	–	–	–	–
	20	–	-0.12	–	–	–	–	–	–
Short Linear	1	–	–	–	–	–	–	–	–
	5	–	–	–	–	–	–	–	–
	10	–	–	–	–	–	–	–	–
	20	–	–	–	–	–	–	–	–
Square Root	1	–	–	–	–	–	–	–	–
	5	–	–	–	–	–	–	–	–
	10	–	–	-0.14	–	–	–	–	–
	20	–	–	–	–	–	–	–	–
Cubic	1	-0.19	-0.11	–	-0.21	–	–	–	–
	5	–	–	–	–	–	–	–	–
	10	–	–	–	–	–	–	–	–
	20	–	–	–	–	–	–	–	–
Exponential	1	–	–	–	–	–	–	–	0.11
	5	–	–	–	–	–	–	–	–
	10	–	–	–	–	–	–	–	–
	20	–	–	–	–	–	–	–	–
Composed	1	–	–	–	–	–	7.20	–	2.40
	5	–	–	–	–	0.23	0.50	–	0.31
	10	–	–	–	–	–	–	–	–
	20	–	–	–	–	–	–	–	–

Table 3.1: The effect of varying the number of hidden units. Each entry computes the difference between the error calculated for $n_h = 3$ and the error calculated for $n_h = 8$ normalized by noise standard deviation (s). Positive values indicate that $n_h = 3$ has larger error. All differences less than 0.10 have been eliminated from the table in order to highlight larger differences. The most significant differences occur in the composed model when $noise = 1$ or 5, which indicate that three hidden units are insufficiently flexible to model the function.

Model	Noise	Total user CPU time (seconds) per sample size							
	stddev	5	10	15	20	25	50	70	100
Linear	1	2.43	3.07	3.80	3.27	3.84	2.93	4.56	4.17
	5	2.75	3.50	3.49	3.50	1.76	3.20	2.07	3.05
	10	3.16	4.80	3.38	3.95	2.30	5.61	5.16	3.22
	20	4.57	3.86	3.15	4.05	5.28	6.54	9.32	5.30
Short Linear	1	2.86	2.29	2.94	2.50	3.81	4.08	4.60	3.59
	5	3.48	1.73	3.13	2.76	2.89	4.63	3.52	4.71
	10	2.34	2.74	4.42	2.77	1.98	3.54	3.57	2.28
	20	2.02	2.19	2.27	4.22	2.28	2.06	2.33	3.02
Square Root	1	3.82	2.11	4.75	4.54	3.65	4.61	5.27	5.66
	5	3.37	4.02	1.80	1.61	5.35	4.90	4.04	5.12
	10	3.57	3.78	3.12	4.96	4.88	2.11	3.97	13.98
	20	3.26	2.72	2.76	5.40	3.26	4.76	4.10	9.29
Cubic	1	5.49	4.94	4.57	5.76	4.00	8.01	13.15	13.95
	5	4.36	8.13	4.06	2.89	2.20	4.25	9.03	8.56
	10	3.29	3.39	4.00	4.44	2.62	5.09	3.38	6.65
	20	2.82	4.35	3.04	6.20	4.69	4.73	5.99	4.16
Exponential	1	4.01	3.01	3.59	3.08	4.52	3.70	10.46	9.53
	5	3.33	3.59	3.10	4.81	4.13	4.41	4.80	11.53
	10	2.95	5.86	4.28	5.72	6.15	4.96	7.75	3.03
	20	2.41	2.21	2.92	5.06	4.06	4.00	7.43	6.96
Composed	1	4.91	3.88	3.29	3.56	4.66	3.60	4.59	6.62
	5	3.86	4.04	3.34	3.83	3.06	4.08	4.55	7.50
	10	4.79	4.62	4.22	3.84	4.08	4.20	5.77	5.66
	20	2.22	4.65	4.74	4.00	6.94	3.76	4.10	5.48

Table 3.2: Total CPU time in seconds for $n_h = 3$ (as computed by the first argument of the Allegro Common Lisp function `EXCL::get-internal-runtimes`) on a SPARC Ultra 1. In general, CPU time increases with the number of samples, but is constant with respect to signal noise.

Model	Noise	Time ratio $(n_h = 8)/(n_h = 3)$ per sample size							
	stddev	5	10	15	20	25	50	70	100
Linear	1	5.44	5.94	5.38	5.03	4.10	5.28	3.37	5.47
	5	2.53	7.81	8.35	4.15	7.58	6.06	3.49	5.38
	10	5.22	3.06	5.61	3.33	1.63	8.19	2.31	3.68
	20	3.62	4.00	4.18	6.25	6.81	5.65	9.03	8.29
Short Linear	1	4.64	7.04	6.58	4.93	6.34	3.38	0.76	1.50
	5	6.20	4.39	5.46	3.22	7.21	1.69	3.12	7.22
	10	1.89	1.55	8.34	2.60	1.78	2.76	6.22	5.07
	20	1.24	1.56	1.76	3.87	1.78	2.86	1.39	2.18
Square Root	1	3.80	9.42	4.63	4.93	5.89	4.42	3.85	4.13
	5	3.61	3.49	3.31	3.13	6.03	3.19	1.95	8.45
	10	6.93	6.32	2.86	4.09	7.16	2.61	6.05	2.99
	20	2.93	4.20	1.23	0.66	10.26	10.51	12.73	3.77
Cubic	1	2.96	0.67	4.56	2.21	10.75	5.37	2.18	3.44
	5	3.89	4.67	5.69	8.10	1.15	5.74	3.15	1.05
	10	1.17	4.95	4.28	3.95	2.07	4.87	1.38	4.43
	20	1.03	5.59	6.30	5.89	5.74	1.78	5.05	1.57
Exponential	1	2.50	5.34	4.15	3.56	2.06	6.07	2.24	3.26
	5	4.08	2.52	5.43	3.04	2.46	3.57	2.12	4.21
	10	3.60	2.93	3.56	3.27	4.30	7.68	4.39	9.68
	20	4.80	3.14	2.67	3.86	9.57	3.21	6.19	5.21
Composed	1	3.80	4.17	6.09	5.59	2.72	2.81	3.77	4.25
	5	3.84	2.94	7.39	4.75	2.22	8.39	5.48	1.78
	10	4.64	2.34	3.40	7.60	4.03	4.53	2.71	4.07
	20	5.45	5.86	5.49	8.06	4.09	3.89	1.50	0.88

Table 3.3: Ratio of CPU times for $n_h = 8$ versus $n_h = 3$. The cost of using eight hidden units is between roughly 2 and 13 times as much as for eight units.

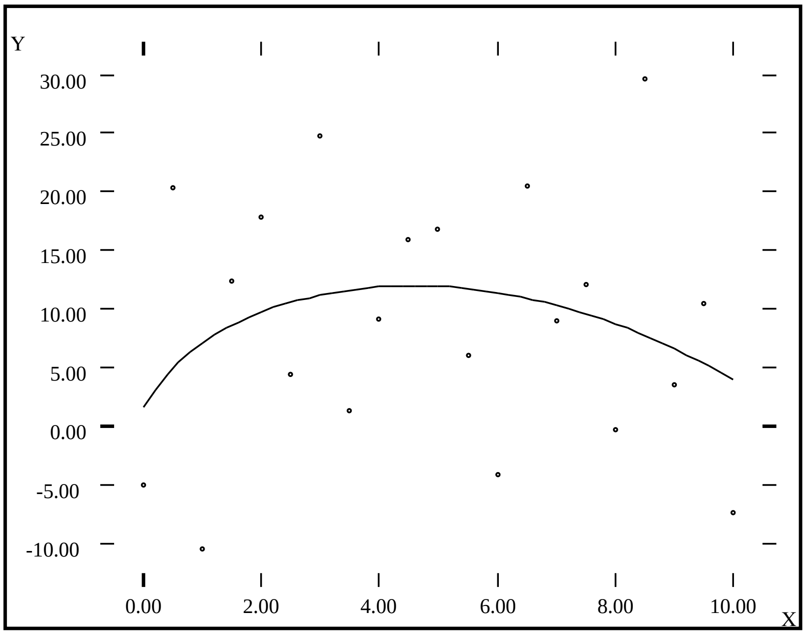


Figure 3.12: An example of over-fitting with an unconstrained optimizer. The data is from the short linear model with noise standard deviation of 10. The model uses 3 hidden units. Because of the relatively small signal-to-noise ratio, the unconstrained optimizer produces a non-monotonic fit.

The value of constrained optimization Table 3.4 summarizes the results comparing MSQUID to unconstrained optimization for $n_h = 3$. For small sample sizes, the unconstrained optimizer produces a less accurate prediction that does MSQUID. Since the unconstrained optimizer cannot utilize knowledge of monotonicity, it may even select non-monotonic functions, which lead to greater error (see Figure 3.10). By using this knowledge, MSQUID is able to reduce the degree of over-fitting.

Table 3.5 summarizes the difference in cost of the two methods in terms of wall-clock time[14]. MSQUID is significantly slower for small sample sizes, but as sample size increases its speed is on a par with the unconstrained optimizer. Part of the dependency on sample size is due to the fact that the UNCMND optimizer uses finite differencing to compute gradients in the search space (which requires more calculations of the objective function)

[13]Errors for the composed model at low noise levels are unreliable since the estimator with $n_h = 3$ is not flexible enough to properly model the function.

[14]We use wall-clock (elapsed) time because the unconstrained optimizer is called as a subprocess by Lisp and the Allegro CPU timer is unable to time subprocesses.

Model	Noise stddev	$(error(MSQUID) - error(unconstrained))/s$							
		5	10	15	20	25	50	70	100
Linear	1	–	–	–	–	–	–	–	–
	5	-0.42	–	–	–	–	–	–	–
	10	–	–	–	–	–	–	–	–
	20	-0.61	–	-0.48	–	–	–	–	–
Short Linear	1	-0.23	–	–	–	–	–	–	–
	5	-0.43	-0.19	-0.14	–	-0.14	–	–	–
	10	-0.28	-0.20	-0.11	–	–	–	–	–
	20	-0.38	–	-0.67	-0.47	–	–	–	–
Square Root	1	0.72	–	–	–	–	–	–	–
	5	–	–	–	–	–	–	–	–
	10	–	–	–	–	–	–	–	–
	20	–	–	-0.45	–	–	–	–	–
Cubic	1	0.15	–	–	-0.20	–	-0.26	-0.16	–
	5	-0.27	-0.13	–	–	–	–	–	–
	10	-0.10	-0.73	-0.42	–	–	–	–	–
	20	-0.14	-0.26	-0.22	-0.64	–	–	–	–
Exponential	1	–	-0.15	–	–	–	–	–	–
	5	–	–	–	–	–	–	–	–
	10	–	-1.63	–	–	–	–	–	–
	20	-0.17	–	–	–	–	–	–	–
Composed	1	-4.32	–	0.22	–	0.37	0.15	0.24	-2.40
	5	–	–	–	–	–	0.24	–	–
	10	-0.38	–	–	–	–	–	–	–
	20	–	–	–	–	–	–	–	–

Table 3.4: Normalized error difference between MSQUID and unconstrained optimization using $n_h = 3$. Positive values indicate that MSQUID has larger error. In general, MSQUID has lower error, especially for cases where there the sample size is small. As sample size becomes large, the error difference becomes negligible[13].

while OPT does not[15].

These results show that while unconstrained optimization and MSQUID may be approximately as accurate at large sample sizes, unconstrained optimization is no more efficient (at least when using UNCMND). And, while unconstrained optimization is more efficient than MSQUID at small sample sizes, it is much less accurate[16].

3.5.2 Quality of the Confidence Band

The quality of the confidence bound for a dataset generated by corrupting a function $y(x)$ with noise can be measured by determining how much of $y(x)$ lies outside the bounds computed by MSQUID for the dataset. We use the function

$$e(y, \underline{y}, \overline{y}, D, h) = \sum_{i=0}^{(\overline{d}-\underline{d})/h} \left[g(y(\underline{d}+ih) - \overline{y}(\underline{d}+ih)) + g(\underline{y}(\underline{d}+ih) - y(\underline{d}+ih)) \right] \tag{3.3}$$

to measure the quality of the confidence band over the interval $D = [\underline{d}, \overline{d}]$ where $[\underline{y}(x), \overline{y}(x)]$ is the bound computed by MSQUID at x, h is the step-size between comparison points, and $g(x)$ returns x if $x > 0$ and zero otherwise. This function approximates the integral of the error defined by the portion of the true function that lies outside the envelope. Table 3.6 shows the value of e at several different sample sizes. As can be seen, the bound captures the true function in all cases except for the composed function at low noise. This is because the fit at $n_h = 3$ is not flexible enough to generate a good estimate which leads to the incorrectness of the envelopes. Increasing n_h to 6 improves the error as shown in the last entry of the table.

3.5.3 Recommendations

As the above analysis suggests, MSQUID performs reasonably well on estimating and bounding a monotonic function corrupted with noise. There are, however, conditions under which its performance is degraded by using too inflexible a model or by becoming trapped by local minima in the parameter space. This section suggests strategies that maximize the effectiveness of MSQUID for fitting monotonic functions. These results will be used in the next chapter where MSQUID is used as a component in a larger system.

[15] In fact, our studies indicate that UNCMND uses between roughly 10 and 100 times as many function evaluations as does OPT.

[16] While a faster unconstrained optimizer (for example, UNCMND augmented to use an analytic gradient calculation) might be found, this would not improve the accuracy problems inherent in *any* unconstrained optimizer at small sample sizes. Thus, in this case, the most efficient strategy would be to use OPT for small sample sizes (where it is more accurate) and the faster optimizer for large sample sizes (where it is more efficient).

Model	Noise	Time ratio MSQUID/unconstrained per sample size							
	stddev	5	10	15	20	25	50	70	100
Linear	1	4.48	1.44	2.21	1.38	1.59	0.60	1.05	0.40
	5	2.44	2.42	1.91	1.56	0.81	0.65	0.50	0.60
	10	1.91	2.90	1.63	1.51	0.87	1.31	0.96	0.50
	20	3.99	2.47	2.06	1.30	1.41	1.78	1.96	1.22
Short Linear	1	1.96	1.33	2.26	1.30	1.91	1.29	0.87	0.74
	5	3.40	1.10	2.38	1.58	1.30	1.36	1.15	0.76
	10	1.66	1.50	3.07	1.86	0.82	0.94	0.72	0.33
	20	1.70	1.25	1.30	2.12	0.93	0.66	0.59	0.41
Square Root	1	2.90	0.93	2.13	2.45	1.36	1.49	0.85	0.61
	5	4.16	1.92	0.74	0.59	2.94	1.86	1.03	1.88
	10	2.52	1.69	1.89	1.95	0.92	1.89	0.40	2.05
	20	7.30	1.04	1.23	1.56	1.58	0.88	0.63	1.39
Cubic	1	3.01	2.65	1.14	1.78	2.26	2.14	5.55	2.40
	5	2.33	4.67	2.56	1.27	1.22	1.05	2.86	1.63
	10	3.00	1.90	1.66	1.98	0.96	1.14	0.74	0.90
	20	1.33	4.24	1.81	2.31	3.94	1.46	0.66	2.48
Exponential	1	7.55	1.00	1.50	1.25	4.71	0.80	1.80	1.06
	5	2.06	3.93	2.02	1.66	2.00	1.74	0.99	1.95
	10	2.08	2.36	3.05	3.21	1.26	3.68	1.24	1.86
	20	0.90	1.50	1.45	2.43	1.32	0.75	1.17	0.80
Composed	1	3.44	2.09	1.67	1.70	2.31	0.64	1.09	0.86
	5	2.65	2.62	1.18	1.77	0.89	1.60	0.91	1.06
	10	3.64	2.69	1.95	1.31	2.02	1.53	1.06	0.95
	20	1.58	4.32	2.38	1.70	1.82	0.83	0.88	1.23

Table 3.5: Ratio of wall-clock times for MSQUID versus unconstrained optimization for $n_h = 3$. The cost of unconstrained optimization is much less for small sample sizes, but decreases to approximately the same cost as MSQUID at 100 samples.

Model	Noise	$e(y, \underline{y}, \overline{y}, D, h)$ per sample size					
	stddev	15	20	25	50	75	100
Linear	1	0	0	0	0	0	0
	5	0	0	0	0	0	0
	10	0	0	0	0	0	0
	20	0	0	0	0	0	0
Short Linear	1	0	0	0	0	0	0
	5	0	0	0	0	0	0
	10	0	0	0	0	0	0
	20	0	0	0	0	0	0
Square Root	1	0	0	0	0	0	0
	5	0	0	0	0	0	0
	10	0	0	0	0	0	0
	20	0	0	0	0	0	0
Cubic	1	0	0	0	0	0	0
	5	0	0	0	0	0	0
	10	0	0	0	0	0	0
	20	0	0	0	0	0	0
Exponential	1	0	0	0	0	0	0
	5	0	0	0	0	0	0
	10	0	0	0	0	0	0
	20	0	0	0	0	0	0
Composed	1	0	0	0	29.0	56.9	57.5
	5	0	0	0	6.0	11.7	49.6
	10	0	0	4.2	0	15.9	19.9
	20	0	0	0	0	0	0
Composed $n_h = 6$	1	-	-	-	0	3.8	0
	5	-	-	-	0	0.4	4.8
	10	-	-	-	0	0	0
	20	-	-	-	0	0	0

Table 3.6: Confidence bound error. Each entry represents the area between the outside of the MSQUID-computed 95% confidence bound and the true function that generated the data as computed in Equation 3.3 over the domain D of the individual test functions. The step-size h is such that 200 steps are taken across the domain. All models use $n_h = 3$ except for the last entry which uses $n_h = 6$. The true function is enclosed by the envelope in all but the composed model. Increasing n_h in this case leads to a better enclosure.

- Use a fixed number of hidden units.
  A large number of hidden units does not appreciably improve the estimate (except in cases where the function has widely varying sign of the second derivative) while it does produce significant cost. Thus selecting a "reasonable" number of units for a task will reduce the amount of computation. Based on our experiments, we choose $n_h = 3$ for functions that do not have sign changes in curvature.

- Verify that the data is indeed monotonic.
  Like all regression methods, MSQUID will find a monotonic fit even for data that is not monotonic. Thus, it is important to ensure that a dataset is monotonic before fitting. In the next chapter, we implement this check with a simple slope test.

- Use prior knowledge to guide the fitting process.
  If the variance of the noise is known, this information can be used to check the goodness of fit and thus avoid local minima in the model space.

3.6 Related Work

This work is related to several approaches to function estimation using neural networks. Cybenko [Cybenko, 1989] has shown that any continuous function can be represented with a neural net that is similar to one that we use. With our method, we assume a priori that the true function is monotonic. This assumption restricts our attention to a smaller class of functions which means that less data is needed to compute a reasonable estimate. Additionally, we compute a confidence measure on our estimate in the form of an envelope.

The VI-NET method [Leonard *et al.*, 1992] also uses a neural network for computing estimates of general functions. It is notable in that it provides a confidence measure on its estimate and can determine when it is being asked to extrapolate. By using radial basis functions instead of sigmoidal ones, it is able to handle different variances across the function. This is especially useful in applications where there is no a priori information about g. Because it allows non-constant variances, it would be difficult to get a VI-NET to return simultaneous confidence bands, which are important since we wish to bound all curves that could have generated the data-stream. In contrast, our approach can only handle fixed-variance problems, but since variance will often track either $\mathbf{x}$ or y, we can make monotonicity assumptions about σ^2 if it should prove necessary. Under such circumstances, variance stabilization techniques [Draper and Smith, 1981] should prove useful in fixing the variance.

This work is also related to monotonic function estimation [Joerding and Meador, 1991, Hellerstein, 1990, Kruskal, 1964], particularly the NIMF estimator [Hellerstein, 1990] which also determines envelopes for multivariate monotonic functions. NIMF allows for a more general noise model (zero mean, *symmetrically* distributed) and uses a non-parametric statistical method for determining point confidence bounds. These bounds, however, are much weaker than those derived with methods that first compute an estimate. This is not surprising, since we are assuming more about the noise than NIMF does. Finally, NIMF produces point bounds only, and it is unclear how these bounds could be made into confidence bands of reasonable width.

3.7 Discussion

This paper has described a method for computing envelopes for functions described solely by monotonicity information and a stream of data. It improves over existing methods for function estimation by providing a simultaneous confidence band that encloses all functions that could have generated the data-stream. Using monotonicity information provides several benefits to function estimation :

- Less data is required to obtain a reasonably precise model.
- Problems with over-fitting are reduced.
- When combined with simultaneous confidence bands, portions of the band can be eliminated, thus further improving model precision.
- While the class of monotonic functions may seem restrictive for modeling continuous systems, we can represent many non-monotonic functions as compositions of monotonic ones. For example, in a system of two cascaded tanks, the level of water in the lower tank is typically a non-monotonic function of time. The flow into the lower tank, however, is the composition of a monotonically increasing function of the amount of water in the upper tank and a monotonically decreasing function of the water in the lower tank. By using semiquantitative simulation methods, we can represent this composition and thus simulate systems with non-monotonic behaviors.

The method is applicable for cases where sample variance is fixed. Future work includes adding variance stabilization methods to handle cases where the data has a non-constant variance. We also plan to add the ability to impose further constraints such as stating that the function must pass through zero.

Our method for deriving bounds for monotonic functions plays a key role in the construction of semiquantitative models, especially for monitoring and diagnosis tasks where process data is readily available. Because the precision of the resulting envelopes is a function of the amount of data used to compute them, our bounding method also provides a systematic method for shifting the precision of a semiquantitative model along a continuum from purely qualitative to exact.

3.8 Nomenclature

$\mathbf{x}$	The domain of the function $(y = g(\mathbf{x}))$.
n	The dimension of $\mathbf{x}$.
n_h	The number of hidden units in the network.
p	The number of weights in the network (a total of $(n+2)n_h + 1$).
$w_{i[i,j]}$	The weight from x_i to hidden unit j.
$w_{o[j,1]}$	The weight from hidden unit j to the output.
$\mathbf{w}$	A vector of all weights.
$\hat{\mathbf{w}}$	The estimated weight vector computed by MSQUID.
$\hat{y}$	The estimate of y.

Chapter 4

Refining Imprecisely-defined ODE Systems

This chapter describes SQUID, a system identification method which uses semiquantitative inference to refine a model space defined by an SQDE. SQUID is robust in the face of uninformative data and can be faster than traditional search-based refinement methods.

4.1 Motivation

Constructing models of physical processes is difficult because such systems are often imprecisely understood. Yet, by virtue of being physical, it is often possible to observe their behavior under different circumstances. By interpreting these observations, we can reduce our uncertainty about the process and refine our model. We thus learn a better model of the system through experience.

