

USING POLYNOMIAL APPROXIMATIONS TO
DISCOVER QUALITATIVE MODELS

by

Reha Kamil Gerçeker

B.S. in Computer Engineering, İstanbul Technical University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2006

USING POLYNOMIAL APPROXIMATIONS TO
DISCOVER QUALITATIVE MODELS

APPROVED BY:

Prof. A. C. Cem Say
(Thesis Supervisor)

Prof. Ethem Alpaydm
(Thesis Supervisor)

Prof. Kuban Altinel

DATE OF APPROVAL: 16.06.2006

ABSTRACT

USING POLYNOMIAL APPROXIMATIONS TO DISCOVER QUALITATIVE MODELS

The set of ordinary differential equations (ODE) of a physical system determines the precise numerical behavior that will be exhibited in the future. Abstracting an ODE into a qualitative differential equation (QDE) system removes all quantities from the equations and leaves only the structure of the equation system intact. However, a QDE is still very useful in anticipating the future when the qualitative behavior of the system is important rather than the exact numerical solution. Another name for a QDE is a qualitative model. Qualitative relationships make up a qualitative model and it is possible to analyze how a change in one of the variables in the system affects the other variables using these relationships. In other words, qualitative models help us understand the underlying mechanism which determines the behavior that we observe.

Writing qualitative models from scratch is a difficult problem which needs to be done by an intelligent expert with domain specific knowledge. Automating the discovery of qualitative models from observations is a difficult problem of machine learning. Various algorithms have been proposed for the solution of this problem in the literature. This thesis presents a new algorithm called LYQUID for the solution of the same problem. The algorithm uses polynomials fitted on observed numerical data as approximations to the underlying real world functions; discovery of qualitative relationships is then performed over those polynomials rather than the original data samples. LYQUID is shown to be a fast and successful learning algorithm which performs very well on benchmark models and tolerates high levels of noise. The algorithm not only relaxes the restrictions over how the data are sampled but it is also capable of working with missing data.

ÖZET

NİTEL MODELLERİN BULUNMASINDA POLİNOM YAKLAŞIKLAMALARININ KULLANILMASI

Fiziksel bir dizgede değişkenlerin gelecekteki sayısal davranışı o dizgeye ait adi diferansiyel denklem takımı tarafından belirlenir. Bu denklem takımındaki sayısal büyüklüklerin kaldırılması ile elde edilen nitel diferansiyel denklem sistemi ise adi denklem sisteminin yalnızca yapısını ifade eder. Bir nitel denklem sistemi, dizgedeki değişkenlerin davranışlarının öngörülmesinin değişkenlerin gelecekteki sayısal değerlerinin bulunmasından daha önemli olduğu durumlarda çok kullanışlıdır. Nitel denklem sistemleri aynı zamanda nitel model olarak adlandırılır. Bir nitel modeldeki ilişkiler, değişkenlerden birisinin değerindeki bir değişimin dizgedeki diğer değişkenleri nasıl etkileyeceğini incelemek için kullanılabilir. Diğer bir deyişle, nitel modeller dizgenin bizce gözlemlenen davranışına neden olan arkaplandaki mekanizmanın anlaşılmasına yardımcı olurlar.

Nitel modelleri sıfırdan yazmak kolay değildir ve alanında bilgi sahibi bir uzman tarafından yapılması gerekir. Nitel modellerin gözlemlerden otomatik olarak öğrenilmesi daha da zordur ve otomatik öğrenme alanındaki problemlerden bir tanesidir. Tezde bu problemin çözümüne yönelik olarak LYQUID adı verilen bir algoritma önerilmektedir. Algoritma gözlemlenmiş sayısal örneklerin üzerine polinom eğriler oturarak örnekleri oluşturan gerçek fonksiyonlara yaklaşıklık bulmaktadır; değişkenler arasındaki nitel ilişkilerin bulunmasında ise esas örnekler yerine bu polinomlar kullanılır. LYQUID'in yazındaki denektaş problemleri üzerinde başarılı sonuçlar veren, hızlı ve gürültüye toleransı yüksek olan bir algoritma olduğu gösterilmiştir. Algoritma, örnekleme nasıl yapıldığı ile ilgili kısıtlamaları kaldırmanın ötesinde değişkenlerin belli zaman aralıklarında örneklenmediği durumlarda da başarılı olabilmektedir.

TABLE OF CONTENTS

| | |
|---|------|
| ABSTRACT | iii |
| ÖZET | iv |
| LIST OF FIGURES | vii |
| LIST OF TABLES | viii |
| 1. INTRODUCTION | 1 |
| 1.1. Qualitative Reasoning | 1 |
| 1.2. QSIM Vocabulary | 2 |
| 1.3. Qualitative Modeling | 3 |
| 1.4. Outline | 4 |
| 2. MATHEMATICAL BACKGROUND | 6 |
| 2.1. Polynomial Curve Fitting | 6 |
| 2.2. Fitting Piecewise Polynomials | 11 |
| 2.3. Polynomial Representation, Operations and Complexity | 13 |
| 2.4. Evaluating Equality of Polynomials | 15 |
| 2.5. Checking Existence of Polynomial Roots | 16 |
| 3. A NEW ALGORITHM: LYQUID | 20 |
| 3.1. The Problem | 20 |
| 3.2. Inputs, Outputs and Initialization | 21 |
| 3.3. Units | 22 |
| 3.4. Piecewise Polynomial Operations | 23 |
| 3.5. Generation of Hidden Variables | 24 |
| 3.6. Discovering Qualitative Constraints | 28 |
| 3.7. The Algorithm in Pseudo-Code | 32 |
| 3.8. Overall Complexity | 34 |
| 4. EXPERIMENTAL RESULTS | 37 |
| 4.1. Experiments on Clean Data | 37 |
| 4.1.1. U-tube | 37 |
| 4.1.2. Cascaded Tanks | 39 |
| 4.1.3. Coupled Tanks | 42 |