Consider a problem that is common in Electrical Engineering classes: you are given a box with several inputs and outputs and are asked to determine what is inside the box. To solve this problem, one needs to construct a model of the device inside the box based solely on experimentation. A typical solution to this problem is to experiment with the system, varying the inputs and observing the outputs. By gaining some experience with the system, it is then possible to observe general properties of the system and thus hypothesize an imprecise model. Once a plausible model has been constructed, further experimentation and analysis can refine the model further. In the limit of infinite data (and infinite time!) a precise model of the system could be constructed.

More concretely, this model learning method can be broken into two components (see Figure 4.1):

1. Select a model space.

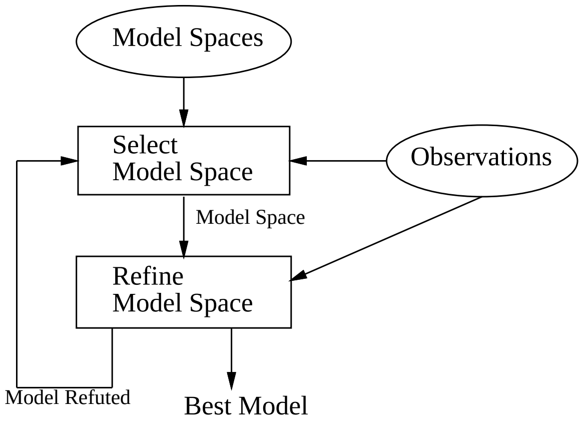


Figure 4.1: Model learning from data. Model learning can be broken into two phases, model space selection and model space refinement.

Starting with a prior set of model spaces (e.g., models of different order), select a single model space based on the observations.

2. Refine the model space.
   Given a model space, attempt to reduce it as much as possible so that it contains only model instances that could have plausibly generated the observations. If the model space is refined to zero size then refute the model and hypothesize a new model space.

Since selecting amongst model spaces is a very different problem for selecting a model from a model space, this decomposition permits independent solutions to these problems. Existing work in qualitative reasoning [Coiera, 1989, Richards *et al.*, 1992, Cem Say and Kuru, 1996] addresses the model selection problem. In this chapter, we concentrate on model refinement.

4.1.1 An Application: Process Monitoring

The monitoring of continuous processes can also be cast as a model learning problem (see Figure 4.2). Here, the goal is to select a model that best describes the process using a set of measurements. In this case, the set of model spaces is typically more structured than in the general case (since it probably contains specific information about faults) and the model space refinement (or *tracking*) step emphasizes matching data and model predictions rather than the reduction of the model space. In fact, traditional monitoring approaches typically use model spaces consisting of a single model which simplifies tracking since it is easy to derive predictions from them. However, there are problems with using such models:

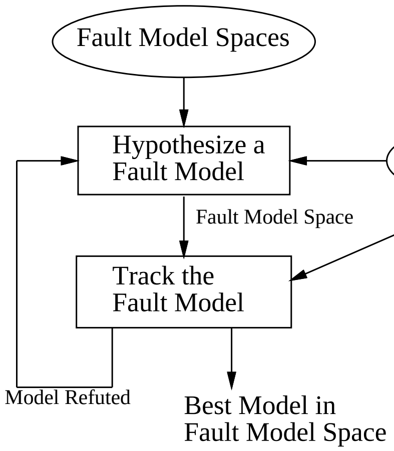


Figure 4.2: Monitoring can be viewed as a specialized form of function learning where fault model spaces are tracked, yielding a best fit.

- Precise models are unrealistic.
  Because physical systems contain components and functions whose values may vary from their design specifications, a precise model is at best an approximation of a set of models. If this space is large then the *model-process mismatch* problem may lead to erroneous behavioral predictions.

- Precise models are too specific.
  Often, a particular failure mode may encompass a wide variety of models. For example, a leak may be represented as a monotonic function of a pressure difference. Ideally, we would like to consider this as one model space, however using precise model representations would require many models, each describing a different parameterization of the monotonic function. This needlessly complicates the model space selection process.

- Using precise models complicates the tracking problem.
  Because of the model-process mismatch problem it may be difficult to conclude that a deviation is due to a fault or due to a bad model.

4.1.2 Traditional Refinement Methods – System Identification

Refining a model space given a set of measurements is the subject of *system identification* [Ljung, 1987], a field closely related to control theory. Traditional system identification uses six steps (see Figure 4.3):

1. An experiment is run to collect process data. This experiment should be informative, i.e., it should contain data that describes the system dynamics adequately. For example, data collected while a process is quiescent is not sufficiently informative for constructing a dynamic model.

2. A model space is selected. In traditional methods, model spaces are defined in terms of precise parametric models. Thus, the model space is defined by the parameter space of the model.

3. An identification criterion is selected. Typically, this is something like minimizing the squared error between elements of the model space and the data.

4. The model space is searched for the single model that best meets the criterion given the data. Typical methods use gradients in the parameter space to converge on this model[1].

5. The resulting model is validated. Validation is necessary in order to ensure that the choices made for performing the identification were acceptable. Validation typically involves comparing the prediction of the model with an independent set of data generated by experiment and comparing the error.

6. If validation is successful, the model is accepted. If it fails, we choose to collect more data (step 1), select a new parameterization (step 2), identification criterion (step 3), or search strategy (step 4) and repeat the loop.

4.1.3 Why System Identification is Insufficient for Monitoring Tasks

Given that system identification is a well-understood method for refining models, why is it not an adequate method for solving the monitoring problem? The reason is that the assumptions required for system identification are not always met in a monitoring framework. Traditional system identification relies on the following assumptions:

- Informative data can be collected across the operating region(s) of interest.
- An appropriate parametric model space can be found.
- An efficient search of the model space is possible.
- There is sufficient data to perform validation.

For monitoring problems, however, these assumptions cannot always be met:

[1]Although the search results in the selection of a *single* model instance, the final model space still contains a set of models since the parameters have confidence intervals due to the finite size of the dataset.

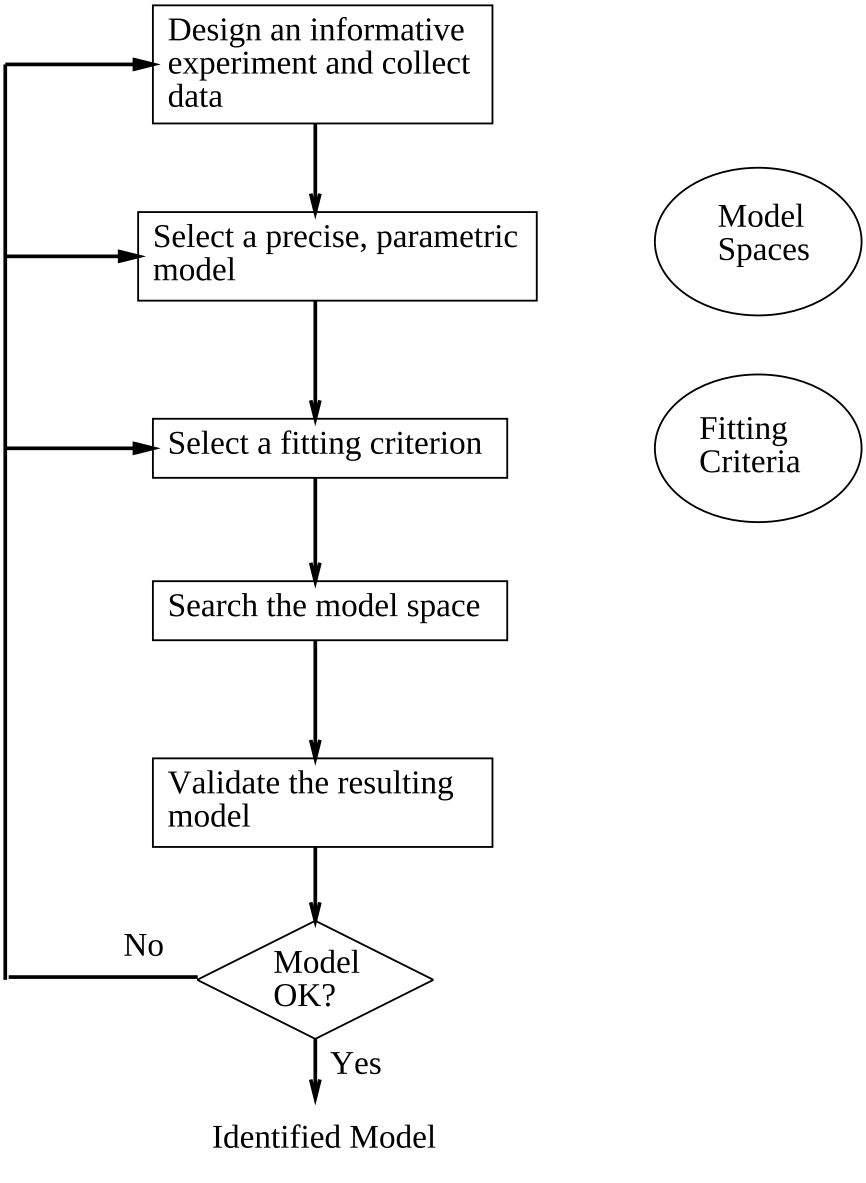


Figure 4.3: The steps in system identification.

- Because systems that are in continuous operation do not always exercise their full dynamics over a fixed period of time, they will often generate uninformative data. This can cause the identification method to converge to the wrong model, resulting in a model that may falsely signal a discrepancy with the observations.

- Because our knowledge of the underlying process is uncertain, we may not be able to parameterize model imprecision. Even if we can, this may result in many models to check, thus increasing the amount of refinement we must do.

- One possible solution to the imprecise model problem is to use a non-parametric (or more appropriately, a highly-parameterized) model, for example, by replacing unknown functions with neural-network approximators. However, the addition of such functions greatly increases the size of the parameter space that must be searched, which can impede the efficient search of the model space.

As a result of these problems, monitoring applications cannot use the power of refinement that system identification provides in their tracking component. In order to overcome this weakness, we have developed a new identification method called SQUID (for SemiQUantitative IDentification) that uses model spaces defined by SQDEs and is robust in the face of uninformative data while remaining computationally tractable.

4.2 Details of SQUID

4.2.1 Overview

Refinement can be viewed as a process where measurements are matched to a model space. The portion of the model space that does not match is refuted, resulting in a better description of the underlying process. Because measurements and models are not directly comparable, we must match model *predictions* to observations and then re-map the results into the model space (see Figure 4.4). We define the *trajectory space* of a model space to be the set of all trajectories produced from each individual model in the model space. The mapping from model space to trajectory space is done via a simulator and the reverse mapping is done by an *inverse simulator*. Note that the quality of the simulation processes has a great deal of impact on refinement. In particular, we require that the simulation and inverse simulation methods be conservative so that portions of the model and trajectory spaces are not eliminated through simulation artifacts.

In traditional identification, only one model at a time from the model space is matched against the measurements. While this approach simplifies both the simulator (which can use standard numerical methods) and the inverse simulator (which is trivial since there is a one-to-one mapping between a precise model and its behavior), it forces

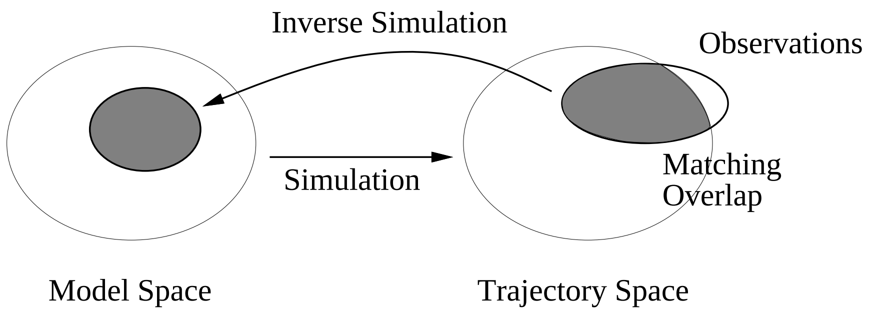


Figure 4.4: Refinement matches observations to the trajectory space of a corresponding model space. It then infers a smaller model space that is consistent with the smaller trajectory space. In traditional identification, this process is simplified because only one trajectory is compared at a time, so the simulation and refinement operators are easy to implement.

one to view the model space as a collection of independent models, ignoring any natural similarity between models.

In contrast, SQUID views the model space as a collection of sets of models that are related by abstract properties of their associated trajectories (monotonicity, location of critical points, etc.). Matching is then performed between these abstract properties of the trajectory space and the corresponding properties of the observations (see Figure 4.5). We call this process *trend matching* to distinguish it from the *raw data matching* of traditional methods.

Trend matching has several advantages over raw data matching:

- Abstract properties describe sets of models, thus trend matching can rule out multiple models at once.

- Since abstract properties are simpler than precise functional forms, trend matching can be cheaper than data matching.

While trend matching has many desirable characteristics, it requires more general methods for simulation and inverse simulation since regions of the model and trajectory spaces are considered rather than single elements. SQUID adopts the SQSIM framework for describing model spaces and trajectory spaces. Model spaces are represented using the SQDE, and by applying SQSIM, their corresponding trajectory spaces can be generated.

Using SQSIM provides a representation and simulator suitable for model refinement. What remains is to define the inverse simulator that maps the trajectory space back to the model space. As we will show, this process can can also be implemented using the Q2

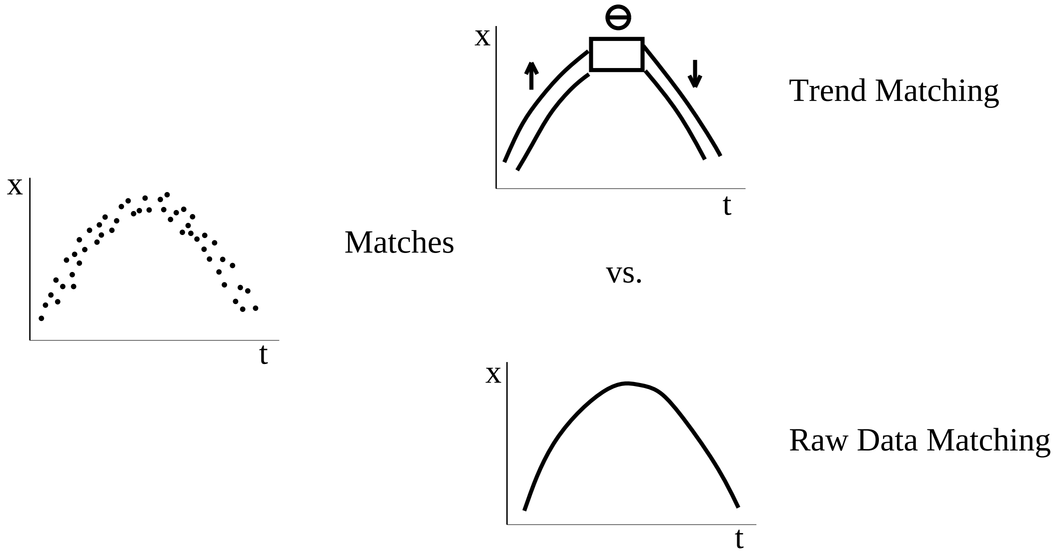


Figure 4.5: Trend matching uses abstract properties of the trend (monotonicity, location of events, and dynamic envelope) to compare data to models (top). Traditional methods use specific function matches for comparison (bottom).

portion of SQsim. The resulting refinement method provides a robustness guarantee with respect to uninformative data that traditional identification cannot provide.

There are three steps to SQUID:

1. Form an SQ Trend for each measured variable. This trend consists of trajectory, event, and dynamic envelope information for each measured variable.

2. Map the SQ Trends to the SQ behavior generated by SQsim for the SQDE.

3. If no match is found, the SQDE is refuted. Otherwise, refine the SQDE by using the inverse simulator to rule out those portions of the model space that could not have generated the match.

These steps are discussed more fully in the following sections. As an aid to following the discussion, we will apply SQUID to the model $y'' = -9.8, \quad y(0) = 0, \quad y'(0) = v_0, \quad v_0 \in [20, 60]$ with data drawn from the same model with $v_0 = 50$ and additive noise with standard error of 5. This model represents the effect of gravity on an object thrown into the air with an initial velocity v_0.

4.2.2 Requirements for Using SQUID

SQUID makes the following assumptions about the identification problem:

- A semiquantitative model of the process to be identified exists in the form of an SQDE.
- Each measured variable has the following properties:
  - The measurement signal can be viewed as a "pure" signal corrupted by additive Gaussian noise of zero mean and fixed variance.
  - The measurements are sampled at a frequency fast enough such that the dynamics of the pure signal can be reconstructed.
  - The variance of the noise is known (although this value may be conservative).

4.2.3 Forming the Semiquantitative Trend

The SQ trend of a measurement stream consists of qualitative, event and dynamic envelope descriptions for each variable in the measurement set. The first two components are generated through a process referred to as qualitative filtering (or *binning*) which breaks the measurement stream into monotonic segments (or *bins*). The dynamic envelope is generated by fitting envelopes to bins using MSQUID.

Qualitative Filtering (Binning)

Qualitative filtering breaks the measurement stream into monotonic regions and intervening extrema. Each region (or *bin*) has a *sign* ($\uparrow, \ominus$, or $\downarrow$) which provides a segment of the qualitative trajectory description of the trend.

The qualitative kernel function The qualitative filter operates by applying a *qualitative kernel function* to a window of fixed size that is slid across the measurement stream. We denote the window starting at index i of the measurement stream by $\mathbf{w}_i$. The kernel function $k(\mathbf{w}_i)$ returns one of three *kernel values*:

- $\uparrow$ if $\mathbf{w}_i$ contains a monotonically increasing segment,
- $\downarrow$ if $\mathbf{w}_i$ contains a monotonically decreasing segment, and
- $*$ if the monotonicity of $\mathbf{w}_i$ is unknown.

To determine monotonicity, the kernel function computes the slope of a linear least-squares fit to the data within the window using the formula

$$slope = \frac{\sum_i (t_i - \overline{t})(y_i - \overline{y})}{\sum_i (t_i - \overline{t})^2}$$

where (t_i, y_i) is the ith data-point in the window and the bar indicates the average value over the window. Because the measurement stream includes noise, it is not sufficient to use the slope directly to determine the sign returned by the kernel function. Instead we must determine if the slope is significantly different from zero. In our implementation, a slope is considered non-zero if it falls outside 3.5σ confidence range where

$$\sigma = \frac{\sigma_v}{\sqrt{\sum_i (t_i - \bar{t})^2}} \tag{4.1}$$

and σ_v is the given standard deviation of the measurement stream. This confidence range gives an almost 100% certainty that a slope outside of the range is not zero. If the slope does not fall outside this range, the kernel value is $*$.

The behavior of the kernel function is related to the width of the window and the dynamics of the measurement stream. Since our goal is to detect the qualitative dynamics of the underlying signal contained in the noisy measurements, we must select a window large enough to filter the noise, yet small enough to capture signal dynamics. In this work, we assume that the minimum window size necessary to ignore noise is given[2]. This window size is equivalent to the frequency needed for constructing a low-pass filter for the signal and is based on knowledge of σ_v, the standard deviation of the noise component of the measurement stream.

The binning strategy Binning assigns signs to data-points in the measurement stream. Note that this is not straightforward since $k(\mathbf{w}_i)$ describes the slope over all of $\mathbf{w}_i$ and not just at a single point. Instead, signs can be determined only by comparing the kernel values of adjacent windows. Let $sign(i)$ be the sign assigned to measurement i and consider two adjacent windows $\mathbf{w}_i$ and $\mathbf{w}_{i+1}$. There are four cases to consider (see Figure 4.6):

1. If $k(\mathbf{w}_i) = k(\mathbf{w}_{i+1}) =\uparrow$ or $\downarrow$ then $sign(i+1) = k(\mathbf{w}_{i+1})$ since any extremum in $\mathbf{w}_{i+1}$ could only occur after measurement $i+1$.

2. If $k(\mathbf{w}_i) =\uparrow$ and $k(\mathbf{w}_{i+1}) =\downarrow$ (or $k(\mathbf{w}_i) =\downarrow$ and $k(\mathbf{w}_{i+1}) =\uparrow$) then there must be an extremum inside $\mathbf{w}_{i+1}$ since there is a slope in change.

3. If $k(\mathbf{w}_{i+1}) = *$ then $\mathbf{w}_{i+1}$ may or may not contain an extremum and so no sign assignment can be made.

4. If $k(\mathbf{w}_i) = *$ and $k(\mathbf{w}_{i+1}) =\uparrow$ or $\downarrow$ then a sign assignment can be made. Let $j < i$ be the last data-point that has a sign. There are two cases:

[2] Actually, it is possible to determine this window size directly from the measurement stream using techniques developed for *scale-space filtering* (see Section 4.5.1).

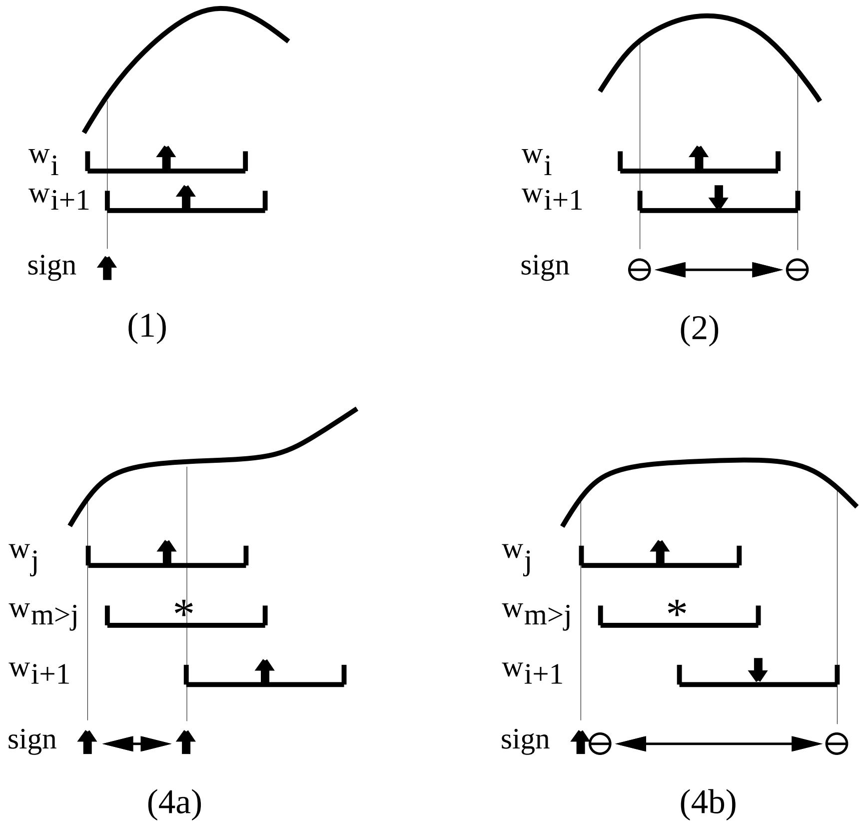


Figure 4.6: Determining signs from kernel values. Numbers in parentheses refer to the corresponding cases in the text.