| | |
|---|----|
| 4.2. Experiments on Noisy Data | 48 |
| 4.2.1. U-tube | 48 |
| 4.2.2. Cascaded Tanks | 50 |
| 4.3. Other Experiments | 50 |
| 4.3.1. Arbitrarily Sampled Data | 50 |
| 4.3.2. Missing Data | 51 |
| 4.4. Interpreting the Results | 52 |
| 5. RELATED WORK | 56 |
| 5.1. QMN | 56 |
| 5.2. LAGRANGE and LAGRAMGE | 60 |
| 5.3. SQUID | 62 |
| 5.4. Q _{OPH} | 63 |
| 5.5. Other Algorithms | 65 |
| 6. CONCLUSION | 68 |
| 6.1. Overview | 68 |
| 6.2. LYQUID's Contributions | 71 |
| 6.3. Further Work | 72 |
| REFERENCES | 75 |

LIST OF FIGURES

| | | |
|-------------|---|----|
| Figure 2.1. | Sine wave (solid line) approximated by two polynomials on $[-\pi, \pi]$ | 12 |
| Figure 3.1. | $p_1(t)$ (dash) and $p_2(t)$ (dash dot) | 25 |
| Figure 4.1. | Benchmark models: u-tube, cascaded tanks and coupled tanks . . | 37 |
| Figure 4.2. | U-tube samples with $N(0, 1)$ Gaussian noise: $level_A$ (left), $level_B$ (right) | 48 |
| Figure 4.3. | Cascaded tanks samples with $N(0, 0.5)$ Gaussian noise: $amount_A$ (top left), $amount_B$ (top right), $inflow$ (bottom) | 51 |

LIST OF TABLES

| | | |
|------------|--|----|
| Table 1.1. | ODE vs. QDE of an undamped simple spring system | 3 |
| Table 2.1. | A Sturm sequence on $(-4, 4)$ | 17 |
| Table 2.2. | $V(t)$ not affected from the zero at t_0 | 18 |
| Table 3.1. | An example list of variables maintained by LYQUID | 26 |
| Table 3.2. | Symbols appearing in Algorithm 3 | 33 |
| Table 3.3. | Symbols that exist implicitly inside Algorithm 3 | 33 |
| Table 4.1. | Discoveries from the clean u-tube dataset | 38 |
| Table 4.2. | Discoveries from the clean cascaded tanks dataset using 5^{th} degree polynomials | 41 |
| Table 4.3. | Discoveries from the clean cascaded tanks dataset using 11^{th} degree polynomials | 42 |
| Table 4.4. | List of variables generated in the coupled tanks experiment | 44 |
| Table 4.5. | Monotonic function discoveries from the clean coupled tanks dataset | 46 |
| Table 4.6. | Correctness of the discoveries in Table 4.5 | 47 |
| Table 4.7. | Discoveries from the noisy u-tube dataset | 49 |
| Table 4.8. | Discoveries from the arbitrarily sampled dataset | 52 |

| | | |
|------------|--|----|
| Table 4.9. | Discoveries from the dataset with missing data | 53 |
| Table 5.1. | Variable representations of QMN and LYQUID | 57 |
| Table 5.2. | LYQUID’s discoveries on the comparison dataset | 59 |
| Table 6.1. | Time-series data for two observed variables | 72 |

1. INTRODUCTION

1.1. Qualitative Reasoning

Computers are very good at numerical computation but it is not his skills of numerical computation that makes a man intelligent. One of the most important qualities of human intelligence is that it is capable of perceiving and anticipating physical behavior. Physical behavior can of course be modeled numerically, but it is something more general. Consider, for example, a ball being thrown up in the air. There is an infinite number of numerical solutions to this physical system according to the ball's initial velocity and gravitational attraction; however, under reasonable assumptions, all of these different solutions in fact map to the same physical behavior in the sense that the ball rises up in the air for some time but eventually falls back down to the ground. Alternatively, when you aim to throw the ball up to some specific height, there is again an infinite number of numerical possibilities, whereas, the number of different behaviors in such a setting is only three: *(i)* the ball does not reach the aimed height and falls down, *(ii)* the ball stops instantaneously at the aimed height and falls down, *(iii)* the ball goes over the aimed height and falls down. In addition to being capable of making numerical simulations of physical systems, computers are also able to predict possible behaviors of a physical system using qualitative reasoning.

Qualitative reasoning has been around for almost over 20 years. The Qualitative Process Theory [1] and the QSIM formalism [2, 3] are the two primary theories in the field of qualitative reasoning. All of the work in this field of research use one of these two theories as the starting point. Both of the theories provide the vocabulary to specify relationships between variables in a physical system. These relationships are qualitative; they do not express numerical relationships between the variables and they do not involve quantities. The work in this thesis is carried out based on the QSIM formalism and that is the reason why a section is devoted to explain the QSIM vocabulary.

1.2. QSIM Vocabulary

Qualitative relationships in QSIM are expressed with a small vocabulary and their meanings are very straightforward. Thinking of x , y and z as variables in a physical system, explanations of the QSIM constraints are as follows:

minus $\text{minus}(x\ y)$ means that x and y are equal to each other except with opposite signs.

d/dt $\text{d/dt}(x\ y)$ means that y is the time derivative of x . In order to express higher order derivatives, a chain of d/dt constraints must be used with intermediate variables inserted.

add $\text{add}(x\ y\ z)$ means that x and y add up to z .

mult $\text{mult}(x\ y\ z)$ means that z is the product of x and y .

M⁺, M⁻ $\text{M}^+(x\ y)$ means that there exists a function f such that $y = f(x)$ and $\frac{dy}{dx} > 0$. In this case f is a monotonically increasing function. In other words, it implies that x and y have the same directions of change, i.e. they increase, decrease and come to quiescence together. In fact, the formal definition of an M^+ is a little more complicated and it will be given below. The only difference with an $\text{M}^-(x\ y)$ is the derivative's sign condition, which is replaced with $\frac{dy}{dx} < 0$. In this case f is a monotonically decreasing function and x and y have opposite directions of change.

Let \mathfrak{R}^* denote the extended real number line which includes the endpoints $-\infty$ and $+\infty$. The formal definition of an M^+ involves a concept called reasonable function [3]:

Definition 1 *Where $[a, b] \subseteq \mathfrak{R}^*$, the function $f : [a, b] \rightarrow \mathfrak{R}^*$ is a reasonable function over $[a, b]$ if*

1. f is continuous over $[a, b]$,
2. f is continuously differentiable on (a, b) ,
3. f has only finitely many critical points in any bounded interval, and

4. the one sided limits $\lim_{t \rightarrow a^+} f'(t)$ and $\lim_{t \rightarrow b^-} f'(t)$ exist in \mathfrak{R}^* . Define $f'(a)$ and $f'(b)$ to be equal to these limits.

Definition 2 M^+ is the set of reasonable functions $f : [a, b] \rightarrow \mathfrak{R}^*$ such that $f' > 0$ over (a, b) .

QSIM performs qualitative simulation of M^+ and M^- according to these formal definitions. However, our application does not deal with these strict definitions but assume that any function f encountered during an execution of the algorithm is reasonable and it suffices only to look only for $f' > 0$ or $f' < 0$.

1.3. Qualitative Modeling

Every physical system is governed by a set of ordinary differential equations (ODE). These equations involve quantities and getting rid of the quantities by using some kind of qualitative vocabulary transforms the equations into a set of qualitative differential equations (QDE). The QDE of a system is also called as the qualitative model of the system. For example, Table 1.1 shows the ODE and QDE of an undamped simple spring system, where the QDE is expressed in the QSIM vocabulary. x , v and a are the position, velocity and acceleration of the object attached to the spring; k and m appearing in the ODE are the spring constant and the object's mass.

Table 1.1. ODE vs. QDE of an undamped simple spring system

| ODE | QDE |
|---------------------|-----------------|
| $v = \frac{dx}{dt}$ | d/dt(x v) |
| $a = \frac{dv}{dt}$ | d/dt(v a) |
| $a = \frac{-k}{m}x$ | $M^-(a$ $x)$ |

A QDE corresponds to a family of ODEs. As in the case of the undamped simple spring, for every different value of k and m the system has a unique ODE, whereas its QDE never changes. In other words, the QDE is related to the structure

of the equation system which determines the system's behavior, whereas, the ODE determines the unique numerical solution.

Formulating and solving differential equation systems belonging to real world systems is a main problem in several engineering disciplines. This is necessary to understand why the system behaves the way it does and to anticipate how the system will behave in the future. However, situations arise frequently where it is difficult to even formulate an ODE for a given system, let alone solve it to reveal the exact functions for its variables. In such cases, moving into the qualitative domain helps not only with the modeling problem but also allows simulation of the system with less precise information.

System identification deals with the problem of discovering the mathematical model of a system from observed input and output variables. The main problem in system identification problems is the size of the initial search space. Narrowing down the huge search space into descriptions of possible models is called structural identification; deciding for the precise model from that subset is a problem of parameter estimation. In this respect, qualitative modeling is a really expressive tool in representing structure in the structural identification stage. Using qualitative modeling, it is possible to express relationships between system variables without ever having to find out what they precisely are. Therefore, discovering the qualitative model of a physical system is an important step in understanding the underlying mechanism which determines the system's behavior. Different aspects of qualitative reasoning and benefits of qualitative reasoning based approaches for system identification are examined thoroughly by Travé-Massuyès, Ironi and Dague [4]. Similarly, Bratko and Šuc have reviewed different techniques used in learning qualitative models in detail [5].

1.4. Outline

This thesis proposes a new algorithm, which is called LYQUID, for automated discovery of a qualitative model from observed numerical data. The algorithm approaches the problem from a rather new perspective; it first fits polynomials to the

observed numerical samples. These polynomials represent each observed variable as a function of time. Discovery of qualitative constraints between the observed variables is then carried out using these polynomials rather than the original samples. The algorithm is very lenient about how data are sampled; neither the observed quantities must be sampled with the same frequency nor the samples have to be equally spaced in time. Even absence of samples at some intervals could be tolerated. Moreover, the intuition behind the algorithm, i.e. fitting polynomials of time to the data samples, proves to be very strong against noise over the samples. It is that same basic idea which also lowers computational complexity; LYQUID's complexity at the model discovery stage does not depend on the number of observed samples.