(a) If $sign(j) = k(\mathbf{w}_{i+1})$ then all points between j and $i+1$, inclusive, must have the same sign since there has been no explicit sign change.

(b) If $sign(j) \neq k(\mathbf{w}_{i+1})$ then there is an extremum somewhere between j and the end of $\mathbf{w}_{i+1}$.

2

Given this interpretation for changes in kernel values, we can construct a qualitative filtering method. The filter outputs a sequence of bins where each bin is a contiguous sequence of data-points with the same sign. The implementation uses two bins to construct these monotonic regions – a current bin (or *cbin*) which holds points in the current monotonic region and an unknown bin (or *ubin*) which holds points whose monotonicity is unknown.

The qualitative filtering process proceeds as follows:

1. Apply the qualitative kernel to $\mathbf{w}_i$.

2. - If $k(\mathbf{w}_i) = *$, append measurement i to *ubin*.
   - Else if $k(\mathbf{w}_i) \neq *$ and *ubin* is not empty then
     - If $k(\mathbf{w}_i) = sign(cbin)$ then append all of *ubin* to *cbin* and then append measurement i to *cbin* and clear *ubin*.
     - Else if $k(\mathbf{w}_i) \neq sign(cbin)$ then output *cbin*, create a new bin with sign $\ominus$ and append all points in *ubin* and then all points in $\mathbf{w}_i$ and output it. Finally, set *cbin* to a new bin with sign $k(\mathbf{w}_i)$. Advance the window to the measurement following the last one in $\mathbf{w}_i$.
   - Else if $k(\mathbf{w}_i) = sign(cbin)$ then append measurement i to *cbin*.
   - Else it is the case that $k(\mathbf{w}_i) \neq *$ and $k(\mathbf{w}_i) \neq sign(cbin)$. Output *cbin*, create a new bin with sign $\ominus$, and append all points in $\mathbf{w}_i$ and output it. Finally, set *cbin* to a new bin with sign $k(\mathbf{w}_i)$. Advance the window to the measurement following the last one in $\mathbf{w}_i$.

3. Advance the window to measurement $i + 1$ and goto Step 1.

The above procedure uses a fixed-size window to compute kernel values. Unfortunately, a single window size is insensitive to slopes below the 3.5σ threshold dictated by Equation 4.1. By selecting a larger window size, we can reduce the threshold at which a kernel function returns $\uparrow$ or $\downarrow$ (since σ depends on the window size via the summation in the denominator), but this larger window size might miss dynamics in the signal. One approach to this problem would be to filter the measurement stream with windows of all sizes and then select the "best" window size based on some criterion[3]. For our needs, however, it suffices to guarantee that we can detect *any* extremum which is not due to signal noise. Thus, we only want to consider larger filter windows when the data suggests that they may be needed.

In our implementation, we opportunistically select increasing window sizes whenever the filter runs into a large region of windows with kernel value of $*$. Starting with a window size of N that is selected to filter out noise, if more than $3N$ points collect in *ubin* we create a new window of size $2N$ and re-process the measurement stream starting at the first measurement in *ubin*. If the kernel value for this larger window is not $*$, then this window is used to bin the data. If the kernel value for this larger window is $*$ then the measurement stream continues to be processed by both window sizes and if the smaller one finds a kernel value other that $*$, that window is used to bin the data and the larger window is discarded. If *ubin* increases to size $6N$, then an even larger window of size $4N$ is created and the data is filtered with it as well as the smaller windows. Whenever a non-$*$ kernel value is detected,

[3]This criterion could be used to determine the "natural" scales of the signal as is done in scale-space filtering.

this value is used to bin the data and all larger windows are discarded. In this way, the filter selects the appropriate window size as dictated by the measurement stream.

This binning process has several properties:

- It is conservative in that bins of sign $\ominus$ will be larger than necessary. This implies that the monotonic regions will also be conservative since some of their end-points will be contained in the adjacent $\ominus$ bins.
- Signals that include regions with differing time-scales are properly binned.
- Since binning is based purely on the measurement stream, it is in no way affected by the model space to be examined.

Figure 4.7 shows the results of binning data from the data-stream.

Fitting the Bins

Binning determines the qualitative description of the measurement stream. The next step is to determine the quantitative aspects of the data, i.e., the event and dynamic envelope descriptions. Events in the measurement stream correspond to $\ominus$ bins since they represent the precise instants at which a variable reaches an extremum. Dynamic envelopes correspond to the monotonic bins since they represent the time-intervals between the extrema. Given this correspondence, we can find events and dynamic envelopes in the trend as follows:

- For each $\ominus$ bin, the width of the event associated with the bin is determined by the beginning and ending times of the bin. The height is determined by the maximum and minimum values over the bin.
- For each monotonic bin, use MSQUID to determine the dynamic envelope. Since we know the data in the bin is monotonic and we have an estimate on the noise, we can reliably produce a bound over the interval.

The fitting process has the following properties:

- The event descriptions are conservative since they over-bound both the width (time) and height (value) of the event.
- The fitting process is independent of the model space. Thus, the cost of fitting can be amortized over multiple model spaces.

Figure 4.8 shows the result of fitting the bins in Figure 4.7.

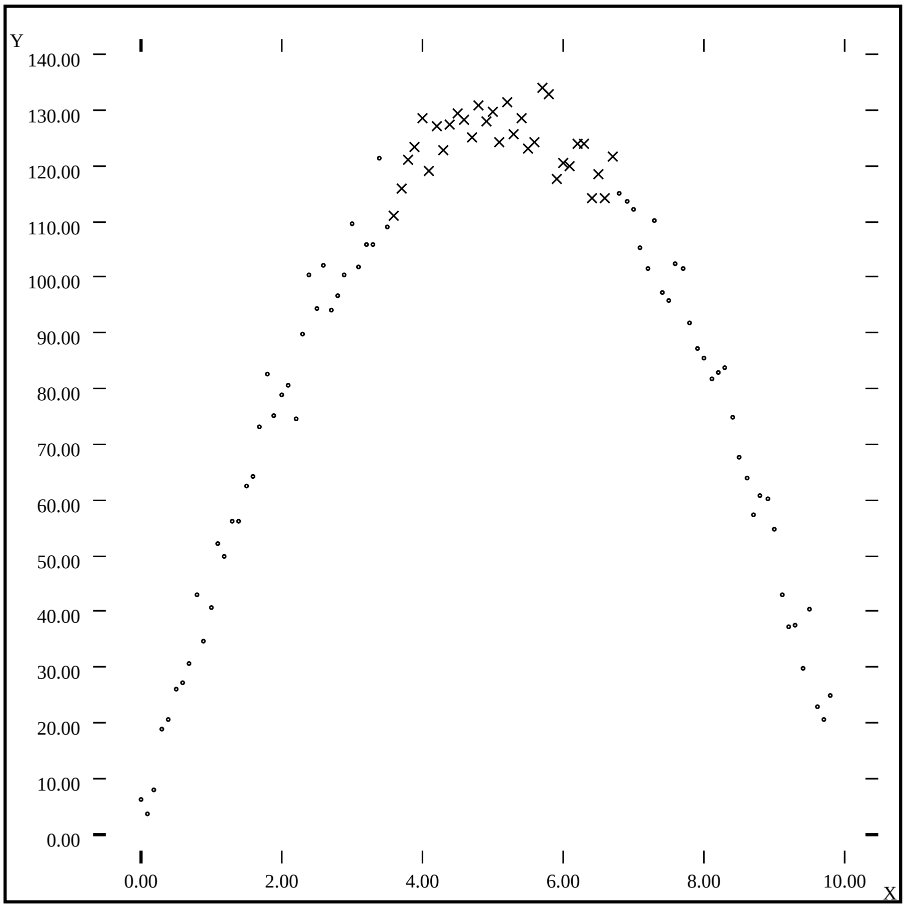


Figure 4.7: Binning a measurement stream. Binning finds three regions for this data – one maximum (x's) surrounded by two monotonic regions.

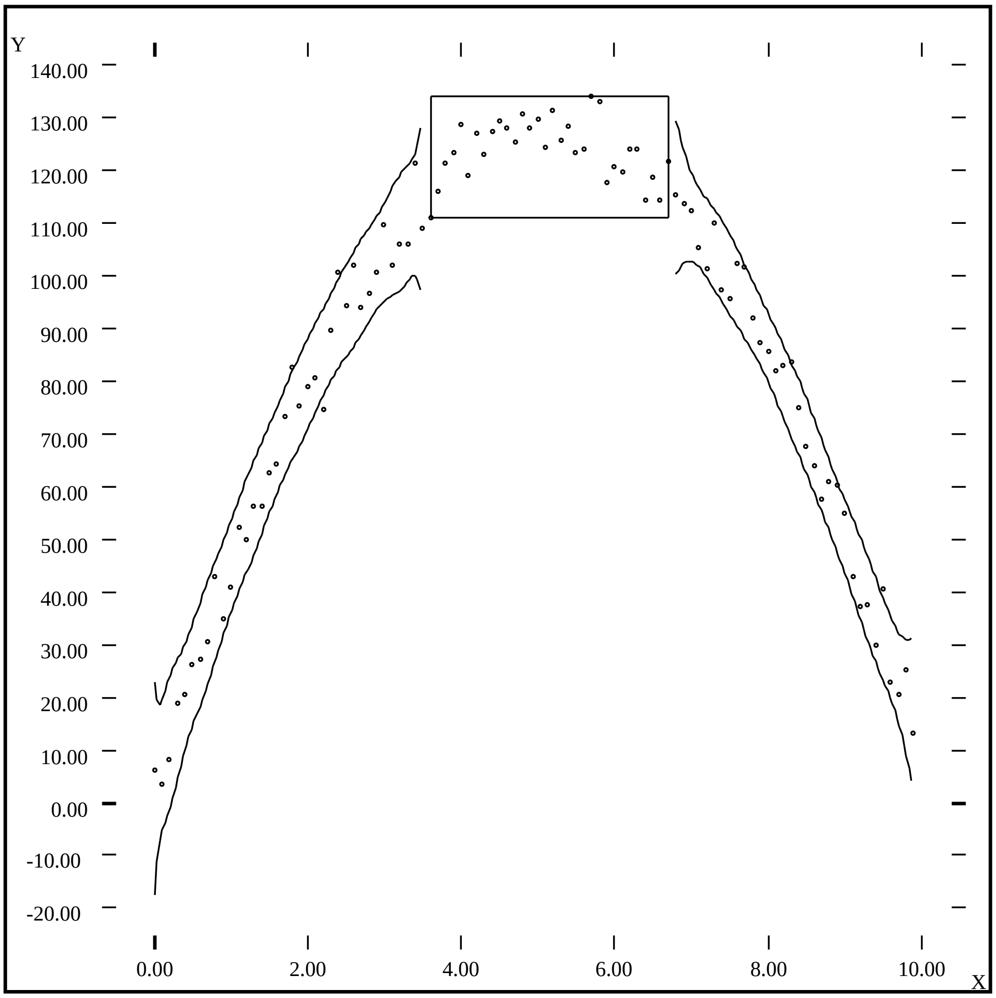


Figure 4.8: Fitting bins. Each $\ominus$ bin is bounded by the smallest box that includes all points in the bin. Each monotonic bin is enclosed using MSQUID.

4.2.4 Mapping SQ Trends to SQ Behaviors

Once the SQ trend of the measurement stream has been computed, it can be mapped to the SQ behavior generated from the SQDE by SQsim. Since both descriptions presumably describe the same physical system, we expect that they should overlap. This overlap represents the section of the trajectory space that is consistent with both descriptions and will normally be smaller than the trajectory space defined by the SQ behavior alone.

We compare the SQ trend and SQ behavior by mapping each of their components separately. For each component, we seek to reduce the size of the SQ behavior so as to yield a smaller trajectory space for the model.

Qualitative Mapping

For the descriptions to be consistent, their qualitative descriptions must match. Intuitively, this match should provide a one-to-one correspondence between the bins of the trend and the qdirs of the behavior, however there are several reasons why the match may be weaker:

- The trend may be a prefix of the behavior.
  The SQ behavior is normally simulated over the time interval $[0, \infty]$. Since the measurement stream contains data over a finite time interval $[0, T]$, it is possible that it may end before some of the qualitative changes in the SQ behavior take place. Using prefix matching eliminates this problem.

- The SQ behavior may contain undetectable extrema.
  Because the behavioral trajectory may include qdir changes of very small magnitude, they may be undetectable in a noisy measurement stream. Thus, we relax the matching process by requiring that the trend regions appear in the proper order within the SQ behavior.

These conditions weaken qualitative mapping considerably. In particular, they permit multiple mappings between a trend and a behavior[4]. However, because qualitative matching compares the relatively simple representations of ordered lists of the three symbols $\uparrow$, $\downarrow$, and $\ominus$, it can be performed with low computational cost. It is thus relatively easy to eliminate SQ behaviors that do not match the SQ trend at this level of description.

Event Mapping

Event mapping ensures that events are consistent between the SQ trend and the SQ behavior. Consistency is maintained by asserting the event information from the trend into

[4]SQUID orders the mappings based on the most exact ordering first. Subsequent mappings are considered only when the current mapping fails.

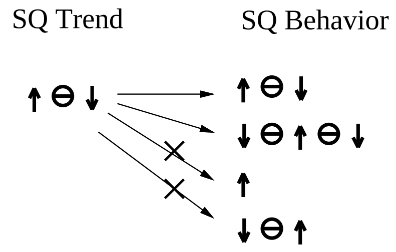


Figure 4.9: Qualitative mapping maps prefixes of the SQ trend to a SQ behavior. Because behavioral distinctions may not be detectable in noisy data, the matching process requires only that qdirs of the trend occur in proper order in the behavior.

the behavior and running Q2 over the augmented description. Note that if the trend event is larger than the behavior event (because the measurements are not very constraining), it will not reduce the existing bounds on the event. If, however, it is more precise, Q2 will propagate this precision to other variables, which may refute an inconsistent mapping.

Because event mapping operates on the fixed set of symbols defined by the SQ events, its cost is independent of the complexity of the SQ trend.

Dynamic Envelope Mapping

Dynamic envelope mapping ensures consistency between the dynamic envelopes for each monotonic region of the trend (as determined by MSQUID) and the dynamic envelope of the behavior. The dynamic envelope for each trend region holds over the time-range of the measurements that compose the region. Consistency between the trend monotonic regions and the SQ behavior is maintained by intersecting the dynamic envelopes of each description over the time-range of the monotonic region. If the intersection is empty then the behavior is refuted.

Trend $\rightarrow$ behavior mapping provides the following benefits:

- By mapping each component of the trend and behavior separately, it eliminates mismatches more efficiently. For example, a decreasing behavior can be ruled out by an increasing trend without resorting to detailed numerical analysis.
- The mapped trajectory space is conservatively reduced since qualitative mapping is conservative and the event and dynamic envelope descriptions produced by binning and fitting are conservative. Producing a conservative trajectory space aids in providing a robust inverse simulation method.

4.2.5 Refining the Model

Model refinement is the process of mapping a trajectory space back to the model space that generated it. In the case of SQUID, this mapping takes an SQ behavior and determines the SQDE that covers the smallest set of ODEs that could have produced it while preserving the guarantee that no ODE is excluded unless it is genuinely impossible. This process is referred to as *inverse simulation.* Note that there are two sources of imprecision in the SQDE – variable uncertainty and static envelope uncertainty. The inverse simulator must therefore refine these sources to refine the model.

This section begins by describing a method for refining variable uncertainty by using Q2 to derive bounds on independent variables from dependent ones. This process hinges on deriving bounds for the derivatives of state variables so that Q2 can be run on a behavioral snapshot. Two methods for computing the derivatives of state variables are presented – inference from the bounds of the state variables and inference from the trend dynamic envelopes.

Next, a method for refining static envelopes is described. This method excludes portions of the envelope that are inconsistent with the ranges determined by the behavioral snapshot used to refine variable uncertainty.

Refining Variable Uncertainty

We use the Q2 interval propagator to refine variable bounds. At a time-instant, an ODE system becomes a system of algebraic equations whose left-hand sides are the instantaneous derivatives of the state variables. Q2 solves this algebraic equation system for each model variable by manipulating the SDE portion of the SQDE to form equations such that for each equation of n variables, n equations are generated with a different variable on the left-hand side. For example, for the equation

$$x' = f(c, x)$$

Q2 produces equations whose left-hand sides are x', c, and x. In particular, one of these equations is

$$c = g(x', x).$$

With this equation, Q2 can thus run the SDE "in reverse" and derive constraints on independent variables (c) from dependents (x and x')[5]. If, as we assume, our measurements reduce the bounds on the dependent variables, Q2 will be able to reduce the bounds on the independents. If, however, the measurements do not contain enough information (for

[5]Note that this property is provided since the mathematical operations that can occur in each equation are compositions of the arithmetic operators (which have clearly defined inverses) and monotonic functions (which have user-specified inverses).

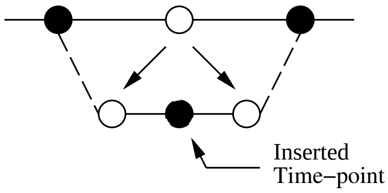


Figure 4.10: Time-point insertion generates a new SQsim time-point state (and adjacent time-interval states) and inserts it into an existing behavior. The ranges for each variable are initially set to the range determined by the dynamic envelope at the desired time. Q2 then propagates these ranges, resulting in improved bounds on all model variables.

example, because we do not have adequate observability, too much noise, or uninformative data) then the measured bounds will be greater than the original bounds and Q2 will be unable to reduce the model space, but no information will be lost. This is in sharp contrast to standard system identification which can be led astray by uninformative data.

As part of SQsim, Q2 runs only at qualitative time-point states which are by definition time-instants. As part of the SQUID event-mapping process, Q2 is also run to unify the trend and behavior event descriptions. Thus, event mapping is a refinement operation. Unfortunately, since Q2 requires a time-point state, it cannot be used over time-intervals since variables over a time-interval do not represent instantaneous values. The trend dynamic envelopes, however, *do* provide instantaneous values for variables and we would like to exploit this information for refinement. Thus, to extend Q2 propagation into time-intervals, we require a method for generating *snapshots* of the SQ behavior at any time.

A simple approach for generating snapshots is to create a time-point state where the ranges of each variable are determined from the dynamic envelope of the SQ behavior at the time of interest. Unfortunately, this ignores the fact that the SQ behavior only implicitly records derivative information between qualitative time-points. To explicitly determine these values, Q2 uses the Mean Value Theorem at adjacent time-point states by solving equations of the form:

$$span(x', t_1, t_2) = \frac{x(t_2) - x(t_1)}{t_2 - t_1}$$

where all variables take on interval values and $span(v, t_1, t_2)$ is the interval convex hull of the variable v over the time interval $[t_1, t_2]$. We could utilize this method for computing derivatives at the snapshot by splicing the new state into the proper interval of the SQ behavior and then use the adjacent states to compute derivative values (see Figure 4.10). This method is the essence of the Q3 behavior refinement method [Berleant and Kuipers, 1992] and the time-point insertion method of MIMIC.

While time-point insertion can improve bounds on the derivatives of the model, it is very expensive to use. In particular, inserting new time-points into an SQ behavior increases the number of events that Q2 needs to process, since each time-point generates a new event for each variable. Because the Mean Value Theorem equations can propagate changes to adjacent time-point states, it is possible that changes to an event $x(t_2) = x_2$ may update the event $x(t_1) = x_1$ which will then force consideration of other events at t_1 in addition to the events at t_2. This process can continue to cascade to other time-point states resulting in even more computation. As more time-points are inserted, the number of events and their corresponding equations increases, exacerbating the problem even further. Such propagations limit the number of events that can be inserted into a behavior.

Another problem with time-point insertion is that it requires that a "gap" exist such that there is no overlap in the time range of the time-points surrounding the interval in which to splice. If no such gap exists, the behavior must be processed so that a gap is created (see Figure 4.11). These methods involve splitting the time-ranges of the surrounding events and attempting to rule out the region of overlap. This process can be very expensive and is not always guaranteed to find a gap[6].

Because of these complexities, we have chosen a different method for computing derivatives at new time-point states. We begin with the dynamic envelope prediction for each derivative. Next, we improve this using the following methods:

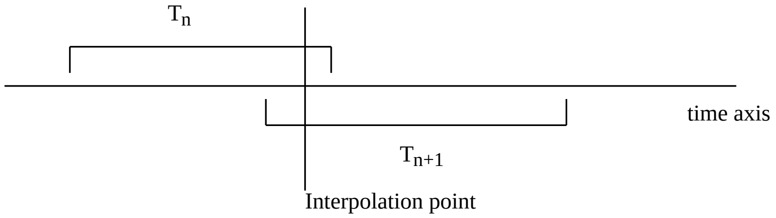
Inferring Derivative Bounds from State Variables If we know the sign of the first and second derivatives of a state variable x over an interval $I = [t_0, t_2]$, we can infer a bound on the derivative x' by computing slopes over subranges of I. interval. Consider the case where x is monotonically increasing over I with a decreasing second derivative (i.e., concave down). Assume we are interested in the derivative of x at $t_1 \in I$. Note that the following facts hold:

1. If the slope at t_1 is m, then for all $t < t_1$, $x'(t) \geq m$,
2. $x(t)$ lies within the dynamic envelope for x over I, and
3. at any $t < t_1$ the maximal slope of x over $[t, t_1]$ is

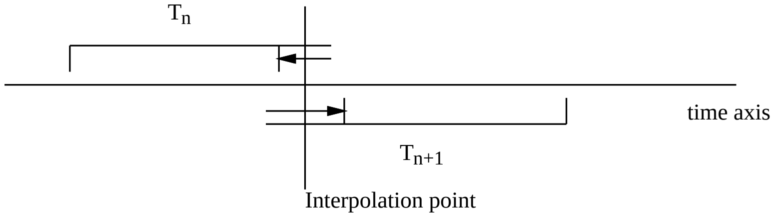
$$m(t, t_1) = \frac{\overline{x}(t_1) - \underline{x}(t)}{t_1 - t}.$$

Therefore, the maximum slope at t_1 is bounded by the maximum value of $m(t, t_1)$ for $t_0 \leq t \leq t_1$ (see Figure 4.12). Similarly, the minimum slope at t_1 is bounded by the minimum value of $\frac{\overline{x}(t) - \underline{x}(t_2)}{t_2 - t}$ or zero. Additional equations can be derived for monotonically increasing but concave down and monotonically decreasing concave up or down functions.