The next chapter will explain the mathematical background behind the work carried out in this thesis. These include all polynomial related issues as well as the algorithms used in fitting polynomials to the samples. The third chapter will go deep into the details of LYQUID including its computational complexity. The fourth chapter summarizes the results of experiments in which LYQUID was tested over benchmark models in various different settings. The fifth chapter analyzes similar algorithms in literature and also compares different approaches with LYQUID's. The sixth and final chapter concludes the work and talks about possible future directions.

2. MATHEMATICAL BACKGROUND

The new algorithm proposed in Chapter 3 has a mathematical background related to polynomials. The algorithm calculates univariate polynomial approximations to sampled data and then operates on these polynomials during the rest of its execution. Apart from ordinary algebraic and differential operations on polynomials, the algorithm needs to check for the equality of two polynomials and the existence of polynomial roots over a given interval. This chapter explains the mathematical foundations of all polynomial related operations performed by the algorithm.

2.1. Polynomial Curve Fitting

When an arbitrary and possibly unknown function $f(t)$ is sampled over time, it is possible to use the observed timepoint–value pairs to reconstruct an approximation to the original function in the observation interval. There are several interpolation techniques for this purpose like generalized least–squares, Lagrange polynomials or spline interpolation [6]. Although none of these methods has been proven to be superior to the others in terms of accuracy of the approximation, polynomial least–squares techniques are preferred in our specific application. The reasons behind this decision will be discussed soon.

Suppose that a function $f(t)$ is sampled at N timepoints to produce $\{(t_i, x_i)\}_{i=1}^N$ timepoint–value pairs. The objective in a least–squares approximation is to find an $\bar{f}(t)$ such that

$$\bar{f}(t) = \sum_{k=0}^d a_k \Phi_k(t) \quad (2.1)$$

where $\{\Phi_k(t)\}_{k=0}^d$ are a finite sequence of basis functions defined for every t_i . The coefficients denoted by a_k in this equation are to be determined so that the least–squares

error criterion

$$E = \sum_{i=1}^N [f(t_i) - \bar{f}(t_i)]^2 = \sum_{i=1}^N \left[x_i - \sum_{k=0}^d a_k \Phi_k(t_i) \right]^2 \quad (2.2)$$

is minimized. Note that E is a function of a_k ($k = 0, 1, \dots, d$). It is easy to solve the optimization problem by taking partial derivatives of E for all a_k and setting each derivative equal to 0. This process would yield $d + 1$ equations with $d + 1$ unknowns and the resulting system is called the normal equations. It is possible to solve for a_k by solving the system of normal equations.

Picking $\Phi_k(t) = t^k$ as basis functions is a very straightforward and reasonable approach. Such a choice produces a polynomial least-squares approximation when substituted into Equation (2.1). This approach is reasonable because recalling Taylor series from elementary calculus, polynomials can approximate arbitrary functions within a margin of error which depends on the degree of the approximating polynomial. Recall that approximation error of the Taylor expansion of a function tends to zero as the degree of the Taylor polynomial tends to infinity. In other words, it is possible to increase the accuracy of the polynomial least-squares approximation by using a larger sequence of $\{\Phi_k(t)\}$ or by simply picking a larger d as it determines the length of the sequence.

This choice of basis functions, on the other hand, poses a problem. When $\Phi_k(t) = t^k$ are used as basis functions, the matrix which represents the system of normal equations is ill-conditioned [6, 7]. Calculating the inverse of an ill-conditioned matrix is not a numerically stable procedure; Ralston and Rabinowitz [7, p. 252] argue that the calculated inverse may even be affected by round-off errors in floating-point representation or arithmetic, let alone the measurement errors made during sampling. Since inverse of the original ill-conditioned matrix is required to solve the system of normal equations, this straightforward approach generally produces inaccurate results.

Another method, which is more stable numerically and results in more accurate

solutions most of the times, is using a set of orthogonal functions as $\{\Phi_k(t)\}$. A very famous and widely used way of formulating Equation (2.1) by using orthogonal basis functions is the Fourier transform. The coefficients $\{a_k\}$ in this well-known transform are called the Fourier coefficients and the set of orthogonal basis functions is the set of cosines, i.e. $\Phi_k(t) = \cos(kt)$, $k = 0, 1, \dots, d$. It is also known that every function satisfying the Dirichlet conditions can be expressed as a sum of an infinite number of cosines [6, p. 61]. Almost all practical functions satisfy Dirichlet conditions, which basically require that the function has (i) finite number of discontinuities, (ii) finite number of extrema, (iii) finite energy (that is, the function is absolutely integrable) over its domain. Therefore, similar to Taylor series, practically all functions can be expressed with Fourier series. Only the sets of basis functions are different in the two cases.

Just like the set of cosines, it is also possible to construct a finite set of polynomials which satisfy the orthogonality condition and use them as basis functions. According to the definition, a set of polynomials $\{p_k(t)\}_{k=0}^d$ is orthogonal over a set of points $\{t_i\}_{i=1}^N$ with respect to a weight function $w(t)$ if the condition

$$\sum_{i=1}^N w(t_i) p_u(t_i) p_v(t_i) = 0 \quad \text{if } u \neq v \quad (2.3)$$

is satisfied. The weight function $w(t)$ is an arbitrary function with the requirement that $w(t) > 0$ for all t in w 's domain. Ralston and Rabinowitz [7] prove that a set of orthogonal polynomials $\{p_k(t)\}$ satisfy the recurrence relation

$$\begin{aligned} p_{k+1}(t) &= (t - \alpha_{k+1}) p_k(t) - \beta_k p_{k-1}(t) \quad k \geq 0 \\ p_0(t) &= 1, \quad p_{-1}(t) = 0 \end{aligned} \quad (2.4)$$

where α_{k+1} and β_k are constants determined by

$$\alpha_{k+1} = \frac{\sum_{i=1}^N w(t_i) t_i [p_k(t_i)]^2}{\sum_{i=1}^N w(t_i) [p_k(t_i)]^2} \quad (2.5)$$

$$\beta_k = \frac{\sum_{i=1}^N w(t_i) [p_k(t_i)]^2}{\sum_{i=1}^N w(t_i) [p_{k-1}(t_i)]^2} \quad (2.6)$$