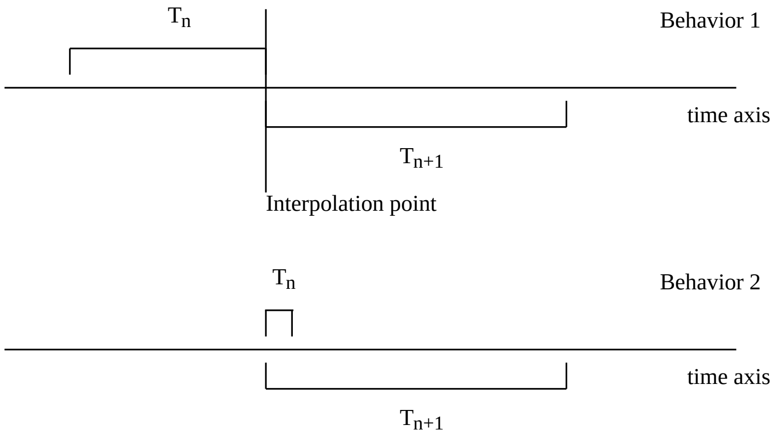
[6]Note that while the SQ trend separates regions of differing monotonicity, the mapping process may place multiple qualitative time-point states within one such region. Thus, trend mapping by itself is not enough to guarantee a gap between qualitative time-points.



(a) Time-points T_n and T_{n+1} overlap. There is no point that is clearly in the interval between them.



(b) TIS attempts to create a gap between the time-points by shrinking their ranges. Each range can be shrunk by searching for subranges that are inconsistent with the Q2 equations for the behavior.



(c) Behavior Splitting attempts to create a gap by splitting the behavior into two behaviors, each with a subset of the total range of T_n. Note that interpolation can only occur in the first behavior.

Figure 4.11: What to do when there is no gap. Overlapping time-point descriptions prevent interpolation to reduce interval size (a). Q3 uses two methods for finding gaps – Target Interval Splitting TIS (b) and Behavior Splitting (c). The cost of these methods can make state-spliced measurement insertion prohibitively expensive.

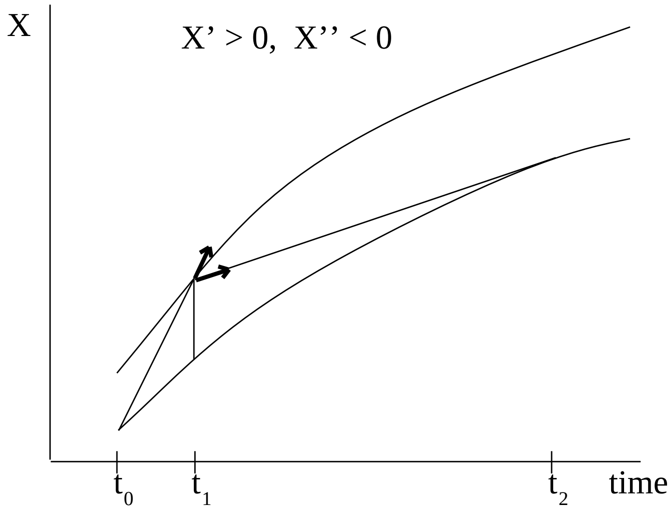


Figure 4.12: Computing bounds on the derivative of a state variable from its dynamic envelopes. The signs of the first and second derivatives of x provide information to determine bounds on $x\prime$ from the dynamic envelope of x.

Note that this calculation is essentially what time-point insertion computes, except without the overhead of generating additional qualitative time-point states.

Note that we can determine bounds on the interval I from the width of the SQ behavior events since they are guaranteed not to overlap in time. This is because binning naturally breaks the trajectory into non-overlapping regions of monotonic behavior and mapping assigns these regions to the SQ behavior.

Inferring Derivative Bounds from Trend Dynamic Envelopes If we have measurements for state variables, then MSQUID will compute a fit to the data over each monotonic trend segment. This fit can be used to compute a dynamic envelope for the derivative of the function as well. MSQUID describes the fitting function by the equation

$$\hat{y} = \sigma\left(\sum_{j=1}^{n_h}\left[w_{o[j,1]}\sigma\left(w_{i[1,j]}t + w_{i[2,j]}\right)\right] + w_{o[n_h+1,1]}\right)$$

which has derivative

$$\begin{aligned}\frac{d\hat{y}}{dt} &= \sigma'(z + w_{o[n+1,1]})\sum_{j=1}^{n_h} w_{o[j,1]}\sigma'(w_{i[1,j]}t + w_{i[2,j]})w_{i[1,j]} \\ z &= \sum_{j=1}^{n_h} w_{o[j,1]}\sigma(w_{i[1,j]}t + w_{i[2,j]})\end{aligned}$$

where $\mathbf{w}$ is the weight vector. MSQUID fits this model to the data which produces an estimate $\hat{\mathbf{w}}$ of $\mathbf{w}$.

As described in Section 3.3, one can compute an envelope on a nonlinear function via linearization. Following exactly the steps outlined in that Section, but with $\frac{d\hat{y}}{dt}$ in place of $\hat{y}$, we compute the envelope around the slope to be

$$\begin{aligned}
\underline{e}(t) &= \frac{d\hat{y}(t)}{dt} - \delta(t) \\
\overline{e}(t) &= \frac{d\hat{y}(t)}{dt} + \delta(t) \\
\delta(t) &= s\sqrt{pF(\alpha; p, n-p)}||\mathbf{v^T}\mathbf{R_v}^{-1}
\end{aligned}$$

where s is the sample standard deviation, p is the length of $\mathbf{w}$, F is the α quantile of the F-statistic with p and $n-p$ degrees of freedom, n is the number of samples, $\mathbf{v}$ is the linearization vector such that $\frac{d\hat{y}(t)}{dt} = \mathbf{v^T w}$, $\mathbf{V}$ is the Jacobian of $\mathbf{v}$ with respect to $\mathbf{w}$, and $\mathbf{R_v}$ is defined by $\mathbf{V} = \mathbf{Q_v}\begin{bmatrix} \mathbf{R_v} \\ 0 \end{bmatrix}$.

Note that this envelope requires the computation of a different values for $\mathbf{v}$ and $\mathbf{V}$ than is used for computing bounds on y, but it does use the same $\hat{\mathbf{w}}$ which is the expensive part of the MSQUID algorithm.

Figure 4.13 shows the result of computing the derivative of y in the gravity model using this approach. Note that the bounds are very weak near the end of the monotonic interval (since there are fewer nearby points to constrain the derivative) however the center area contains a very good bound. In contrast, the state variable computation method provides better bounds at the ends of segments since it has more data to work with there. Thus these two methods for computing derivatives are complementary.

Note that if we do not have measurements of a state variable, we may have a related variable $v = f(x)$ for which we could compute an estimate and propagate it to x.

For the gravity model, variable refinement improves only the lower bound of y' at $t = 0$ over its initial value. This leads to the reduction of the initial state uncertainty from $[20, 60]$ to $[36.3, 60]$. The final prediction for y is shown in Figure 4.14.

Refining Static Envelopes

Static envelopes define imprecise functional relationships between a dependent variables and a vector of independent variables in terms of a qualitative description and a pair of bounding functions. A static envelope summarizes a family of parametric relationships and

[8]The upper bound for the state variable computation at $t = 0$ is ∞ and is not shown.

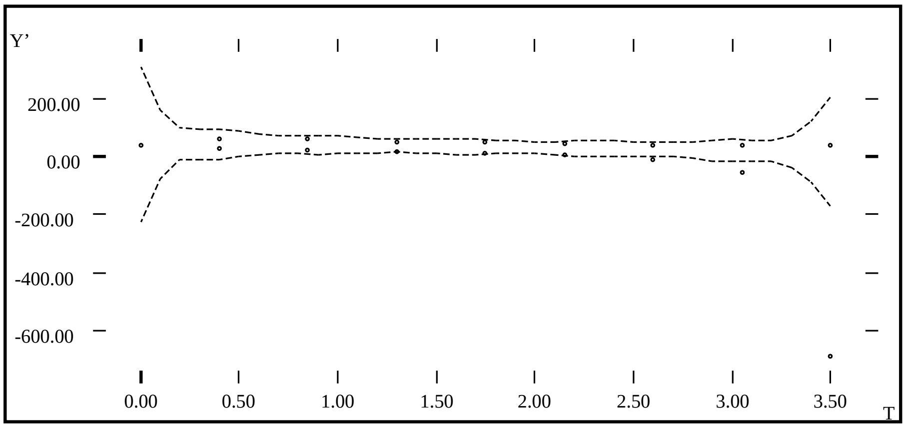


Figure 4.13: Computing the derivative y' of a state variable y from its MSQUID fit for the gravity model. The bounds computed from the state variable envelopes (at selected points) are overlaid for comparison[8].

thus increases the flexibility of the model. In this work, we restrict our interest to univariate monotonic functions of the form $y = f(x), f \in M^{+/-}$.

Refining a static envelope means reducing the width between the bounding functions. Assume that at some time-instant t, the ranges of x and y are $x(t)$ and $y(t)$. These ranges produce a box in the (x, y) plane of the static envelope (see Figure 4.15). This box is analogous to an SQ event description in that the box defines a region in which the true x and y values of the system must lie at t. If the upper-left corner of this box falls inside the static envelope for f then refinement is possible. This is because any point in the region above and to the left of this corner is unreachable. To see this, note that the x value of points in this region are less that the x value of points in the box which is impossible for a monotonically increasing function. We may thus eliminate this region from the envelope. A similar argument rules out the region below and to the right of the lower-right corner of the envelope.

If we also have curvature information about f, we can further refine the static envelope by eliminating regions that violate the curvature assumption. For example, assume that f is concave downwards and that the upper-left corner of the range box is at (x_1, y_1). If we determine the maximal slope at this point to be m (by using the method for inferring derivatives from envelopes, for instance) then we may eliminate all portions of the envelope above the line $y = m(x - x_1) + y_1$ for all $x > x_1$ since any point above this line could only be reached by a path that has a slope greater than m (see Figure 4.16).

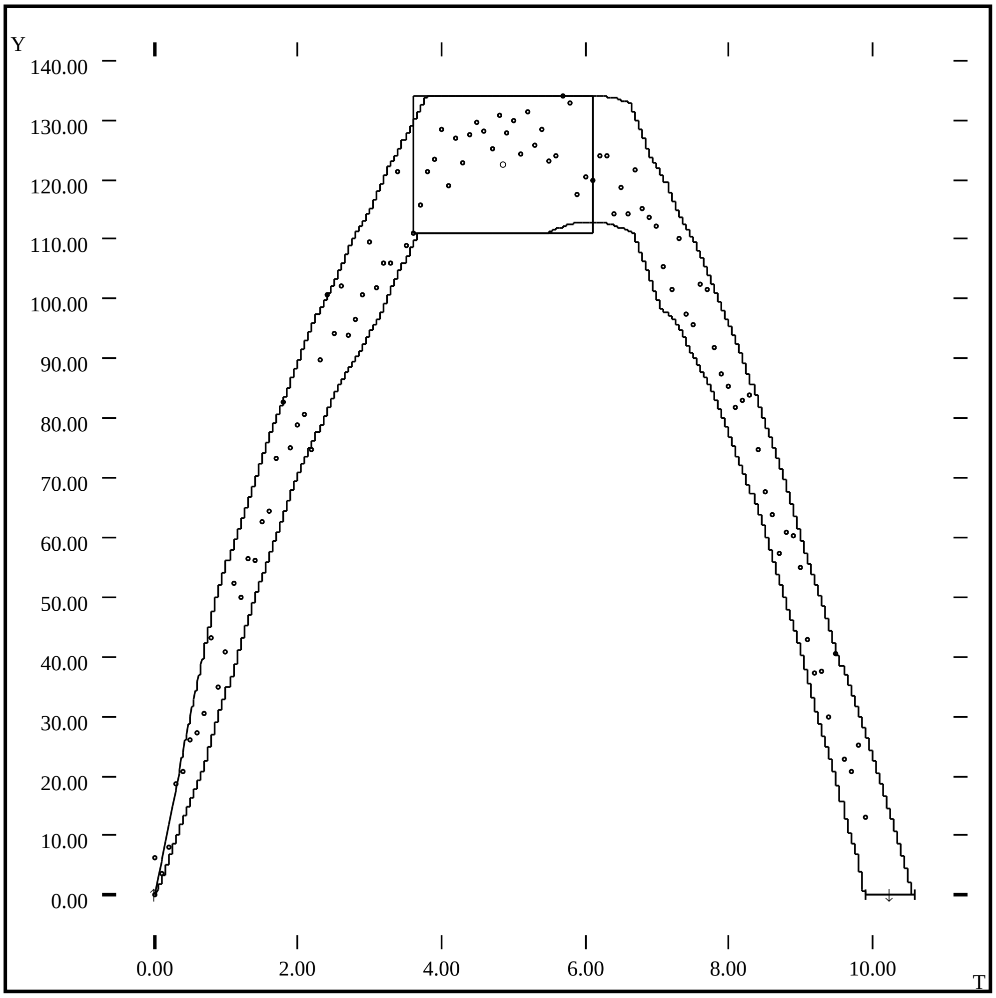


Figure 4.14: The predicted trajectory of the gravity model after refinement. The bound on initial velocity has been reduced from $[20, 60]$ to $[36.3, 60]$. Note that the bound for y near $t = 0$ is better than that of the trend dynamic envelope (Figure 4.8) since the model prediction provides greater constraining power than do the measurements at that time.

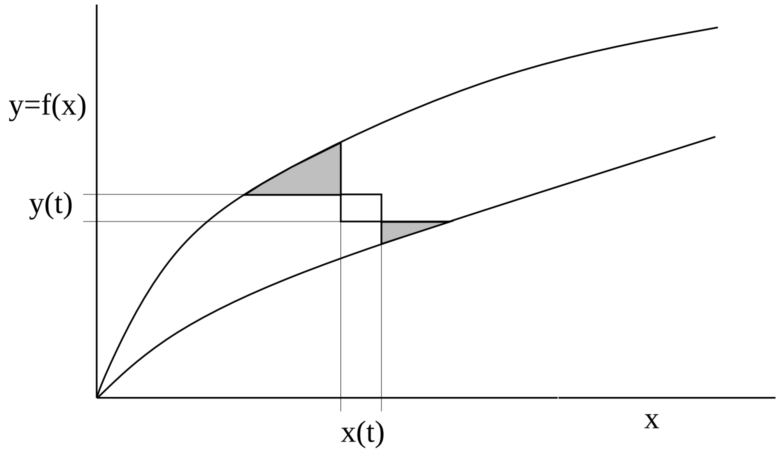


Figure 4.15: Refining a static envelope. Monotonicity requires that an increase in x leads to an increase in y. Thus, if the region in the box contains a point on $f(x)$, the shaded regions cannot contain parts of the function.

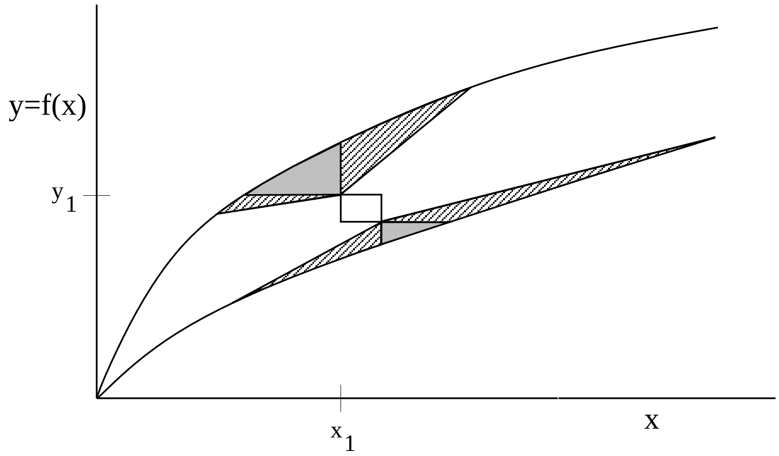


Figure 4.16: Refining a static envelope using curvature information. A concave down function permits us to remove more of the envelope.

Since the gravity model does not include static envelopes, we will demonstrate static envelope refinement on the first-order model

$$A' = 10 - f(A) \qquad\qquad A(0) = 0$$

where $f \in M^+$ with static envelope $2A \leq f(A) \leq 5A$ and the maximum value for A is represented by the landmark $FULL \in [50, 60]$. We use a data source for A computed from $A' = 10 - 3A$ corrupted with additive noise of variance 0.25 and sampled at 10 measurements per time unit. SQUID is then applied to the SQ behavior and data source. Figure 4.17a shows the resulting uncertainty in f as well as the prediction. Note that the static envelope is reduced, although the effect is very localized about the A values of the snapshot states. If we add the further information that f is concave downward (i.e., $f''(A) < 0$) and rerun SQUID on the same data, we obtain the results in Figure 4.17b. Notice that the static envelope is much improved. Figure 4.18 shows the improvement in the prediction both with and without the curvature constraint.

4.3 Experimental Results

This section explores the capabilities and limitations of SQUID as demonstrated through several illustrative models.

4.3.1 The Models

There are four models used in this section:

- A single tank with constant inflow:

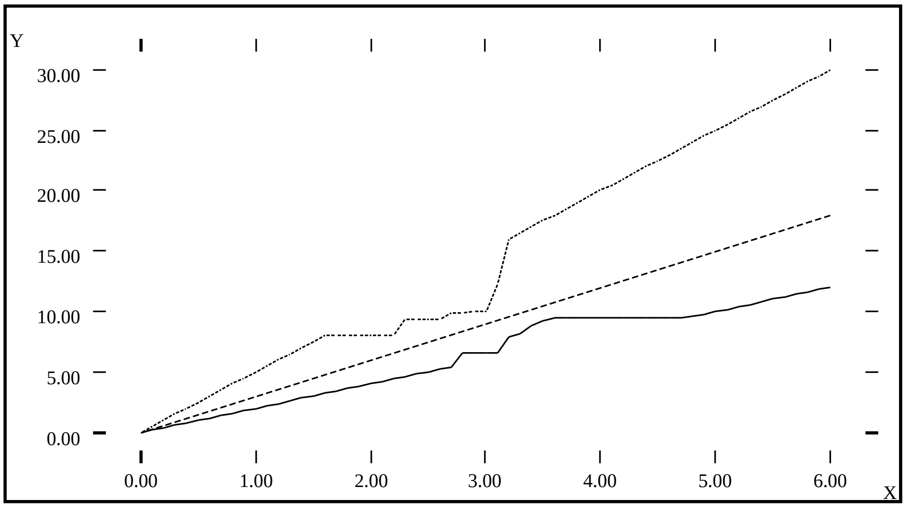
  $$A' = c - f(A)$$

  where c is a constant and f is a monotonically increasing function that is concave down. The variable A has a landmark $FULL = [50, 60]$ which represents the maximum amount that the tank can hold. The system is simulated from $A(0) = 0$.

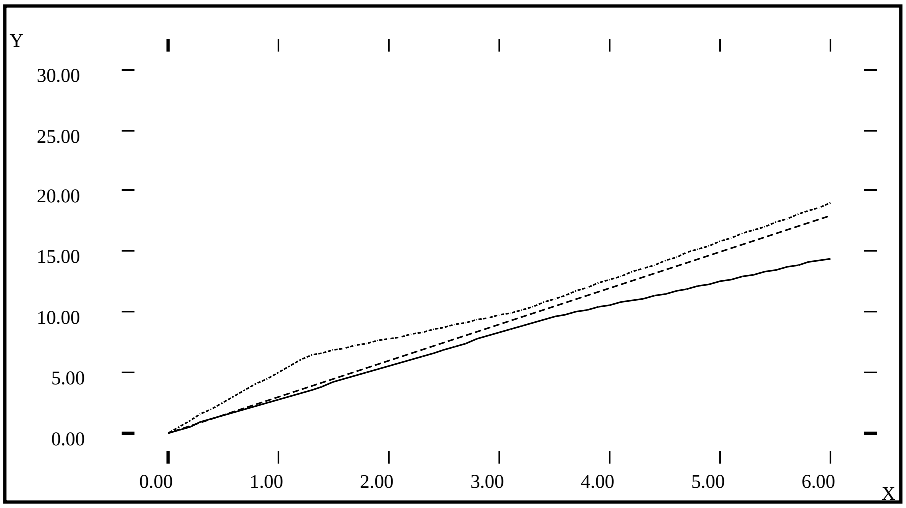
- A two tank cascade:

  $$A' = c - f(A)$$
  $$B' = f(A) - g(B)$$

  where c is a constant and f and g are monotonically increasing functions that are concave down. The variable A has a landmark $FULL = [92, 98]$ and B has the landmark $FULL = [50, 60]$. The system is simulated from $A(0) = FULL$ and $B(0) = 0$.



(a) No curvature constraint.



(b) With curvature constraint.

Figure 4.17: Static envelope refinement. In the first case, no knowledge of curvature or the range of the $FULL$ landmark is given. In the second, the refinement is made with the assumption that the curvature is negative and $FULL \in [50, 60]$. The line representing the true function ($f(A) = 3A$) is shown in each plot.

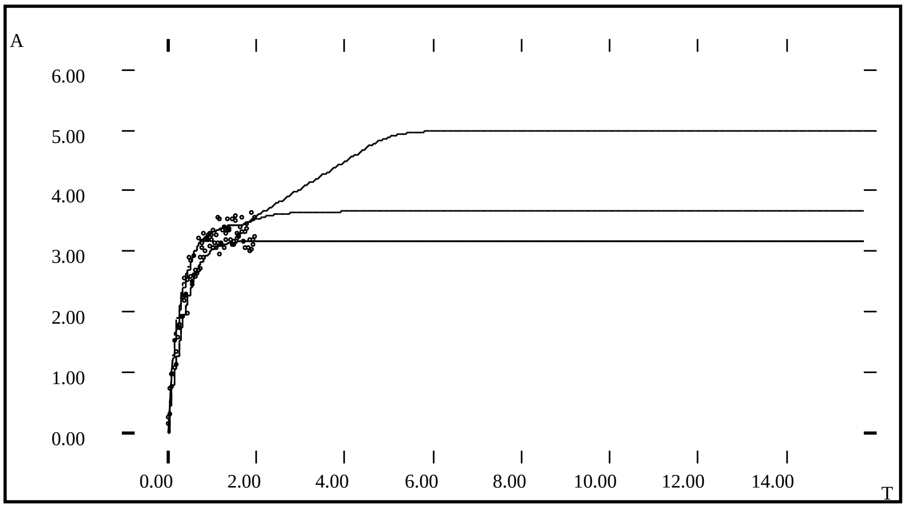


Figure 4.18: Comparison of predictions using the two different refinements from Figure 4.17 together with the data used to generate the prediction. Using the curvature constraint greatly improves the predicted dynamic envelope as seen by the reduction of the upper bound from 5 to 3.6.

- The gravity model

$$y'' = -9.8$$

  simulated from $y(0) = 0$, $y'(0) \in [20, 60]$.

- A Continuously-Stirred Tank Reactor (CSTR)

$$\begin{aligned} \frac{dC_A}{dt} &= \frac{C_{Ai} - C_A}{\tau} - k_0 e^{-E/T} C_A \\ \frac{dT}{dt} &= \frac{T_i - T}{\tau} - h_r k_0 e^{-E/T} C_A \end{aligned}$$

  with $C_{Ai} = 0.9$, $T_i = 340$, $\tau = 10$, $k_0 = 10000$, $E = 5000$, $h_r = -200$.

Each model was simulated using SQsim to produce an SQ behavior tree[9] which was used as input to SQUID along with noisy datasets.

4.3.2 Comparison with Traditional System Identification

One claim made for SQUID is that using a richer model space that includes monotonic functions simplifies the identification process. If we consider the single-tank model, we see

[9]The single tank and gravity models produced only a single behavior.

that this is indeed the case since f is imprecisely-defined and thus cannot be easily captured by a single parametric model.

An alternative approach, however, that does permit an approximation of a single model uses a high-parameter estimator to replace the monotonic function. Since f is monotonic, we can use the MSQUID function estimator and translate the model to $A' = c - \hat{f}(A; \hat{w})$ where $\hat{w}$ is a vector of weights for the function estimator $\hat{f}$. We can then use this model to identify the behavior of the system. Unfortunately, this approach is highly inefficient because the model space is very difficult to search. There are three reasons for the inefficiency:

1. Each step along the gradient of the objective function requires the solution of an ODE system. Since hundreds of iterations may be needed, this can be very costly.

2. The gradient can only be approximated by finite differences.

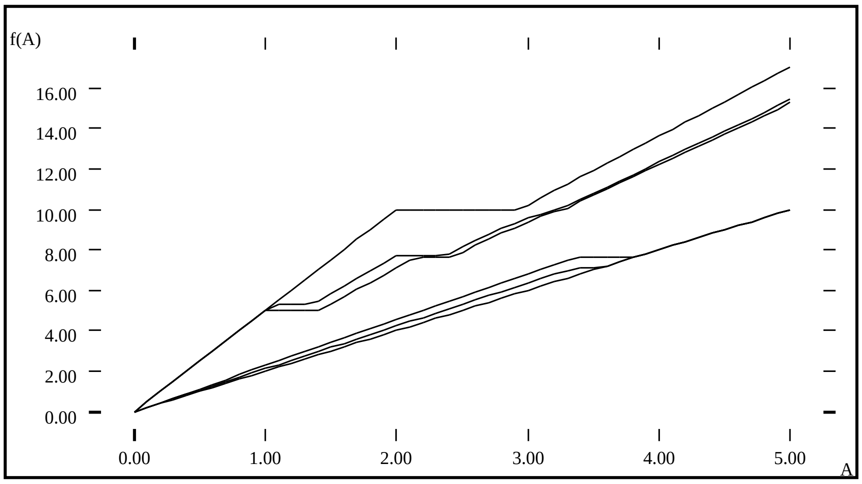
3. The model space may be very sensitive to changes in parameters. Coupled with (2), this means that the gradient search may be unable to compute a reliable derivative which will cause it to converge erroneously, if at all. In fact, in our experiments, we were unable to get the optimizer to converge to any solution.

In contrast, since SQUID does not search this parameter space, it is able to produce a refinement for this example (see Figure 4.18).

4.3.3 The Effect of Uninformativeness

Informativeness is a measure of the degree to which a set of measurements is representative of the dynamics of a process. In general, an informative measurement set provides enough information to adequately reconstruct a model of the system that generated it. The informativeness of data is affected by the level of noise and the amount of dynamics captured in the sample. A design constraint of SQUID is that uninformative data should not bias the refinement process. This section examines uninformativeness as a function of noise and the length of the measurement stream.

We begin by examining the single-tank system with inflow $c \in [5, 10]$ and $2A \leq f(A) \leq 5A$. Figure 4.19 shows the result of using measurement streams of 100 points derived from the model $A' = 10 - 3A$ with noise variances of 0.25, 0.01, and 0.003 in terms of both trajectory and model uncertainty. Note that the static envelope and range on c improve with decreasing noise, thus reducing the model space. Unfortunately, the predicted dynamic envelope shows no improvement as $t \to \infty$. This is due to the effect of multiple uncertainty sources in the model, a situation that we will discuss more fully in Section 4.3.4.



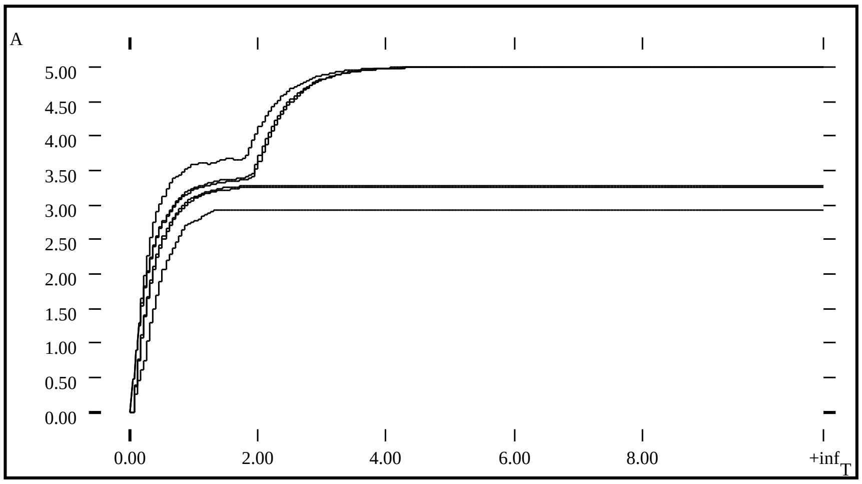
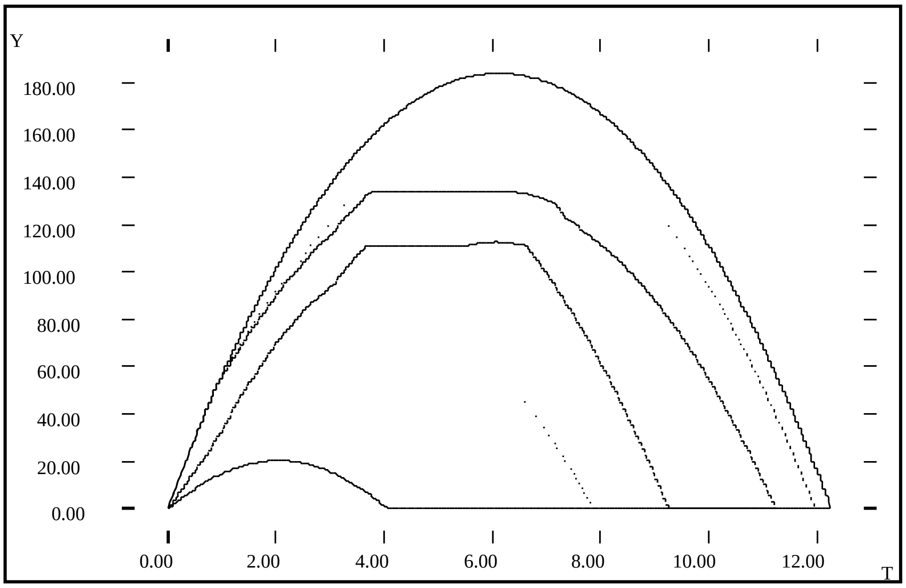


Figure 4.19: The effect of noise on refinement for the single-tank model. As the noise variance decreases, so does the static envelope for f (upper graph). The corresponding ranges on c are $[5.9, 10]$ for variance 0.25, $[7.4, 10]$ for variance 0.01, and $[7.9, 10]$ for variance 0.003. In all cases, a sample of 100 points over the range $t \in [0, 2]$ was used. Although the model improves with decreasing noise, the prediction from the model (lower graph) does not improve as $t \to \infty$. Note that further improvement of the of the dynamic envelope is possible given that it is known that the curvature of the envelope is concave downward, however this information does not lead to improving the model itself, nor does it improve the bound as $t \to \infty$.



# pts	$y'(0)$	y and t at maximum
0	$[20.0, 60.0]$	$[0, 183], [2.0, 6.1]$
30	$[35.2, 60.0]$	$[89, 168], [3.6, 6.1]$
80	$[35.7, 59.8]$	$[112, 134], [3.6, 6.1]$

Figure 4.20: The effect of varying the length of the measurement stream for Y from 30 to 80 points (the sampling rate is 10 samples per unit time). The three envelopes correspond to bounds at 0 (outer), 30 (middle) and 80 (inner). The bounds on the value for $y'(0)$ and the event $y(t) = max$ when $y' = 0$ are given in the table below the graph.

The length of the measurement stream is another factor in informativeness since short streams may not capture all the dynamics of the underlying process. We next examine this effect on the gravity model. Figure 4.20 shows the resulting prediction from identifying increasingly longer measurement streams of y using the gravity model. The first prediction is generated from data that stops before the local maximum while the second makes it slightly past the maximum and the third makes it well past the maximum. Note that the prediction narrows greatly as a function of increasing data length.

4.3.4 The Relation Between Model Uncertainty, Measurement Uncertainty, and Refinement

As can be seen from the tank example in the previous section, the effect of multiple sources of uncertainty can greatly reduce the effectiveness of refinement. This is the flip-side of robustness – because interval arithmetic is conservative, its ability to reduce the model space is also conservative. In this section, we examine the reasons for why this is true.

Refinement in SQUID is affected by both measurement uncertainty and model uncertainty. Measurement uncertainty can be described by the width of the trend dynamic envelope of a measured variable. Model uncertainty can be described by the amount of uncertainty per source (i.e., the widths of the ranges of model parameters and the widths of static envelopes) and the number of uncertainty sources. In order to study the relationship between uncertainty and refinement, we make use of the single tank model

$$A' = C - O_f \qquad\qquad O_f = f(A)$$

where we measure the value of A'[10]. We wish to examine the conditions under which the measurement of of A' permits refinement of $\underline{f}(A)$. Assume that at some time t we have a bound on A' of $[\underline{a}', \overline{a}']$. Intuitively, we would expect that if a model with small uncertainty can be refined by A' then a model with greater uncertainty could also be refined by the same bound.

Single-source uncertainty Consider the single-source uncertainty SQDE derived from the above SDE where $\underline{f}(A) \leq f(A) \leq \overline{f}(A)$ and $C = c$, a precise constant. To refine the model at t, we create a Q2 point-state (or use an existing one) that contains the bounds of all model variables at t. This state represents the uncertainty in $\underline{f}$ by $\underline{O}_f(t)$. Q2 generates the following equations whose left-hand side contains this term:

$$\begin{aligned} \underline{O}_f &= \underline{f}(\underline{A}) \\ \underline{O}_f &= c - \overline{A}' \end{aligned}$$

Let us assume that the bound on $A(t) = [\underline{a}, \overline{a}]$ is such that the first of these equations improves the bound on $\underline{O}_f$ so that $\underline{O}_f = \underline{f}(\underline{a})$. Then in order to improve $\underline{O}_f$ further it must be the case that $c - \overline{a}' > \underline{f}(\underline{a})$. Thus

$$\overline{a}' < c - \underline{f}(\underline{a}) \tag{4.2}$$

defines the maximum value that an observation $[\underline{a}', \overline{a}']$ of $\overline{A}'(t)$ can attain while still permitting refinement of $\underline{f}$.

[10] In this analysis, upper-case names correspond to model variables while lower-case names correspond to scalar values.

Multiple-source Uncertainty Now consider the multiple-uncertainty source SQDE where $\underline{f}(A) \leq f(A) \leq \overline{f}(A)$ and $C \in [\underline{c}, \overline{c}]$. Q2 generates the following equations for defining $\underline{O}_f$:

$$\begin{aligned} \underline{O}_f &= \underline{f}(\underline{A}) \\ \underline{O}_f &= \underline{c} - \overline{A}' \end{aligned}$$

As before, assume that the bound on $A(t) = [\underline{a}, \overline{a}]$ is such that the first of these equations improves the bound on $\underline{O}_f$ so that $\underline{O}_f = \underline{f}(\underline{a})$. Then in order to improve $\underline{O}_f$ further it must be the case that $\underline{c} - \overline{a}' > \underline{f}(\underline{a})$. Thus

$$\overline{a}' < \underline{c} - \underline{f}(\underline{a}) \tag{4.3}$$

defines the maximum value that an observation $[\underline{a}', \overline{a}']$ of$\overline{A}'(t)$ can attain while still permitting refinement of $\underline{f}$.

Let us now assume that both the single- and multiple-source uncertainty models have the same static envelope for f and the same bounds for A at time t. Then the second terms on the right-hand sides of both Equation 4.2 and Equation 4.3 are identical. Since $\underline{c} \leq c$ we see that the value of $\overline{a}'$ that produces an improvement in $\underline{f}$ is *lower* for the multiple-source model than it is for the single source. This means that values of $\overline{a}'$ that satisfy $\underline{c} - \underline{o}_f < \overline{a}' < c - \underline{o}_f$ will refine the single-source model but not the multiple source model. This is contrary to the intuition that models with larger uncertainty should be refined by measurements that can refine models with smaller uncertainty and suggests that our refinement operation is overly conservative.

Collapsing Multiple-source Uncertainty The problem with multiple uncertainty sources is that they are not complementary – uncertainty in one source leads to further conservatism in another. One approach to eliminating this problem is to collapse the uncertainty into a single source. For our example, consider rewriting the SDE as

$$A' = -N_f \qquad\qquad N_f = h(A)$$

where $h(A) = f(A) - C$, $h \in M^+$ and $\underline{f}(A) - \overline{c} \leq h(A) \leq \overline{f}(A) - \underline{c}$. This model replaces uncertainty in C and f with a single function h which contains all the uncertainty. For this model, the relevant Q2 equations for $\underline{N}_f$ are:

$$\underline{N}_f = -\overline{A}' \tag{4.4}$$

$$\underline{N}_f = \underline{h}(\underline{A}) \tag{4.5}$$

$$\tag{4.6}$$

For an observation $[\underline{a}', \overline{a}']$ of $A'(t)$ to improve h, we must have

$$\underline{n}_f < -\overline{a}'.$$

Substituting for $\underline{n}_f$ using Equation 4.5 gives

$$\underline{h}(\underline{a}) = -\overline{a}'$$

and using the definition of h gives

$$\underline{f}(\underline{a}) - \overline{c} < -\overline{a}'.$$

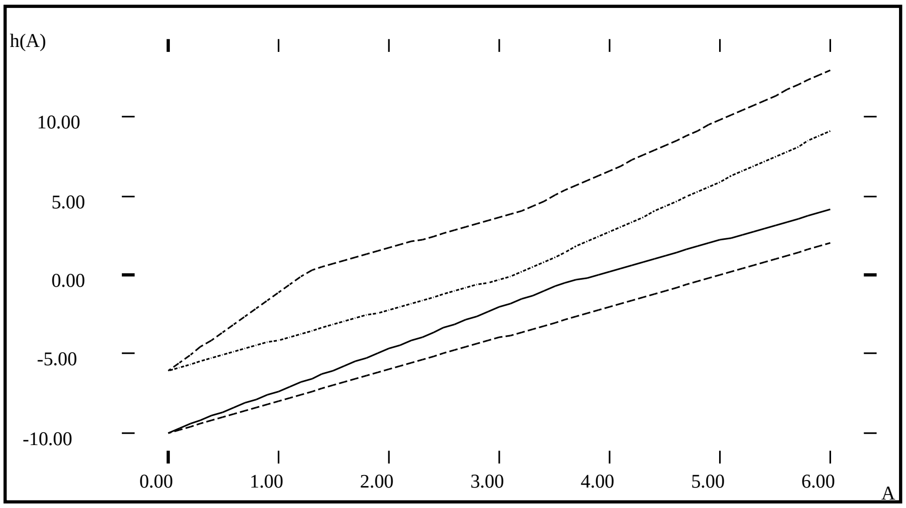
Finally, rearranging terms lead to

$$\overline{a}' < \overline{c} - \underline{f}(\underline{a}). \tag{4.7}$$

Let us assume that the single-, multiple-, and collapsed-uncertainty models have the same static envelope for f and the same bounds on $A(t)$. Then, since $\underline{c} \leq c \leq \overline{c}$ we see by comparing Equations 4.2, 4.3, and 4.7 that the collapsed uncertainty model requires the weakest bound on the upper envelope of A'. This is consistent with the expectation that greater model uncertainty requires less precision in measurement to improve the model. Figure 4.21 shows that this strategy leads to a greatly improved model and prediction.

The Advantages of Multiple-source Uncertainty Collapsing uncertainty sources is a useful method for improving model refinement, however, there is a cost. In the original multiple-source uncertainty model, note that the refinements of c and f are independent in that the conjunction of the statements $C \in [\underline{c}, \overline{c}]$ and $f(A(t)) \in [\underline{f}(A(t)), \overline{f}(A(t))]$ are true. As a result, we can extract the static envelope for f and use it in a different model with the assurance that the refined static envelope will be correct. This property is not true of traditional identification methods since *any* function that produces a satisfactory fit can be chosen. For instance, if we assume that the "true" model for $C - f(A)$ is $c_1 - f_1(A)$, the search may find the function $c_2 - f_2(A)$ where $c_2 = c_1 + d$ and $f_2(A) = f_1(A) + d$. In this case, f_2 is not a model for the true f.

By collapsing multiple-source uncertainty, we also remove the individual constraints on each individual source. This means that it is no longer possible to determine better bounds on these terms. As long as we are not interested in anything but the combination this is fine. However, if we do still care about the bounds on the individual sources, we could still include the constraints on the individual uncertainty sources. This would result in *redundant constraints* in the model which would ensure the best possible overall bound while still providing bounds on the individual sources.



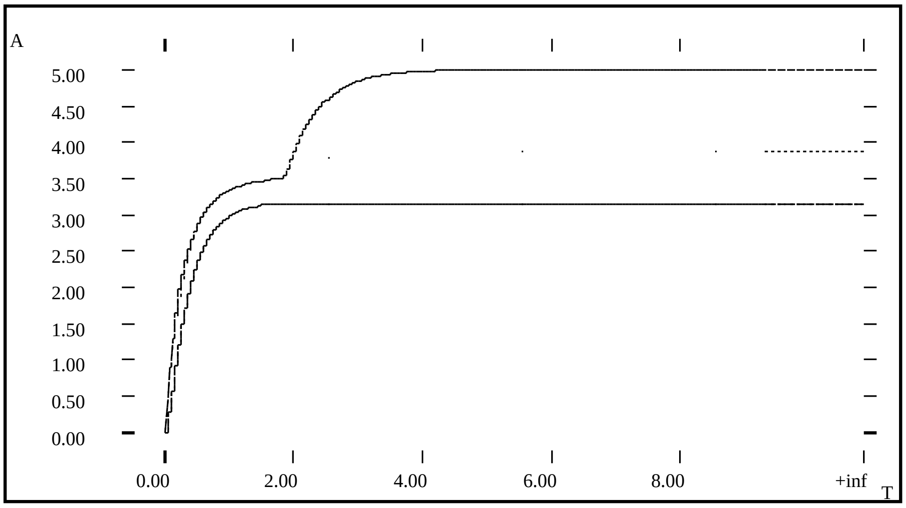


Figure 4.21: The effect of collapsing uncertainty sources. Using the single uncertainty source $h(A)$ rather than $f(A) - C$ leads to improvements in both the bound on $h(A)$ (upper graph) as well as the predicted trajectory whose upper equilibrium bound shrinks from ~ 5.0 to ~ 3.5 (lower graph).

4.3.5 The Effect of Observability

Observability is a measure of the degree to which the internals of the model can be seen. For precise models, observability determines whether the state of the system can be reconstructed from the measured variables. For imprecise models, observability also impacts the degree of refinement that can be achieved.

For our observability study, we examined the two-tank cascade with fixed inflow and ran SQUID on four cases where we measure B, A and B, B and $g(B)$, and A, B, $f(A)$, and $g(B)$. In each case, 100 measurements were generated from the model

$$\begin{aligned} A' &= 25 - 9\sqrt{A} \\ B' &= 9\sqrt{A} - 8\sqrt{B} \\ A(0) = 95 & \qquad B(0) = 0 \end{aligned}$$

with additive noise of variance 4 added to each measured variable. The SQDE is:

$$\begin{aligned} A' &= c - f(A) \\ B' &= f(A) - g(B) \end{aligned}$$

with $c \in [25, 25]$ and static envelopes for both f and g of $[2x, 15\sqrt{x}]$ for $x < 15$ and $[0.1x + 30, 15\sqrt{x}]$ for $x \geq 30$. Table 4.1 shows the results of this test and Figure 4.22 show the static envelopes for f and g when all four variables are measured. As more variables become observable, both static and dynamic envelopes improve. Note that while $f(A)$ improves in the presence of measurements for A, the same is not true of the relationship between B and $f(B)$. This is because the differential equation $B' = f(A) - g(B)$ includes two uncertainty sources whereas $A' = c - f(A)$ includes only one.

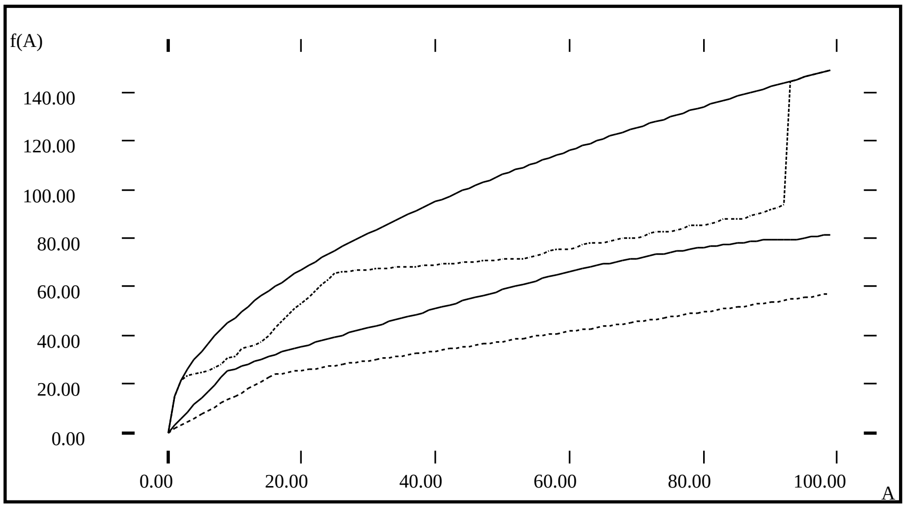
These results demonstrate that the best refinement is obtained when all uncertainty sources are measured. For parameters, this means directly measuring the parameter. For static envelopes, this means measuring both the domain and range of the function.