Equations (2.4), (2.5) and (2.6) are used together to obtain the coefficients of Equation (2.1):

$$a_k = \frac{\sum_{i=1}^N w(t_i) x_i p_k(t_i)}{\sum_{i=1}^N w(t_i) [p_k(t_i)]^2} \quad (2.7)$$

$\Phi_k(t) = p_k(t)$ are used as the basis functions in this case along with the coefficients calculated as above. Note that these orthogonal basis polynomials are found by substituting α and β coefficients of (2.5) and (2.6) in recurrence relations of (2.4). It is important to observe that the orthogonal basis polynomials $\{p_k(t)\}_{k=0}^d$ and their coefficients a_k ($k = 0, 1, \dots, d$) in (2.1) are calculated separately for every given set of data points which are denoted by $\{(t_i, x_i)\}_{i=1}^N$. That is, the set of basis polynomials and their coefficients used in approximation may differ for different datasets which are sampled from even the same function.

Algorithm 1 is adopted from Ralston and Rabinowitz [7, p. 259] and it is the pseudo-code for the algorithm which computes a polynomial approximation to an unknown but observable function. It is assumed that this unknown function is sampled at N different timepoints to give a dataset denoted by $\{(t_i, x_i)\}_{i=1}^N$ and this dataset is input to the algorithm. The number of basis polynomials to use or, equally, the degree of the polynomial approximation, which is denoted by d , is also an input parameter of the algorithm. The algorithm calculates the coefficients defined by Equations (2.5), (2.6) and (2.7) together with the orthogonal basis polynomials which are determined by the recurrence relations of (2.4). γ and ω appearing in the algorithm are intermediate coefficients. Finally, the algorithm also calculates figures denoted by \bar{x}_i ; \bar{x}_i is the output of the resultant polynomial approximation at timepoint t_i . In other words, \bar{x}_i is an estimate for the observed x_i .

Algorithm 1 Polynomial fitting algorithm

for $i = 1$ to N **do**

$$p_0(t_i) \leftarrow 1$$

$$p_{-1}(t_i) \leftarrow 0$$

$$\bar{x}_i \leftarrow 0$$

end for

$$\gamma_0 \leftarrow N$$

$$\beta_0 \leftarrow 0$$

for $k = 0$ to d **do**

$$\omega_k \leftarrow \sum_{i=1}^N x_i p_k(t_i)$$

$$a_k \leftarrow \omega_k / \gamma_k$$

for $i = 1$ to N **do**

$$\bar{x}_i \leftarrow \bar{x}_i + a_k p_k(t_i)$$

end for
if $k = d$ **then**

stop

end if

$$\alpha_{k+1} \leftarrow \sum_{i=1}^N x_i [p_k(t_i)]^2 / \gamma_k$$

for $i = 1$ to N **do**

$$p_{k+1}(t_i) \leftarrow (t_i - \alpha_{k+1}) p_k(t_i) - \beta_k p_{k-1}(t_i)$$

end for

$$\gamma_{k+1} \leftarrow \sum_{i=1}^N [p_k(t_i)]^2$$

$$\beta_{k+1} \leftarrow \gamma_{k+1} / \gamma_k$$

end for

2.2. Fitting Piecewise Polynomials

One of the parameters of Algorithm 1 is the degree of the polynomial used in curve fitting and it is denoted by d . When this degree increases, the accuracy of the fit automatically increases; remember that the degree of the complete Taylor series expansion of an arbitrary function tends to infinity. However, when a high degree approximation is used for a noisy dataset, there is also the possibility of overfitting unnecessarily to the noise over the data. Therefore, this parameter needs to be tuned according to the complexity of input data.

d determines the accuracy of the approximation and there are two main factors that hinder accuracy:

- d being insufficiently small for the input time interval. As an example to the problem here, consider the sine wave. Setting $d = 5$, unfortunately, it is not possible to have a polynomial fit with an acceptable square error over the whole $[-\pi, \pi]$ interval even when using perfectly clean samples from the sine wave; a single fifth degree polynomial approximates the sine wave with reasonable error only on $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$ and additional polynomials are needed in order to approximate the rest of the wave. A single seventh degree polynomial, on the other hand, gives a low error approximation over the whole $[-\pi, \pi]$ interval. This is illustrated in Figure 2.1. The plotted polynomials are fifth and seventh degree Taylor expansions of the sine function around $t = 0$.
- Noise over the samples. A polynomial of degree d may actually be sufficient to fit perfectly clean samples on a time interval. However, data are never such clean and square error of the approximation could still be high because of the noise over data samples.

Inaccuracy caused by the first problem is not tolerable, whereas, inaccuracy related to noise must be allowed to some extent. When d is fixed to some value, the only way to increase accuracy of the fit is to increase the number of polynomials used in approximation, by splitting the large input time interval into smaller sub-intervals.