4.3.6 SQUID as Applied to Monitoring

SQUID has been described as a method for improving monitoring applications. In this section, we examine the behavior of SQUID on such problems. We focus on two tasks of importance in monitoring – detection of model-data mismatch and detection of model drift.

We begin by examining the behavior of SQUID on the model-data mismatch problem. Recall that we can break monitoring into two phases – selecting an appropriate structural model and tracking the selected model. Considerable reduction of computation is

[12]The absolute envelope widths for no measurements are 6357, 3764, 782, and 2772 when measured over $A \in [0, 100]$, $B \in [0, 70]$, and $t \in [0, 40]$ for A and B.



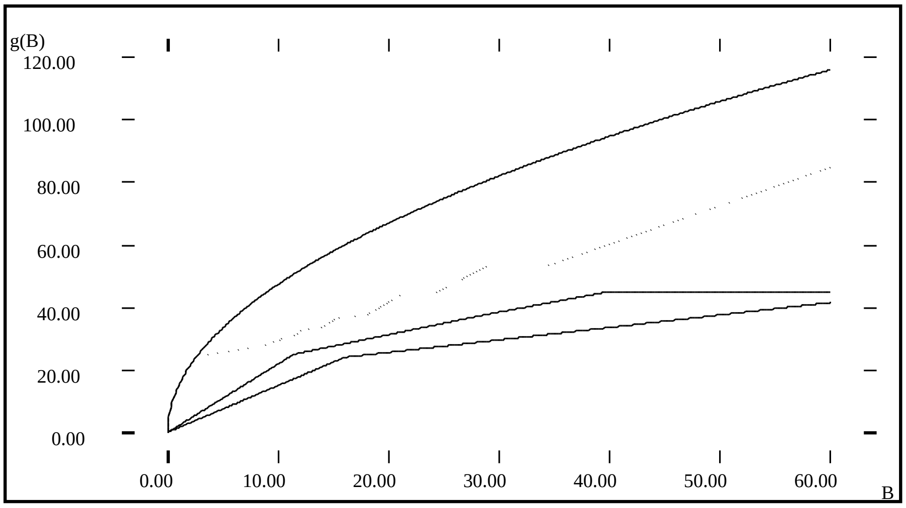


Figure 4.22: Refinement of f and g in the two-tank cascade when A, B, $f(A)$, and $g(B)$ are measured. The outer envelopes are the initial ones provided by the SQDE.

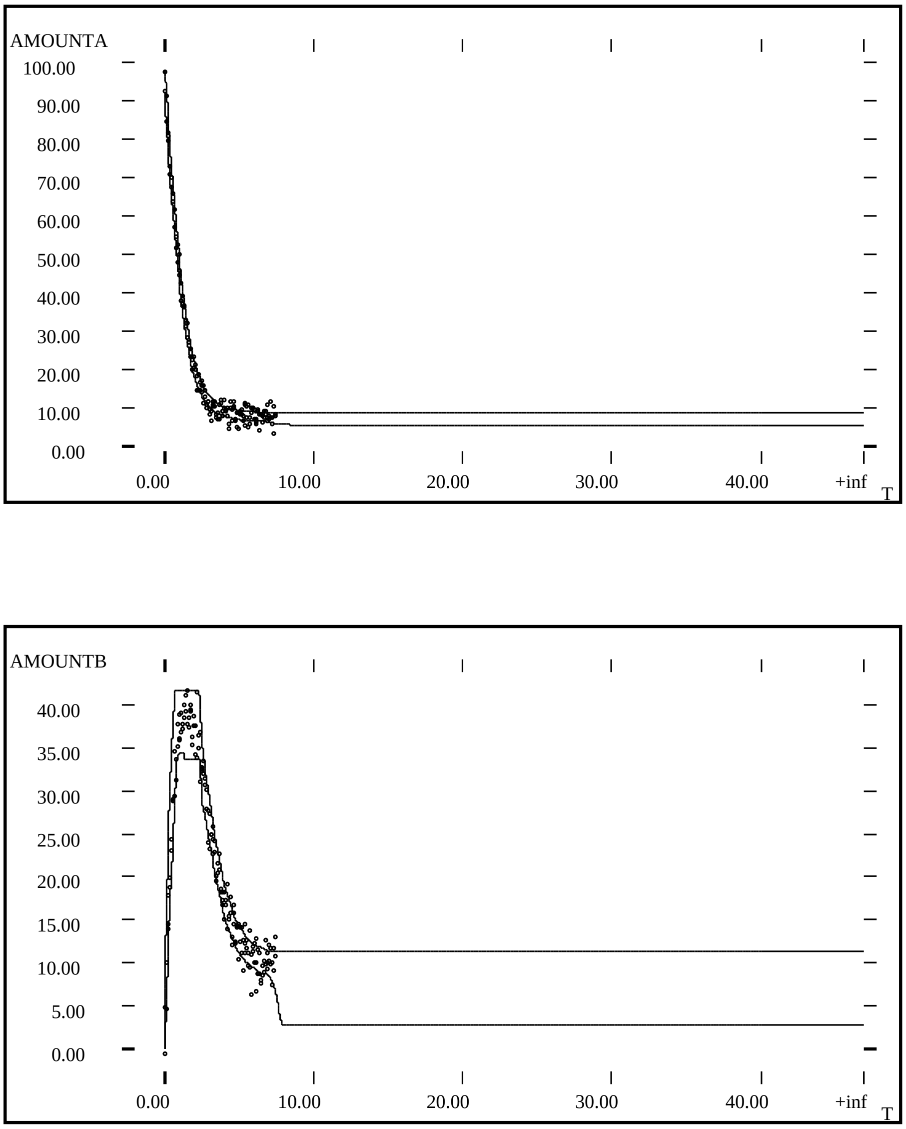


Figure 4.23: Dynamic envelopes for A and B together with measurements for the two-tank cascade using the envelopes in Figure 4.22.

Measured Variables	Envelope area ratio			
	$f(A)$	$g(B)$	$A(t)$	$B(t)$
None	1.00	1.00	1.00	1.00
B	1.00	0.97	1.00	0.13
A and B	0.51	0.97	0.28	0.13
B and $g(B)$	1.00	0.50	1.00	0.13
A, B, $f(A)$, and $g(B)$	0.28	0.50	0.24	0.12

Table 4.1: The effect of observability in the two-tank cascade. Each entry represents the ratio of the width of the envelope when selected measurements are made to the width with no measurements. Envelope width is defined to be the integral of the difference between the upper and lower bounds over the domain of interest[12]. As more variables become observable, both static and dynamic envelopes improve.

possible if we can refute an incorrect structural model quickly since this reduces the numerical computation required. As an example, consider the case where we have selected a second-order model to monitor a data-stream, but in reality, our data comes from a first-order model. Figure 4.24 shows the result of refining the gravity model using a data-stream from the single-tank model $A' = 50 - \frac{50}{110}A$ with variance in A of 25. At up to 60 points, SQUID determines that the gravity model could correspond to the given data. Implicit is the assumption that the maximum value has not been reached. With the addition of 20 more points, however, the qualitative filter determines that the trend has no maximum before $t = 6$ and so the model is refuted.

A more subtle case of model mismatch occurs when the model is structurally correct, but the true system does not lie within the model space because the parameters and/or monotonic functions of the true system lie outside the bounds and static envelopes of the SQDE. Consider a two-tank cascade with $c \in [22.5, 27.5]$ which represents $\pm 10\%$ error on the nominal value $c = 25$ and bounds on f and g in the range $[8\sqrt{x}, 10\sqrt{x}]$. We wish to examine two separate possibilities:

1. At what point does an error in c cause refutation?

2. At what point does an error in g cause refutation?

We use a data-stream consisting of measurements for A and B with a variance of 4 from the nominal system where $c = 25$, $f(A) = 9\sqrt{A}$, and $g(B) = 8\sqrt{B}$.

In the first case, we find that the model is refuted with a 100 point data set when the true inflow is outside the range $15 \leq c \leq 35$. For values of c between 15 and 18, refinement greatly reduces the initial bounds on c while for other values, the initial ranges hold. For the second case, we vary the true $g(B) = k\sqrt{B}$ by varying k. SQUID refutes the model for $k \leq 6$.

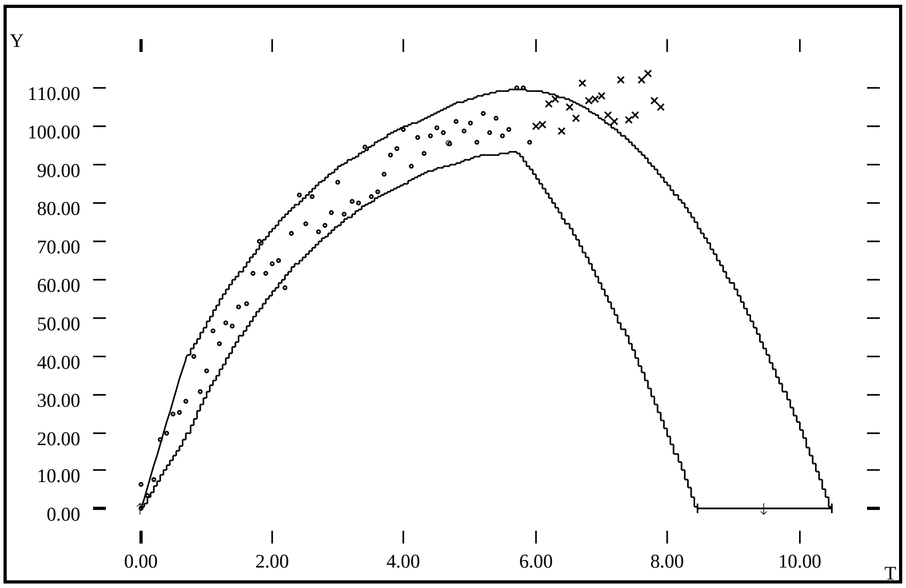


Figure 4.24: Structural mismatch between data and model. At up to 60 points (dots), SQUID finds a portion of the initial model space that is consistent with the measurements. After 80 points (crosses), the model is refuted.

These examples demonstrate that SQUID can detect both structural and non-structural mismatches between a model and a measurement stream. Structural mismatches are easier to detect, however, since they normally exhibit a qualitative trend description that differs from the qualitative behavior description of the model. Since only the qualitative filter is required to detect this difference, structural mismatches are normally caught before the more expensive refinement operation is performed.

As a final example, we examine the effect of drifting faults in a CSTR model. We begin with the CSTR model

$$\begin{aligned} \frac{dC_A}{dt} &= \frac{C_{Ai} - C_A}{\tau} - k_0 e^{-E/T} C_A \\ \frac{dT}{dt} &= \frac{T_i - T}{\tau} - h_r k_0 e^{-E/T} C_A \end{aligned}$$

with $\pm 5\%$ in uncertainty in the nominal value $h_r = -200$ and starting at steady-state 1 ($C_A = 0.933$, $T = 353.36$). Section 2.3 gives a detailed discussion of this model and its behaviors when simulated from this state. We begin monitoring C_A using measurements with variance 0.0001 from a model starting at steady-state 1. At $t = 25$, we introduce a gradual fault in the input temperature set-point T_i such that $T_i(t) = 340 + \frac{20}{75}(t - 25)$. As the set-point shifts, SQUID refines the model as shown in Figure 4.25. Eventually, the data no longer matches the behavior space of the model and the model is refuted. Note that in this case refinement is not necessary since the original lower bound is sufficient to detect a discrepancy. Thus SQUID could have simply run its mapping component, skipping refinement without any loss of diagnostic capability. Note however, a more complex analysis shows that the improvement in the lower bound has now caused more points in the region $t \in [20, 40]$ to fall outside dynamic envelope. This could be used as a signal that the model is shifting (since normally one would expect old measurements to remain in envelope). Thus, refinement could be used to detect the discrepancy earlier.

4.4 Analysis

In this section, we summarize the performance results for SQUID and provide some analysis of the complexity of the method with an eye toward improving its efficiency.

4.4.1 Performance

Comparison to Traditional System Identification

Since we are evaluating SQUID in comparison with traditional system identification, it is important to understand the situations in which one method is better than another. We can define several types of identification problem and see how each method does on them. We look at the following:

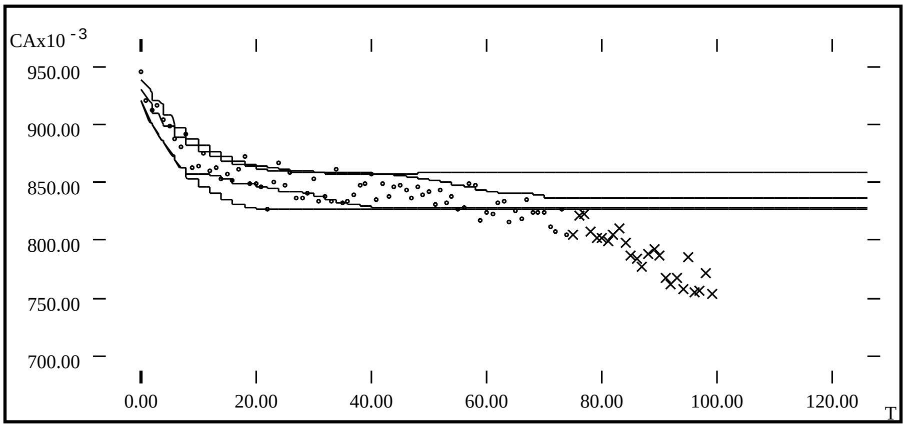


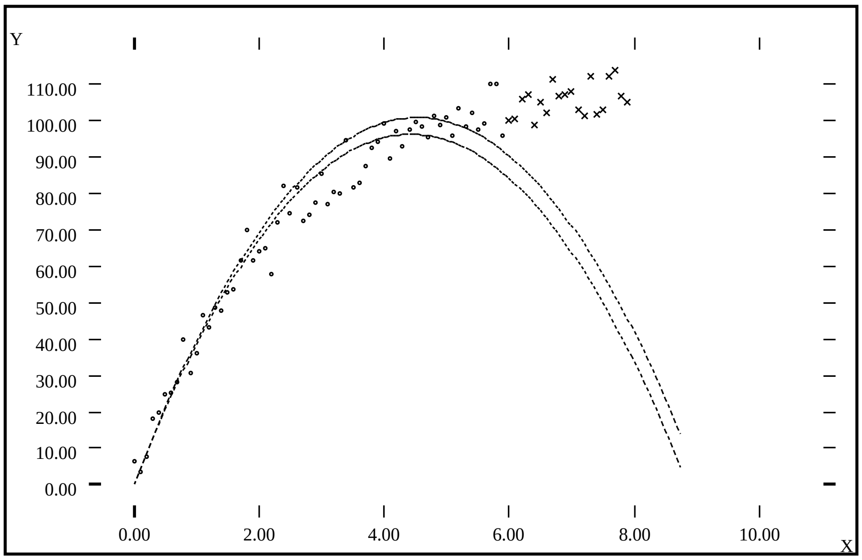
Figure 4.25: Refuting a drifting fault in a CSTR. The inner dynamic envelope is generated from data up to $t = 70$. The outer envelope is generated from the original SQDE. Note that the model is clearly in error from $t = 70$ onward.

- No functional uncertainty, small number of parameters.
  In this case, we have a precise functional form and the search space is small. This is exactly the situation that traditional identification excels at and it produces a better refined model space (see Figure 4.26). SQUID can refine its initial model, however the ultimate refinement represents a much larger segment of the model space.

- No functional uncertainty, large number of parameters.
  In this case, while the model is precise, the parameter space is large, making gradient-based search difficult. Searching the parameter space is difficult and traditional identification may fail to converge. Since SQUID does not search the model space, it is unaffected by the size of the parameter space and can still produce a refinement.

- Functional uncertainty.
  Since traditional identification cannot express imprecise functional models, it must approximate them using a highly parameterized model. This leads to the previous case.

- Uninformative data.
  In this case, traditional identification may be led astray by reducing the model space too far because it is unable to get an accurate measure of the underlying dynamics. In contrast, since SQUID eliminates inconsistent portions of a model space, uninformative data will prevent any further elimination from taking place.



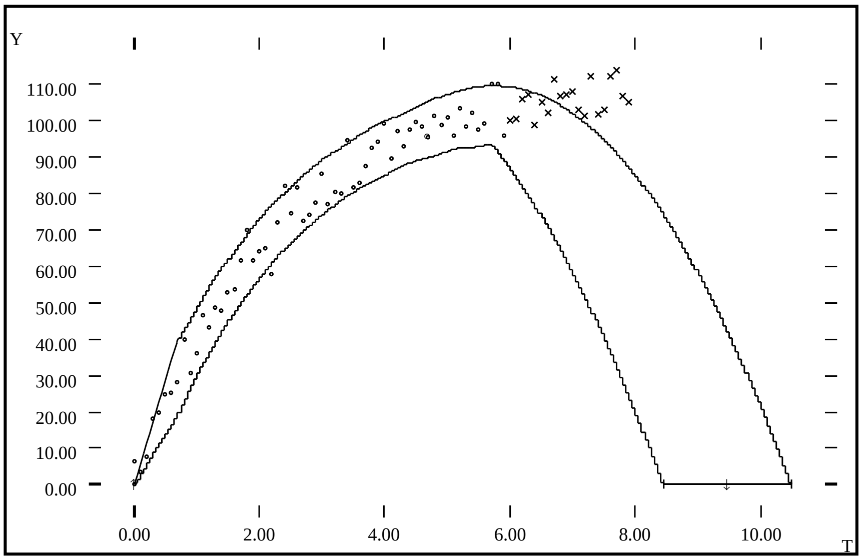


Figure 4.26: Comparison of the dynamic envelopes from traditional identification (top) and SQUID (bottom) for the gravity model given inconsistent data. Traditional identification fares much better in the case of functionally precise models with small parameter spaces (in this case, one dimensional), illustrated here by less overlap with the data.

Factors Affecting Refinement

As has been discussed in Section 4.3, there are a number of factors that affect SQUID's ability to refine models. We summarize these factors here.

- Model uncertainty vs. measurement noise.
  The uncertainty present in the SQDE determines the maximum amount of noise tolerable in the measurements. A model with large amounts of uncertainty is thus able to make more use of a noisy dataset than can a model with small amounts of uncertainty.

- Multiple Sources of Uncertainty.
  When multiple uncertainty sources exist, measurement bounds must be *tighter* than in the single-source case for refinement to take place. The solution to this problem is to collapse uncertainty sources. By taking the collapsing method to its limit, we can reduce each equation in the SDE to

  $$x_i' = f_i(\mathbf{x})$$

  thereby creating a single uncertainty-source for each equation. The disadvantage in this approach is that individual uncertainty sources are no longer distinct.

- The set of observable variables.
  The degree to which the trajectory space can be measured is determined by the observability of the model. In particular, SQUID does best when all imprecisely known constants and state variables as well as both sides of all monotonic functions are measured.

- Prediction imprecision.
  As stated in Chapter 2, there are some models for which SQSIM produces numerous spurious behaviors. Since SQUID relies on SQSIM to determine the trajectory space associated with an SQDE, if this trajectory space is large then SQUID must work harder to reduce the trajectory space. This can reduce both the effectiveness and efficiency of refinement.

4.4.2 Complexity

This section analyzes the time complexity of SQUID with emphasis on identifying sources of inefficiency and strategies for removing them. Bearing in mind that worst-case complexity analyses (especially of prototype implementations) have little bearing on "normal" cases, we describe both the theoretical and the empirical complexity of the key components of SQUID to show which operations are most efficient. There are four major components of SQUID:

Simulating the initial SQDE with SQSIM **(Section 2.2)** SQSIM is composed of the QSIM, Q2, and NSIM simulators. The complexity of a simulation is dependent on the form of the SQDE and the length of the longest behavior in the behavior tree. Let c be the number of constraints in the SQDE, s the number of state variables (i.e., variables with a `d/dt` constraint), and t the length of the longest behavior.

- The cost of generating a QSIM behavior tree is worst-case exponential in t (due to branching caused by multiple successors to a qualitative state) although the empirical cost is closer to $o(ct)$ [Kuipers, 1986][13].
- The major costs of using Q2 on a single state are:
  - Equation generation.
    Equation generation creates the set of equations that hold for the events determined by a qualitative time-point state. For each variable in a constraint, Q2 generates one equation with the event rmag[14] on the left-hand side. Since all constraints have at most three arguments[15], this results in $o(c)$ equations. Each derivative constraint, however, must check for previous states for which it can apply special constraints holding over regions of the behavior where the qdir of the curvature of a state variable is constant. Thus, equation generation takes $o(c + st)$.
  - Equation evaluation.
    At the start of Q2, all newly-generated equations are placed on the evaluation agenda. Then, whenever an equation changes the range associated with a particular rmag, all other equations (including those generated at previous qualitative time-points) using this rmag in the right-hand side are added to the agenda (unless they are already present on the agenda). This propagation of changes to other equations is called *following*. The process continues until no further improvement can be made to any range. Since each state generates $o(c + st)$ new equations, there are $o(ct + st^2)$ equations that can be considered in a behavior of length t. Since the cost of evaluating an equation can be bounded by a constant, the time complexity of evaluating each equation once is $o(ct + st^2)$. The cost of following rmag changes is determined in part by the parameter `*epsilon*` that determines when the change to the range of an rmag is significant. In addition, there is a contraction property that guarantees that changes in each range will

[13] We say a function $f(n)$ is $o(g(n))$ if there exists a constant c such that $|f(n)| \leq |cg(n)|$ for all $n \geq 0$.

[14] An *rmag* represents the precise value that a variable attains at a given time. (i.e., in the event $x(t_1) = x_1$, x_1 is an rmag of x).

[15] QSIM3.0 allows multi-variate monotonic constraints which take an arbitrary number of arguments, however SQUID does not make use of these constraints.

always shrink monotonically until they drop below `*epsilon*` [Kuipers, 1994]. Assume that k bounds the number of times that the range of an rmag can change before reaching `*epsilon*`. Then in the worst case, each equation will appear on the agenda k times leading to a total evaluation time of $o(k(ct + st^2))$.

– The cost of NSIM simulation depends on the ODE solver used. For a fixed-step method, the cost of a derivative evaluation depends on the complexity of the SQDE which is $o(c)$ and the derivative must be evaluated at each simulation step. Thus, the complexity of NSIM is $o(Nc)$ where N is the number of simulation steps.

Empirically, the cost of producing the SQ behavior using SQSIM can be considerable. For the CSTR, it is the dominating cost of running SQUID. For the other models, it is less expensive. Note, however, that once the SQ behavior is generated, SQSIM need not be run again. Thus, pre-simulating the SQDE is a viable method for reducing the cost of this step.

The branching factor of the simulation is also a key cost in the mapping and refinement steps of SQUID since these steps must operate on a per-behavior basis. Thus, producing a simulation that produces a small behavior tree is very important to reducing the cost of identification.

Trend Extraction (Section 4.2.3) Trend extraction consists of determining the qualitative, event, and dynamic envelope descriptions of a measurement stream. Each measured variable must have its trend extracted, so the cost is linear in the number of variables. The complexity of this step depends upon the cost of the generating each description component.

- Binning extracts the qualitative trend description. For a fixed-size window, the cost of binning increases linearly with respect to the number of measurements. The computation of the slope of a bin requires $3w$ multiplications where w is the size of the binning window.

- Event extraction is linear in the number of $\ominus$ bins.

- Dynamic envelope extraction involves fitting a monotonic function to the bin using MSQUID and computing the static envelope function for the fit. Computing the static envelope function given the estimate is $o(n_h^3)$ due to the QR factorization required to linearize the expectation surface (see Section 3.3). Computing the estimate depends on the optimization method. The OPT method [Biegler and Cuthrell, 1985] breaks the optimization problem into selecting a direction in the problem space by solving a quadratic programming problem and then using a line search method for selecting the size of the step in this direction. Convergence of OPT is theoretically super-linear, and each step requires the computation of the objective function and gradient. Of these two costs, the cost of computing the objective function dominates and is $o(n_h n_s)$ where n_h is the number of hidden units and n_s is the number of samples to fit.

Trend extraction is comparatively inexpensive because (1) it is purely a function of the measurement stream and is thus unaffected by the number of SQ behaviors, and (2) because binning and dynamic envelope extraction on a memoryless function are comparatively inexpensive operations.

Mapping the trend to the behavior (Section 4.2.4) Mapping correlates the trend and behavior descriptions of the system on a per-behavior basis. For each behavior, the following costs are incurred:

- Generating a single qualitative mapping for an SQ trend with b bins requires sequentially matching each bin to a state in the SQ behavior. The worst-case cost is thus $o(bt)$, however, as b gets longer, the matching process is faster since a full search of the SQ behavior becomes less likely. In cases where $b > t$ there is no mapping and so the time complexity is only $o(b)$ since this is what is needed to determine the length of the trend. Whenever a qualitative mapping fails (due to a failure in event mapping, dynamic envelope mapping, or refinement), a new one must be generated, if possible. This increases the worst-case cost of finding *all* mappings to $o(bt^2)$.
- Event mapping depends linearly on the number of trend events and then on the cost of running Q2 on these events.
- Dynamic envelope mapping cost depends linearly on the number of time-points in the behavioral dynamic envelope.

In addition, the cost of re-simulation once mapping is completed depends linearly on the number of time-points in the dynamic envelope.

Mapping can be expensive because multiple mappings may need to be generated. Qualitative and event mapping, however, are comparatively inexpensive operations, so as long as they can eliminate non-matching behaviors, the cost of mapping can be kept low.

Refinement (Section 4.2.5) Refinement consists of :

- Computing derivative bounds for each state variable at each refinement point. This computation is done by scanning the dynamic envelopes of each state variable and computing bounds from the trend. For each state variable, this process is linear in the length of the monotonic region containing the refinement point of the dynamic envelope.

- Generating a refinement state and running Q2 on it. Since the refinement state is completely separate from other states, this cost is $o(kc)$ (rather than $o(k(ct + dt^2))$ since neither equations from previous states nor derivative equations are generated.

These tasks are repeated for each refinement snapshot (a user-specified quantity).

Refinement is also an expensive operation since dynamic envelopes can contain many points within a monotonic region. One trade-off that could be made is to reduce the number of snapshots taken which reduces the cost of refinement at the expense of producing a smaller improvement to the model. For monitoring applications where refinement is perhaps not as important as discrepancy detection, it may be possible to ignore the refinement step completely (or perform it off-line).

Given this analysis of the factors affecting the complexity of SQUID, we rank the different subsystems in order of decreasing complexity.

1. The initial semiquantitative simulation of the SQDE has the highest complexity. By pre-simulating the SQDE, however, this step can be performed "off-line." Furthermore, since the other components of SQUID operate on a per-behavior basis, the number of behaviors generated by SQSIM has a large impact on the cost of using SQUID.

2. The cost of refinement is next most expensive since it must search for derivative bounds. By reducing the number of snapshots, SQUID would be much faster, however the ability to improve the SQDE is reduced.

3. Mapping is the next highest cost due to the generation of multiple qualitative mappings for each SQ behavior. For each such mapping, however, the cost of event and dynamic envelope mapping is comparatively small.

4. Trend extraction is the least expensive component of SQUID.

4.5 Related Work

In addition to traditional system identification, SQUID can also be compared to other systems developed in the AI community for trend detection, monitoring, and identification.

4.5.1 Trend Detection in Noisy Data

Detecting a trend in the presence of noisy data is the topic of filter theory. One limitation of traditional methods is that it is often necessary to have extra information about the underlying trend such as its power spectrum to define a filter that reconstructs the original signal. Scale-space filtering [Witkin, 1983] eliminates this need by filtering the signal through a family of Gaussian filters whose filter coefficient is continuously varied across a range of values. As the coefficient increases, the signal is smoothed further until it eventually becomes flat. If we plot the inflection points of these filtered data-streams and graph them with time as the abscissa and filter coefficient (or *scale*) as the ordinate, we obtain the *scale space* of the original signal. Curves in scale-space represent the occurrence and disappearance of critical points in the signal as the scale increases. The height of the critical point on each of these curves is the scale at which the underlying inflection point is completely smoothed out. By analyzing the differences in heights of these critical points, the time-scale of the signal over different time intervals can be determined and an appropriate fitting function can be selected to reconstruct the signal at that scale.

Scale-space filtering permits the recovery of a trend with no prior knowledge. Unfortunately, it does so at the cost of filtering the signal through a theoretically infinite number of Gaussian filters. Work has been done on reducing this need [Kulkarni *et al.*, 1991], but it still involves multiple filterings of the data as well as pattern recognition to infer the scale-space portrait from a finite set of points.

In part as a solution to the computational expense of scale-space filtering, Cheung and Stephanopoulos developed the "triangle representation" for trend description together with an algorithm for extracting a trend from noisy data [Cheung and Stephanopoulos, 1990] . By defining a trend in terms of a primitive triangle component that captures the qualitative first and second derivatives of the trend and by describing a method for combining triangles into higher-level constructs (such as trapezoids), they are able to construct a fast algorithm for constructing a scale-space and filtering a signal with it.

The SQUID binning algorithm can be seen as a variant on these trend detection techniques. Since SQUID requires only qualitative filtering (i.e., the location of regions of monotonicity in the signal), the actual filtering technique can be reduced to computing regions of constant slope in the signal. By using multiple-sized windows, it can detect slope changes at different time-scales[16].

4.5.2 Semiquantitative Monitoring and Identification

The use of semiquantitative models for monitoring was the basis for the MIMIC system [Dvorak and Kuipers, 1989]. MIMIC implements both the tracking and hypothesis generation phases of monitoring (see Figure 4.27). It uses the SQDE as a model space representation from which an SQ behavior set is generated by (an earlier version of) SQSIM. Each behavior in the set is then matched to the measurement stream. If a model is refuted,

[16]Conceivably, one could also use the SQUID binner to construct a *qualitative scale-space portrait*, although this is not necessary for SQUID.

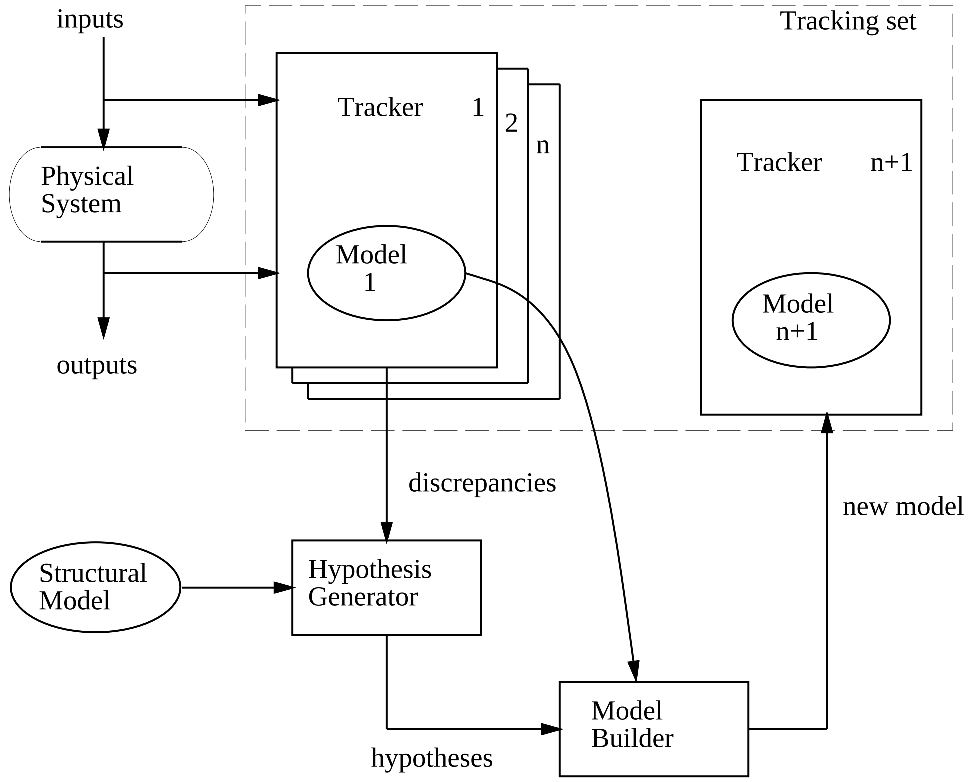


Figure 4.27: The MIMIC monitoring and diagnosis system.

MIMIC enters the hypothesis phase where it suggests new models to track based on the reason for the mismatch and knowledge of the structure of the device being monitored. MIMIC's strengths lie in the use of a robust prediction method (which guarantees that all possible behaviors of a an imprecise model are considered) and a hypothesis generator based on a structural model (which eliminates the need for pre-enumerating the set of possible fault models).

SQUID can be seen as an improvement to the tracking component of MIMIC. It adds a more realistic data model (MIMIC assumed that each measurement had a 100% confidence bound and that the derivative was given) together with a theory of semiquantitative trends which make SQUID more suitable for operating on real data-streams. Also, by shifting the focus from monitoring to identification, SQUID can produce better predictions by including model refinement in the tracking process. These improvements make SQUID more efficient and robust than the tracking method of MIMIC. Of course, SQUID does not address the hypothesis generation component of MIMIC and so these methods are very much complementary.

Another semiquantitative monitoring system is TrenDx [Haimowitz, 1994]. This system also uses a semiquantitative representation of behavior and attempts to fit data to the behavior. Unlike both SQUID and MIMIC, however, TrenDx does not use a model space representation. This has two consequences:

1. Since there is no model space, TrenDx cannot do refinement.

2. The user must generate the SQ behavior by hand.

By sacrificing a model space representation, TrenDx simplifies the tracking component of its monitoring method. This permits more efficient methods for matching (since the user can provide customized behavior segmentation and fitting methods) at the expense of greater sophistication on the part of the user. SQUID chooses to include a model space representation together with simulation to produce an SQ behavior set. While computationally more expensive, focusing on a model allows the user to describe the structure of the process rather than exhaustively describing its behavior. Furthermore, a structural model-plus-simulator is more flexible than a behavioral model in that one can change the structural model and then predict the consequences. This is particularly important in monitoring where faults are manifested as structural changes.

Finally, another use of qualitative methods for identification is embodied in the system of [Capelo *et al.*, 1996]. This system addresses the problem in traditional identification of how to select the best parametric model from a set of potential models. By using the qualitative properties of different parametric models and comparing them to the properties of the measurements, this system can eliminate from consideration those parameterizations that are inconsistent. The method does a form of qualitative trend extraction from the data which is then matched against the qualitative behavior of candidate models based on a larger set of primitives (concave, increasing, linear, etc.) but does not appear to address the problems of noise in the measurements.

Chapter 5

Discussion and Conclusions

5.1 Summary of Conceptual Results

This dissertation investigated methods for modeling, simulation, and refinement of imprecisely-defined systems. While much of the emphasis of this dissertation has been on the development and analysis of implementations of inference mechanisms, there are also several key concepts that have emerged from this research. These are:

- Knowledge about both models and predictions can be represented at a variety of levels from qualitative to quantitative. By selecting a level where a particular type of knowledge is precise, inference methods tailored to that level can provide powerful predictions.

- Both simulation and refinement can be viewed as operations that transform model spaces into trajectory spaces and vice versa. This view provides a way of examining the generation of spurious behaviors in a simulation method. It also clarifies the role of inverse simulation as a method for generating models (or more specifically, refinements of a model) from a trajectory space.

- Raw data can be represented as a semiquantitative trend. This allows significant savings in matching data to models, since the determination of qualitative properties of the trend is inexpensive and can rule out large portions of the trajectory space.

- Monitoring can be viewed as a form of identification of an imprecise model. By focusing on refinement, monitoring becomes more efficient since the predictions from a refined model are more specific than from the original model.

- Identification can be viewed as a refutation process rather than a search process. Since refutation is insensitive to search problems such as local maxima, it is easier to construct refutation-based identifiers that are robust in the face of high-dimensional parameter spaces and uninformative data.

5.2 The Limits of Semiquantitative Simulation

A key component of the work reported in this dissertation is the simulation of imprecisely-defined ODE systems. A central tenet of SQSIM is that the simulator should *guarantee* that all real behaviors of the SQDE are predicted. Unfortunately, this idea does not come without a cost. While qualitative simulation and interval propagation can provide such a guarantee, they also produce spurious behaviors.

If we examine each component of SQSIM we see a different source of spurious behavior and inefficiency.

- QSIM produces spurious behaviors due to ambiguity in qualitative arithmetic. Such ambiguity can result in intractably large behavior trees because unimportant qualitative distinctions cannot be eliminated by the constraint satisfaction algorithm.

- Q2 produces conservative predictions because interval propagation ignores correlations between variables (e.g., $X - X \neq 0$ when X is a non-zero width interval). It is inefficient because the propagation simply follows changes until a fixed-point is reached. Thus, constraints are not applied in an optimal manner which may result in loops.

- NSIM produces conservative predictions because it, too, is based on the principles of interval mathematics.

The key insight of SQSIM is that these sources of conservatism are in some sense orthogonal so the results of each type of simulation can reduce ambiguity in other parts. Unfortunately, in some cases this is still not enough to generate a useful prediction (see the discussion of the CSTR in Section 2.3.1).

Since SQSIM can only predict as well as its underlying methods (qualitative constraint satisfaction and interval arithmetic) allow, it makes sense to ask how one might improve these methods:

Improve the Simulation Methods Over the years, the QSIM algorithm has undergone continual improvement. In addition to the semiquantitative improvements described in this dissertation, other research related to taming intractably branching behaviors has been conducted. In particular, work by Clancy [Clancy and Kuipers, 1993] on elimination of uninteresting distinctions, behavior abstraction, and focusing techniques show promise in reducing the number of spurious behaviors in a qualitative simulation.

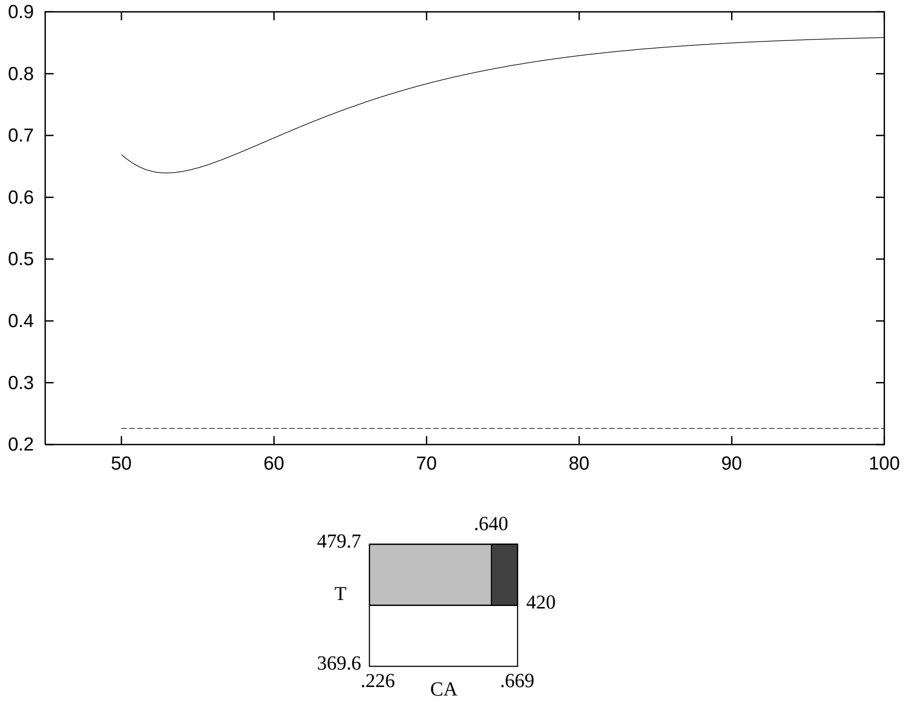


Figure 5.1: Using subdivision to reduce overly-wide dynamic envelopes. At $t = 50$, SQSIM predicts the state space of the CSTR will be within the rectangular region shown in the bottom graph. By running NSIM from an initial state defined by the upper block (gray+black region), we get the predicted dynamic envelope for CA shown in the top graph. Since CA is known to be increasing, we can rule out the black region $CA \in [.640, .669], T \in [420, 497.7]$ since any trajectory starting there must decrease.

Similarly, as described in Chapter 2, there have been a variety of methods developed for reducing spurious behavior in interval simulation of ODE systems. Much of this work, including methods for reducing the effect of ignored correlations would improve the predictions of NSIM. It is also possible to tackle overly wide dynamic envelopes by using search to rule out sections of the trajectory space that are inconsistent with the trajectory and event information (see Figure 5.1).

Apply More Knowledge Another way to tighten predictions is to provide more information to the simulation engine. Of particular interest here is work on constraining the qualitative behavior by way of trajectory descriptions [Brajnik and Clancy, 1996]. This technique can also be used to provide a method for specifying the behavior of exogenous variables in a qualitative simulation.

Eliminate the Coverage Guarantee Another approach is to simply eliminate the guaranteed coverage offered by SQSIM. Recent work on sampling a monotonic function space for use with Monte Carlo simulation [Gazi *et al.*, 1996] suggests that such a method could replace the NSIM portion of SQSIM.

Simplify the Model Space Representation The main representational ingredient that distinguishes SQDEs from parametric models is the inclusion of static envelopes around monotonic functions. While extremely useful for providing a method for abstracting families of functions, they are also very difficult to compute with. By eliminating such descriptions and sticking to purely parameterized models, existing methods for predicting behavior become available [Lohner, 1988, Danès *et al.*, 1993].

It seems likely that for some domains, these types of solutions may overcome the existing limitations of SQSIM. Categorizing the situations in which they may apply is the subject of future work.

5.3 Future Work in Semiquantitative Refinement

This dissertation has developed a prototype for the refinement of semiquantitative models. Using this work as a starting point, we envision a number of fruitful directions for future research.

Improving SQUID SQUID's efficiency is a function of the efficiency of its components. As outlined in the previous section, the SQSIM simulator could be improved to produce better predictions. This would directly improve SQUID's ability to refine models since using inverse simulation on a smaller trajectory space will lead to a smaller model space.

At another level, improvements could be made to the MSQUID method. The most obvious course of action would be to expand the method to multivariate functions. The underlying mathematics already assumes such a framework, so all that would remain is encoding and testing. Identification of multivariate monotonic functions would expand the class of SQDE relations that SQUID could refine.

There are also improvements that could be made to SQUID itself. Primary methods would be to make the algorithm recursive (in other words incremental). Currently, SQUID acts on the entire measurement stream as a batch. Adding new points to the stream requires re-computation over the entire new dataset. Ideally, we would like to make this incremental.

Time-varying systems Processes whose inputs vary with time make up an important class of systems studied by system identification. These types of systems were excluded from our original design due to limitations in the QSIM modeling and simulation method. Recently, techniques for adding such properties to an SQDE have been developed [Brajnik and Clancy, 1996] and should be included in SQUID.

More complex discrepancy detection SQUID uses a relatively simple model of discrepancy detection where the overlap between the SQ trend and the SQ behavior must be empty to refute the model. A more complex method could take into account the *degree* of overlap. This type of measure would be especially useful when comparing the trend and behavior over time since one would assume that as more data becomes available, the refinement should still adequately explain older measurements. If, however, the process is drifting over time, SQUID could determine that the degree of overlap in some region of the behavior has reduced over time, thereby signaling a possible drift.

Coupling with traditional identification SQUID and traditional identification methods share the common task of finding the best description of a process given a model space. They differ in that SQUID uses refutation of regions of the model space while traditional methods use search. Search is expensive over high dimensional model spaces, while refutation is not. An interesting area for future research would be to investigate the use of SQUID-style refinement as a guide to traditional system identification methods. By first reducing the search space with SQUID, the cost of gradient-based search could potentially be reduced.

Adding control actions Adding the ability to control inputs as well as to measure outputs is a necessary component for SQUID to be able to solve "black box" problems such as those described at the beginning of Chapter 4. Recall that the task in this case is to experiment with the device by changing the inputs and observing the outputs in order to identify the system. Such an extension would permit SQUID to encompass the experiment design component of system identification. Control of the underlying process can also be used to reduce the ambiguity inherent in simulating imprecisely-defined systems since the proper experiment can help eliminate possible hypotheses.

5.4 Application Areas

SQsim, MSQUID, and SQUID each have capabilities that can be exploited in applications outside of pure system identification. In this section, we discuss some of these related application areas and what it would take to modify the existing components to work in that particular domain.

5.4.1 SQsim

One promising area for the use of semiquantitative simulation is in the design of physical processes. Since robust design must account for uncertainty introduced in the construction process, SQUID would appear to be an ideal method to apply to such problems. Limited work has already been conducted [Gazi *et al.*, 1993] with encouraging results. Future progress requires further work on reducing the number of spurious behaviors generated by the simulator, a technical problem that is the subject of active research in the Qualitative Reasoning community (see [Clancy and Kuipers, 1993] and [Tokuda, 1996], for example).