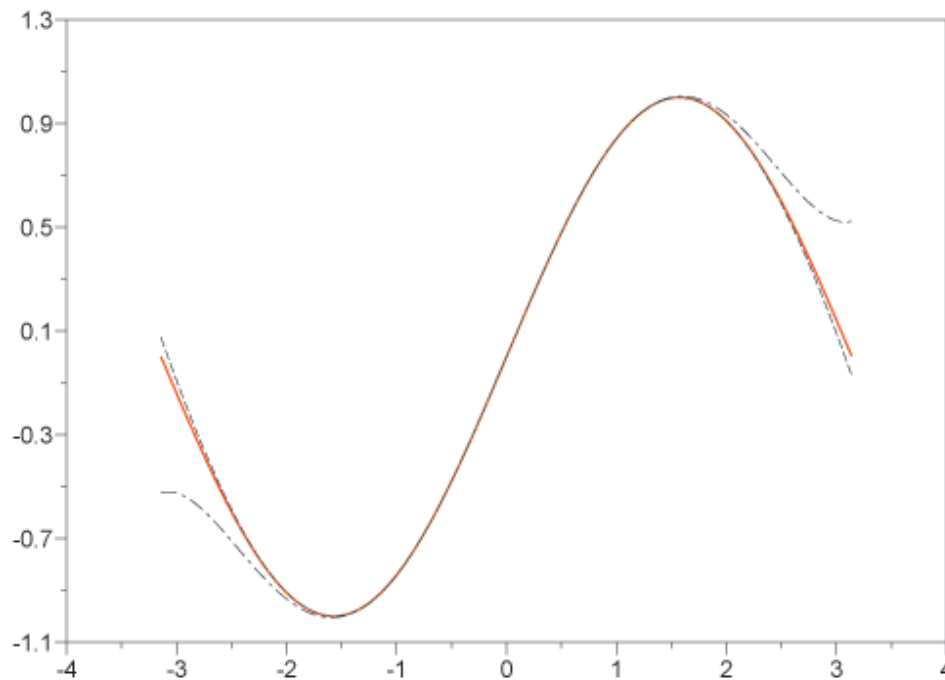


Figure 2.1. Sine wave (solid line) approximated by two polynomials on $[-\pi, \pi]$

This corresponds to using piecewise polynomials rather than fitting a single polynomial on all of the samples.

Algorithm 2 is used to divide a large time interval into smaller sub-intervals to increase the accuracy of approximation. Note that this algorithm makes use of Algorithm 1 as a subprocedure. A parameter called *Tolerance* is used to decide whether the error of approximation is acceptable or not. When the error is not acceptable, the algorithm splits the time interval at the midpoint and fits two polynomials on both the new sub-intervals. In order for this split to be accepted, the accuracy after the split must be improved; parameters *Improvement* and *SplitImprovement* determine the acceptable rate of improvement. Otherwise, the split is rejected and a single polynomial is fitted on the whole interval. If the actual cause of inaccuracy in an approximation is the noise over the data, then this mechanism of accepting or rejecting a split makes sure that the split is really necessary. As a final remark, it should be noted that the partitioning algorithm does not require that the fitted piecewise polynomial be

continuous at transition points between sub-intervals.

2.3. Polynomial Representation, Operations and Complexity

From a computer programmer's point of view, a polynomial is basically an array of real numbers; if the degree of the polynomial is d , then $(d + 1)$ coefficients need to be stored in computer's memory. In reality, when some data samples in the time interval $[a, b]$ are going to be represented with a polynomial $p(t)$, it is a good idea to store coefficients of the polynomial $p(t - a)$ together with the offset a , rather than keeping coefficients of $p(t)$. This helps coefficients of the stored polynomial remain in more reasonable magnitudes, as shown in this example:

$$\begin{aligned} p(t) &= t^5 + 46t^4 + 847t^3 + 7804t^2 + 35983t + 66430 \\ p(t - 9) &= t^5 + t^4 + t^3 + t^2 + t + 1 \end{aligned}$$

Keeping magnitudes relatively small in floating-point arithmetic makes the values of variables more precise. For example, when 24 mantissa bits are used in floating-point arithmetic, the floating-point numbers in the interval $[1, 2)$ are equally spaced being $\frac{1}{2^{24}}$ apart, whereas, numbers in the interval $[2^{30}, 2^{31})$ are equally spaced with a much larger spacing of 64. In other words, when precision matters, it is always a good idea to work with smaller magnitudes in floating-point arithmetic [7, p. 14].

Performing operations on polynomials in the computer is very straightforward. For example, when you want to compute the time derivative of a polynomial, coefficients of the new polynomial is found by decreasing all degrees by one, prior to multiplying the original coefficients with their degrees. As another example, when you want to compute a sum, coefficients of the same degree add up to form the coefficients of the resultant polynomial. Shortly, implementing operations on single polynomials is a simple manipulation of their coefficients.