5.4.2 MSQUID

Function learning is a key task in many scientific endeavors. Often, however, a precise parametric model does not exist, yet there is a need to determine a best fit to a set of data. In such cases, there is often weaker information about the function in the form of qualitative relations between the dependent and independent variables. Using MSQUID, this information can be enough to learn a best fit to the data. As a particular example, consider the problem of fitting a set of data that relates the level of water in a stream to its flow. Such a relationship is of great importance in watershed management since it gives engineers a method to gauge flow into a reservoir from a measurement of stream level. An exact parametric model is unknown since this model would depend on the shape of the stream-bed which is difficult to determine. On the other hand, it is known that level and flow are monotonically related. MSQUID is thus an ideal method for generating such fits from historical data about a stream-bed. Such curves could also be used to detect changes in the shape of the stream over time since this would affect the level-versus-flow profile. MSQUID has been applied to such a problem with promising results.

5.4.3 SQUID

As discussed in Chapter 4, SQUID can be viewed as a method for tracking a behavior and detecting discrepancies in the context of monitoring and diagnosis. An extension of this idea is to apply a combination of SQUID and MIMIC to the task of self-calibrating systems. Here, a system would start out with an initial very general model of the process to be monitored which would provide weak, but guaranteed bounds on the behavior. As time passes and the system gains more experience with the system, it would learn a better model of its behavior that would in turn make it easier to detect when the system fails or drifts from its nominal behavior.

5.5 Conclusions

This dissertation has described work in combining qualitative and quantitative reasoning methods. We have shown that such a combination can lead to robust methods for prediction and data interpretation tasks, especially in engineered systems.

This work can be set in a broader context as well. Its roots started in the goal of automating common-sense reasoning, i.e., the use of symbolic methods for representing and reasoning about properties of continuous systems. By providing a link between these abstract representations and more conventional numerical representations, this work suggests how a computer can reason about real physical processes which are inexactly understood. By coupling a physical process with such a computational process, it should be possible to construct a system that can gain a better understanding of itself. This type of introspection will be a hallmark of future intelligent systems.

Appendix A

Proof that NSIM bounds all behaviors of an SQDE

Theorem 1 *Let $\mathcal{S}$ be an order-n SQDE with state vector $\mathbf{x} = [x_1, \ldots, x_n]$ and let the bounds on $\mathbf{x}$ be given by the vector $\mathbf{X} = [\underline{x}_1, \overline{x}_1, \ldots, \underline{x}_n, \overline{x}_n]$. Let A:$\mathbf{x}' = f(\mathbf{x})$ be any ODE model in the space of $\mathcal{S}$ and $\alpha : \mathbf{X}' = F(\mathbf{X})$ be the extremal system for $\mathcal{S}$ with initial condition $\forall_{i=1,n}\ \underline{x}_i(t_0) \leq x_i(t_0) \leq \overline{x}_i(t_0)$. Then $\forall_{i=1,n}\ \underline{x}_i(t) \leq x_i(t) \leq \overline{x}_i(t)$ for all $t \geq t_0$.*

Proof There are two parts to the proof. The first part shows that the extremal system produced by NSIM guarantees that

$$\underline{f}_i(\mathbf{X}) \leq f_i(\mathbf{x}) \leq \overline{f}_i(\mathbf{X}) \tag{A.1}$$

when

$$\forall_{j \neq i}\ \underline{x}_j \leq x_j \leq \overline{x}_j \text{ and } \underline{x}_i = x_i = \overline{x}_i \tag{A.2}$$

The second part shows that if this relation is true, then the bound follows.

Part One Let $x_i' = f_i(\mathbf{x})$ be the ith ODE in A. Referring to Table 2.1, we can see that for each of the "ground" expressions $e = c, x_i$, or x_j we have that $L(e) \leq e \leq U(e)$ whenever Condition A.2 holds. By induction on e, we can show that the other expressions in the table also obey $L(e) \leq e \leq U(e)$. Thus, letting $e = f_i$ we have $\underline{f}_i(\mathbf{X}) \leq f_i(\mathbf{x}) \leq \overline{f}_i(\mathbf{X})$ for each i.

Part Two Given that (A.1) and (A.2) hold, we now show that $\overline{x}_i(t) \geq x_i(t)$ for all $t \geq t_0$. The proof for the lower inequality is derived similarly. For the purposes of intuition, we prove a slightly weaker version of the theorem in which we assume $f_i(\mathbf{x}) < \overline{f}_i(\mathbf{X})$ when Condition A.2 holds. A complete proof which assumes $f_i(\mathbf{x}) \leq \overline{f}_i(\mathbf{X})$ is given in [Bothe, 1992] and [Hüllermeier, 1995].

The bound holds at t_0 by assumption. For any other time t, we have either

- $\overline{x}_i(t) > x_i(t)$ in which case the result holds trivially, or
- $\overline{x}_i(t) < x_i(t)$ in which case there must be some time $t_1 < t$ such that $\overline{x}_i(t_1) = x_i(t_1)$, or
- $\overline{x}_i(t) = x_i(t)$.

Assume therefore that at time t we have $\overline{x}_i(t) = x_i(t)$ and that $\forall_{j,\tau}\ \underline{x}_j(\tau) \leq x_j(\tau) \leq \overline{x}_j(\tau)$, i.e., time t is the first time that *any* state variable might cross its bounds (if there is some state variable x_h for which the inequality does not hold, we may then take t to be the time when x_h is equal to one of its bounds and let $i = h$). We seek to determine if $\overline{x}_i$ and x_i can cross at t.

Define the difference term $d_i(t)$ and compute its Taylor Series expansion:

$$\begin{aligned} d_i(t+h) &= \overline{x}_i(t+h) - x_i(t+h) && \text{(A.3)} \\ &= (\overline{x}_i(t) - x_i(t)) + h(\overline{f}_i(t) - f_i(t)) + \frac{h^2}{2}(\overline{f}'_i(\zeta) - f'_i(\phi)) && \text{(A.4)} \end{aligned}$$

where $\zeta, \phi \in [t, t+h]$.

By assumption, the first term of this expression is zero. Defining $\epsilon_i(t) = \overline{f}_i(t) - f_i(t)$ and rearranging terms gives

$$d_i(t+h) = h(\epsilon_i(t) + \frac{h}{2}(\overline{f}'_i(\zeta) - f'_i(\phi))) \tag{A.5}$$

Since this is the first time at which a state variable and bounds might cross, Condition A.2 must hold and so $\overline{f}_i(t) > f_i(t)$. Thus $\epsilon_i(t) = \overline{f}_i(t) - f_i(t) > 0$. Since $\epsilon_i(t) > 0$, $d_i(t+h)$ will be positive if $\epsilon_i(t) > \frac{h}{2}|\overline{f}'_i(\zeta) - f'_i(\phi)|$ which will be true for all h such that[1]

$$h < h_{min} = \frac{2\epsilon_i(t)}{|\overline{f}'_i(\zeta) - f'_i(\phi)|}$$

Thus, there is some finite interval $h \in [0, h_{min}]$ over which $\overline{x}(t+h)$ is non-decreasing, hence $\overline{x}_i$ cannot cross below x_i at t. Since this is the first point at which *any* bound could cross, we have shown that there is no time at which an upper bound trajectory could cross that of a member of $\mathcal{S}$ and so the bound must hold over all time.

[1]If $\overline{f}'_i(\zeta) - f'_i(\phi) = 0$ then $d_i(t+h) = h\epsilon_i(t) > 0$.

Appendix B

Proof That MSQUID Can Produce Any Monotonic Function

In this section, we show that the set of functions that MSQUID can represent includes the set of all continuous monotonic functions. For simplicity, we consider only the case of a one dimensional monotonic function (i.e., the function is $y = f(x)$ where x is scalar). The proof would follow directly from [Cybenko, 1989] except that, for reasons of computational efficiency, our constraint that guarantees monotonicity is overly strong. To see this, note that the derivative of y is

$$\begin{aligned} \frac{d\hat{y}}{dt} &= \sigma'(z + w_{o[n+1,1]}) \sum_{j=1}^{n_h} w_{o[j,1]} \sigma'(w_{i[1,j]} t + w_{i[2,j]}) w_{i[1,j]} \\ z &= \sum_{j=1}^{n_h} w_{o[j,1]} \sigma(w_{i[1,j]} t + w_{i[2,j]}) \end{aligned}$$

To ensure monotonicity, this *sum* must be greater than or equal to zero. To simplify the fitting process, however, MSQUID uses the constraint

$$\min_{\mathbf{w}} \sum_i (y_i - \hat{y}_i)^2$$

$$\text{subject to } \forall_{1 \leq j \leq n_h} w_{o[j,1]} \cdot w_{i[1,j]} \geq 0.$$

which requires only that each *term* of the equation must be greater than or equal to zero. Intuitively therefore, MSQUID requires that each sigmoid in the composition must be monotonically increasing which is not the case of the more general constraint.

For clarity, the proof below uses a slightly different definition of the sigmoid function and the network does not include a sigmoid at the final output of the network.

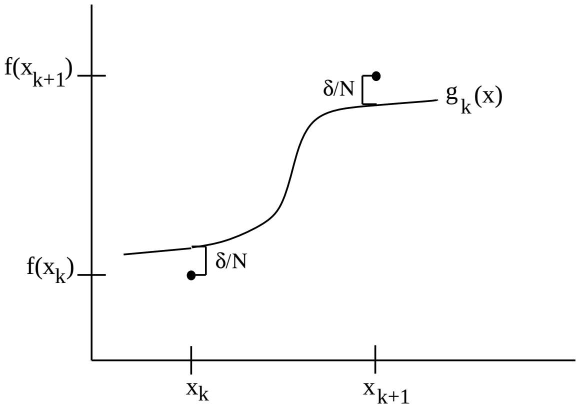


Figure B.1: A geometric view of the proof.

Theorem 2 *Let σ be a sigmoidal function such that $\sigma(x) \to 1$ as $x \to \infty$ and $\sigma(x) \to 0$ as $x \to -\infty$. Then given any $f(x)$ which is monotonic over some region I and $\epsilon > 0$ there is a finite sum of the form $y(x) = \sum_{j=1}^{N} a_j \sigma(b_j x + c_j)$ where $\forall_j a_j b_j \geq 0$ for which $|y(x) - f(x)| < \epsilon$ for all $x \in I$.*

Proof Let $\epsilon = \gamma + \delta$ where δ and γ are nonzero. Select N by partitioning I such that $\forall_{0 \leq k < N}\ f(x_{k+1}) - f(x_k) < \gamma$, i.e., f is subdivided so that there is no more than γ difference between any two contiguous points. We prove that MSQUID can still produce any continuous monotonic function as follows (refer to Figure B.1):

Assume that at x_0 the value of f is $f(x_0)$. Let $y_0(x) = f(x_0)$. Now let $d_k = f(x_{k+1}) - f(x_k)$. Define $g_k(x) = d_k \sigma(bx - (x_k + \frac{h_k}{2}))$ where b is chosen so that $g_k(x) \leq \frac{\delta}{N}$ for $x \leq x_k$ and $d_k - g_k(x) \leq \frac{\delta}{N}$ for $x \geq x_k$. Then define $y_{k+1}(x) = y_k(x) + g_k(x)$.

We wish to show that

$$|y_k(x) - f(x)| \leq k\frac{\delta}{N} + \gamma \tag{B.1}$$

for all $x \in [x_0, x_k]$. Clearly this is true for $k = 0$. For $k = 1$ we have

$$\begin{aligned} |y_1(x) - f(x)| &= |y_0(x) + g_0(x) - f(x)| \\ &= |y_0(x_0) - f(x_0) + g_0(x) + e_1(x)| \end{aligned}$$

where $e_k(x) \leq d_k$ for all $x \in [x_k, x_{k+1}]$. The first two terms inside the absolute value are identical while the third term is less than $\frac{\delta}{N}$ and the final term is less than γ. Thus

$$|y_1(x) - f(x)| \leq \frac{\delta}{N} + \gamma.$$

If we assume that Equation B.1 is true for k then for $k+1$ we have

$$|y_{k+1}(x) - f(x)| = |y_k(x) + g_k(x) - f(x)|.$$

We split this into two cases – $x \in [x_0, x_k]$ and $x \in [x_k, x_{k+1}]$. For $x \in [x_0, x_k]$ we have $g_k(x) \leq \frac{\delta}{N}$ and so

$$|y_k(x) + g_k(x) - f(x)| \leq k\frac{\delta}{N} + \gamma + \frac{\delta}{N} = (k+1)\frac{\delta}{N} + \gamma.$$

For $x \in [x_k, x_{k+1}]$, let $d_k = g_{l,k} + g_{m,k} + g_{u,k}$ where $g_{l,k}$ and $g_{u,k}$ are each less than $\frac{\delta}{N}$. These terms represent the approximation error of the sigmoid over $[x_k, x_{k+1}]$.

$$\begin{aligned}
|y_k(x) + g_k(x) - f(x)| &= |y_k(x_k) + \mu_k + g_l + g_m - f(x_k) - d_k| \\
&= |y_k(x_k) - f(x_k)| + |\mu_k + g_l + g_m - d_k|
\end{aligned}$$

where $\mu_k = y_k(x) - y_k(x_k)$ is the amount that y_k increases over the same interval. Note that this term must be less than $\frac{\delta}{N}$. By (?) we have $g_l + g_m - d_k = -g_u$ and therefore

$$|y_k(x) + g_k(x) - f(x)| \leq k\frac{\delta}{N} + \gamma + |\mu_k - g_u|$$

Now since $\mu_k \leq \frac{\delta}{N}$ and $g_u \leq \frac{\delta}{N}$ the third term of the inequality must be smaller than $\frac{\delta}{N}$ which proves the statement for $k+1$.

Letting $k = N$ gives us that

$$|y_N(x) - f(x)| \leq N\frac{\delta}{N} + \gamma = \epsilon$$

over I. Setting $y(x) = y_N(x)$ and the proof is complete.

Bibliography

[Aubin and Cellina, 1984] J. P. Aubin and A. Cellina. *Differential Inclusions*. Springer-Verlag, 1984.

[Bates and Watts, 1988] Douglas M. Bates and Donald G. Watts. *Nonlinear Regression and Its Applications*. John Wiley and Sons, New York, 1988.

[Berleant and Kuipers, 1992] Daniel Berleant and Benjamin Kuipers. Combined qualitative and numerical simulation with Q3. In Boi Faltings and Peter Struss, editors, *Recent Advances in Qualitative Physics*. MIT Press, 1992.

[Biegler and Cuthrell, 1985] L. T. Biegler and J. E. Cuthrell. Improved infeasible path optimization for sequential modular simulators – II. the optimization algorithm. *Computers and Chemical Engineering*, 9(3):257–267, 1985.

[Biegler, 1985] L. T. Biegler. Improved infeasible path optimization for sequential modular simulators – I. the interface. *Computers and Chemical Engineering*, 9(3):245–256, 1985.

[Bothe, 1992] Dieter Bothe. Multivalued differential equations on graphs. *Nonlinear Analysis, Theory, Methods, and Applications*, 18(3):245–252, 1992.

[Brajnik and Clancy, 1996] Giorgio Brajnik and Daniel J. Clancy. Trajectory constraints in qualitative simulation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 979–984, 1996.

[Capelo *et al.*, 1996] Antonio C. Capelo, Liliana Ironi, and Stefania Tentoni. The need for qualitative reasoning in automated modeling : A case study. In Yumi Iwasaki and Adam Farquhar, editors, *Qualitative Reasoning, The Tenth International Workshop*, pages 32–39, 1996. AAAI Technical Report WS-96-01.

[Cem Say and Kuru, 1996] A. C. Cem Say and Selahattin Kuru. Qualitative system identification: Deriving structure from behavior. *Artificial Intelligence*, 83(1):75–141, May 1996.

[Cheung and Stephanopoulos, 1990] Jarvis Tat-Yin Cheung and George Stephanopoulos. Representation of process trends – part i. a formal representation framework. *Computers and Chemical Engineering*, 14(4/5):495–510, 1990.

[Clancy and Kuipers, 1993] Daniel J. Clancy and Benjamin Kuipers. Behavioral abstraction for tractable simulation. In *Working Papers from the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, pages 57–64, Orcas Island, Washington, 1993.

[Coiera, 1989] E. Coiera. Generating qualitative models from example behaviors. Technical Report 8901, Department of Computer Science, University of New South Wales, 1989.

[Corliss, 1995a] George F. Corliss. Guranteed error bounds for ordinary differential equations. In *Theory of Numerics in Ordinary and Partial Differential Equations*. Oxford University Press, 1995. Also available at `ftp://boris.mscs.mu.edu/pub/corliss/ValidODE`.

[Corliss, 1995b] Geroge F. Corliss. Introduction to validated ODE solving. Technical Report 416, Department of Mathematics, Statistics and Computer Science – Marquette University, Milwaukee, WI 53233, March 1995.

[Cybenko, 1989] G. Cybenko. Approximation by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[Dalle Molle, 1989] David Dalle Molle. Qualitative simulation of dynamic chemical processes. Technical Report AI89–107, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712, May 1989.

[Danès *et al.*, 1993] P. Danès, L. Travè-Massuyès, and J. Aguilar-Martin. A generic method for computing the response of a numerically-known dynamic system to qualitative inputs. In *Proceedings of IEEE-SMC'93*, Le Touquet, France, 1993.

[Draper and Smith, 1981] Norman R. Draper and H. Smith. *Applied Regression Analysis (second edition)*. John Wiley and Sons, New York, 1981.

[Dvorak and Kuipers, 1989] Daniel Louis Dvorak and Benjamin Kuipers. Model-based monitoring of dynamic systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1238–1243, 1989.

[Gazi *et al.*, 1993] Evangeline Gazi, Herbert Kay, Benjamin Kuipers, W. D. Seider, and Lyle H. Ungar. Controller verification using qualitative reasoning (extended abstract). In *American Institute of Chemical Engineers*, 1993.

[Gazi *et al.*, 1996] E. Gazi, W.D. Seider, and L.H. Ungar. Verfication of controllers in the presence of uncertainty: Application to styrene polymerization. *Industrial and Engineering Chemistry Research*, 35(7):2277–2287, 1996.

[Gelb, 1974] Arthur Gelb. *Applied Optimal Estimation.* MIT Press, Cambridge, MA, 1974.

[Haimowitz, 1994] Ira Joseph Haimowitz. *Knowledge-Based Trend Detection and Diagnosis.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1994.

[Hellerstein, 1990] Joseph Hellerstein. Obtaining quantitative predictions from monotone relationships. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 388–394, 1990.

[Hüllermeier, 1995] Eyke Hüllermeier. Numerical simulation methods for uncertain dynamics. Technical Report tr-ri-95-170, Heinz Nixdorf Institut, Universität–GH–Paderborn, Fürstenaallee 11, D-33102 Paderborn, Germany, November 1995.

[Joerding and Meador, 1991] Wayne H. Joerding and Jack L. Meador. Encoding a priori information in feedforward networks. *Neural Networks*, 4:847–856, 1991.

[Kahaner *et al.*, 1989] David Kahaner, Cleve Moler, and Stephen Nash. *Numerical Methods and Software.* Prentice-Hall, Englewood Cliffs, 1989.

[Kay and Kuipers, 1993] Herbert Kay and Benjamin Kuipers. Numerical behavior envelopes for qualitative models. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 606–613, 1993.

[Kay, 1991] Herbert Kay. Monitoring and diagnosis of multi-tank flows using qualitative reasoning. Master's thesis, The University of Texas at Austin, May 1991.

[Kruskal, 1964] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, March 1964.

[Kuipers and Berleant, 1988] Benjamin Kuipers and Daniel Berleant. Using incomplete quantitative knowledge in qualitative reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 324–329, 1988.

[Kuipers, 1984] Benjamin Kuipers. Commonsense reasoning about causality : Deriving behavior from structure. *Artificial Intelligence*, 24:169–204, 1984.

[Kuipers, 1986] Benjamin Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, September 1986.

[Kuipers, 1994] Benjamin Kuipers. *Qualitative Reasoning – Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, MA, 1994.

[Kulkarni *et al.*, 1991] Deepak Kulkarni, Kiriako Kutulakos, and Peter Robinson. Data analysis using scale-space filtering and bayesian probabilistic reasoning. Technical Report FIA-91-05, NASA Ames Research Center, 1991.

[Leonard *et al.*, 1992] J. A. Leonard, M. A. Kramer, and L. H. Ungar. Using radial basis functions to approximate a function and its error bounds. *IEEE Transactions of Neural Networks*, 3(4):624–627, 1992.

[Ljung, 1987] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, Englewood Cliffs, New Jersey, 1987.

[Lohner, 1988] Rudolf Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, Universität Karlsruhe, 1988.

[Lunze, 1989] Jan Lunze. *Robust Multivariable Feedback Control*. Prentice Hall, 1989.

[Markov and Angelov, 1986] S. Markov and R. Angelov. An interval method for systems of ODE. In *Lecture Notes in Computer Science #212 – Interval Mathematics 1985*, pages 103–108. Springer-Verlag, Berlin, 1986.

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

[Reckhow, 1987] K.H. Reckhow. Error analysis, first order. In Madan O. Singh, editor, *Systems and Control Encyclopedia – Theory, Technology, Applications, Volume 6*, pages 1552–1553. Oxford, 1987.

[Richards *et al.*, 1992] Bradley L. Richards, Ina Kraan, and Benjamin Kuipers. Automatic abduction of qualitative models. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 723–728, 1992.

[Tokuda, 1996] Lance Tokuda. Managing occurrence branching in qualitative simulation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 998–1003, 1996.

[Vescovi *et al.*, 1995] Marcos Vescovi, Adam Farquhar, and Yumi Iwasaki. Numerical interval simulation: Combined qualitative and quantitative simulation to bound behaviors of non-monotonic systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1806–1812, 1995.

[Walter, 1970] Wolfgang Walter. *Differential and Integral Inequalities.* Springer-Verlag, Berlin York, 1970.

[Witkin, 1983] Andrew P. Witkin. Scale-space filtering. In *Proceedings of the Eigth International Joint Conference on Artificial Intelligence*, pages 1019–1022, 1983.

[Zukowski, 1986] Charles A. Zukowski. *The Bounding Approach to VLSI Circuit Simulation.* Kluwer Academic Publishers, Boston, 1986.

Vita

Herbert "Bert" Kay, the son of Jack and Olga Veronica Kay, was born on May 1, 1959, in Boston, Massachusetts. He graduated from Belmont High School in June, 1977. In September, 1977, he entered the California Institute of Technology. He received a Bachelor of Science in Engineering and Applied Science in June, 1981. After summering in Europe, he began work as a Development Engineer for the Hewlett-Packard Company in their logic test group in Santa Clara, California. In October, 1986, he took a position as a Software Engineer at Mettler Instrumente AG, in Greifensee, Switzerland. In September, 1988, he entered The Graduate School of The University of Texas and received a Master of Arts in Computer Science in May, 1991. On August 4, 1989, he married Margaret Clare Fletcher, and on June 10, 1995, his twin daughters (!) Sonia and Nina were born.

Permanent Address: 302 E 41St Street
Austin, TX 78751

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.