The described representation scheme of polynomials, on the other hand, has a side effect on polynomial operations. Normally adding or subtracting two polynomials

Algorithm 2 Interval partitioning algorithm

partition is a $\{start, stop, poly, error\}$ tuple

intervals and *result* are stacks of partition

$P \leftarrow d$ -degree polynomial on $S[1..N]$

$e \leftarrow$ square error of P on $S[1..N]$

intervals.push $\{1, N, P, e\}$

while *intervals.empty* is false **do**

interval \leftarrow *intervals.pop*

$e \leftarrow$ *interval.error*

$c \leftarrow$ *interval.stop* - *interval.start* + 1

if $(e/c) \leq Tolerance$ **then**

result.push interval

else

ini \leftarrow *interval.start*

fin \leftarrow *interval.stop*

mid $\leftarrow \lfloor \frac{ini+fin}{2} \rfloor$

$P1 \leftarrow d$ -degree polynomial on $S[ini..mid]$

$e1 \leftarrow$ square error of $P1$ on $S[ini..mid]$

$P2 \leftarrow d$ -degree polynomial on $S[mid + 1..fin]$

$e2 \leftarrow$ square error of $P2$ on $S[mid + 1..fin]$

chgInTotalErr $\leftarrow (e - (e1 + e2)) / e$

chgInInterval1 $\leftarrow (e - 2 e1) / e$

chgInInterval2 $\leftarrow (e - 2 e2) / e$

if *chgInTotalErr* $\geq Improvement$ or

chgInInterval1 $\geq SplitImprovement$ or

chgInInterval2 $\geq SplitImprovement$

then

intervals.push $\{mid + 1, fin, P2, e2\}$

intervals.push $\{ini, mid, P1, e1\}$

else

result.push interval

end if

end if

end while

takes linear time in the degree of the polynomials. However, when offsets of the two operands are different, one of the operands must be translated over time so that the offsets match. Translation of a polynomial over time is an operation whose complexity is quadratic in the degree of the polynomial. That is why even addition and subtraction of polynomials take $O(d^2)$ time in the worst case with this representation scheme. Other binary polynomial operations like multiplication and synthetic division take $O(d^2)$ time even without a mandatory translation of one of the operands and they therefore are unaffected by the time shifting requirement, at least from a complexity theoretic point of view. Unary operations like differentiation and integration are also unaffected, and they simply take $O(d)$ time, since they do not require an additional translation over time.

2.4. Evaluating Equality of Polynomials

Being able to check equality of polynomials is the fundamental requirement for the purpose of our algorithm. Even though two polynomials are equal only when their coefficients are equal, it is generally the case that curve fitting does not produce such exact matches. In other words, some margin of error must be tolerated during the comparison. When comparing two polynomials $p(t)$ and $q(t)$ on a time interval $[a, b]$, this is an appropriate error criterion:

$$e = \frac{\int_a^b [p(t) - q(t)]^2 dt}{b - a}$$

$$equal(p, q) = \begin{cases} true & \text{if } e \leq iTolerance \\ false & \text{otherwise} \end{cases} \quad (2.8)$$

$iTolerance$ is a parameter which determines the margin of error that will be tolerated during comparison. Similar to $Tolerance$, which appeared in Algorithm 2, $iTolerance$ specifies the tolerable square error per unit time. Note that this error criterion is not based on evaluating each polynomial at several timepoints inside the interval of comparison. In fact, error per unit time, which is denoted by e , is based on the coefficients of p and q . This is why coefficients of polynomials are kept in more precise

regions of the floating-point numbers as described in Section 2.3.

2.5. Checking Existence of Polynomial Roots

There are times when it is necessary to determine whether or not a polynomial $p(t)$ has a root at some timepoint $t \in [a, b]$. When the problem is to determine the existence of the root rather than calculating the exact value of it, it is possible to make use of a mathematical concept called Sturm sequences. The simplified discussion in this section has been adopted from [7].

Definition 3 *Let $\{p_1(t), p_2(t), \dots, p_k(t)\}$ be a sequence of polynomials. This sequence is a Sturm sequence on an interval (a, b) if, (i) $p_k(t)$ does not vanish in (a, b) , (ii) at any zero of $p_i(t)$, $i = 2, \dots, k - 1$, the two adjacent polynomials are nonzero and have opposite signs:*

$$p_{i-1}(t)p_{i+1}(t) < 0 \tag{2.9}$$

Definition 4 *Let $\{p_i(t)\}$, $i = 1, \dots, k$ be a Sturm sequence and t_0 be a point in (a, b) such that $p_1(t_0) \neq 0$. $V(t_0)$ is defined as the number of changes of sign of $\{p_i(t_0)\}$, zero values being ignored.*

Note that subscripts of the polynomials in the above definitions do not necessarily reflect their degrees. Table 2.1 shows an example Sturm sequence and illustrates both of the definitions; it is possible to verify that Equation (2.9) holds for the given sequence. Values of $V(t)$ at $t = 1$ and $t = 3$ are undefined according to the definition.

Given a Sturm sequence on (a, b) , $V(t)$ could be used as a tool to find the number of real roots of the first polynomial $p_1(t)$ in the sequence. This arises from the fact that the value of $V(t)$ changes in (a, b) only when t passes through a zero of $p_1(t)$. This fact is a direct result of the property of Sturm sequences stated in Equation (2.9). If t_0 is a

zero of $p_i(t)$ ($i = 2, \dots, k - 1$) (recall that $p_k(t)$ does not vanish in (a, b) , that is, it does not have zeroes), then there is a sign change between the two adjacent polynomials $p_{i-1}(t)$ and $p_{i+1}(t)$ because of the second property of Sturm sequences. Regardless of the multiplicity of the zero at t_0 , there has to be exactly one sign change immediately to the left and right of t_0 as shown in Table 2.2. In other words, existence of a zero at t_0 does not affect $V(t)$ unless it belongs to $p_1(t)$. Table 2.1 exemplifies this statement very nicely.

Then, in order to find out whether or not an arbitrary polynomial $p(t)$ has any roots on an arbitrary time interval (a, b) , the only remaining problem is to construct a Sturm sequence from $p(t)$. Fortunately, it is possible to do this by using the recurrence relation

$$\begin{aligned} p_1(t) &= p(t) \\ p_2(t) &= p'(t) \\ p_{j-1}(t) &= q_{j-1}(t)p_j(t) - p_{j+1}(t) \quad 2 \leq j \leq d, \end{aligned} \tag{2.10}$$

where d is the degree of $p(t)$. The polynomials $q_{j-1}(t)$ in the recurrence relation are the quotients that result from symbolic division of $p_{j-1}(t)$ to $p_j(t)$; these quotients, however, are not used anywhere but instead the remainders are used to continue building the sequence. The resultant sequence of polynomials from Equation (2.10), which is denoted by $\{p_i(t)\}$, $i = 1, \dots, d + 1$, is a Sturm sequence [7]. As a result of this, whenever

Table 2.1. A Sturm sequence on $(-4, 4)$

| Polynomial | $t = -4$ | $t = -2$ | $t = 1$ | $t = \frac{9}{8}$ | $t \approx 2.32$ | $t = 3$ | $t = 4$ |
|------------------------------------|----------|----------|---------|-------------------|------------------|---------|---------|
| $p_1(t) = (t - 1)(t - 3)(t^2 + 1)$ | + | + | 0 | - | - | 0 | + |
| $p_2(t) = t^3 - 3t^2 + 2t - 1$ | - | - | - | - | 0 | + | + |
| $p_3(t) = t^2 + t - 2$ | + | 0 | 0 | + | + | + | + |
| $p_4(t) = -8t + 9$ | + | + | + | 0 | - | - | - |
| $p_5(t) = -25$ | - | - | - | - | - | - | - |
| $V(t)$ | 3 | 3 | ! | 2 | 2 | ! | 1 |

Table 2.2. $V(t)$ not affected from the zero at t_0

| Polynomial | $t = t_0 - \epsilon$ | $t = t_0$ | $t = t_0 + \epsilon$ |
|--------------|----------------------|-----------|----------------------|
| $p_{i-1}(t)$ | + | + | + |
| $p_i(t)$ | +/- | 0 | +/- |
| $p_{i+1}(t)$ | + | - | 0 |

| Polynomial | $t = t_0 - \epsilon$ | $t = t_0$ | $t = t_0 + \epsilon$ |
|--------------|----------------------|-----------|----------------------|
| $p_{i-1}(t)$ | - | - | - |
| $p_i(t)$ | +/- | 0 | +/- |
| $p_{i-1}(t)$ | + | + | + |

it is necessary to determine whether or not a $p(t)$ has a root in a time interval $[a, b]$, the approach is (i) to construct a Sturm sequence from $p(t)$ using Equation (2.10) and (ii) to calculate $V(a) - V(b)$ for this sequence. If $p(a) \neq 0$, $p(b) \neq 0$ and $V(a) - V(b) = 0$, then $p(t)$ has no roots in $[a, b]$.

Generating a Sturm sequence using the recurrence relation in Equation (2.10) requires symbolic division of two polynomials. This is necessary because every polynomial in the sequence, except the first two, is the negative of the remainder that results from the symbolic division of the two previous polynomials in the sequence. Normally, symbolic division of two polynomials is an $O(d^2)$ operation. However, in Equation (2.10), the difference of the degrees of the numerator and the denominator in the division operation $\frac{p_{j-1}(t)}{p_j(t)}$ is always one. As this difference is always constant, complexity of the division operation automatically decreases to $O(d)$. Since $d - 1$ division operations have to be performed according to Equation (2.10), generating the sequence of polynomials takes $O(d^2)$ time. Calculation of $p(a)$ and $p(b)$ requires evaluating p at two timepoints and it simply takes $O(d)$ time. Calculation of $V(a) - V(b)$ is a little more complex and it requires construction of a sign table (similar to Table 2.1). There needs to be two columns in this table, one for timepoint a and one for timepoint b , and there has to be $d + 1$ rows where each row corresponds to one of the polynomials in the Sturm sequence. In other words, size of the table is $O(d)$. Each spot in the table must be filled in with a sign and those signs are found by evaluating Sturm polynomials at

timepoints a and b . Since evaluating a polynomial at a timepoint is an $O(d)$ operation, constructing this sign table and therefore calculating $V(a) - V(b)$ takes $O(d^2)$ time in total. As a result of this long discussion, no matter how complex it looks, the overall procedure of detecting the existence of a root of an arbitrary $p(t)$ in an arbitrary interval $[a, b]$ is an $O(d^2)$ operation.

3. A NEW ALGORITHM: LYQUID

3.1. The Problem

Let there be a real world system which is governed by an unknown set of differential equations. Since there are no differential equations at hand to solve quantitatively, the functions which are solutions of this unknown system of equations are also unknown. However, assume that the quantitative behavior of the variables belonging to this system are observable over time. That is, any observer, who decides to measure and sample any variety of quantities in the system, is able to do so. The problem then is to find qualitative relationships between the observed variables and possibly some unobserved (hidden) variables. These relationships make up a system of qualitative differential equations (QDE) for the observed real world system. Such a QDE is a very good starting point to understand and explain the physical behavior of the system; it may also act as a starting point for the discovery of the underlying set of ordinary differential equations (ODE), which is unknown in the first place, through parameter estimation where the QDE acts as the template for the ODE.

The main problem, even if the ODE of the observed system was known, is to find functions of time for the variables in the system. Mathematically speaking, these functions explain past and future behavior of the quantities they belong to. However, it is impossible to arrive at an analytical solution to reveal these underlying functions with the ODE being undetermined in the first place. Even if that is the case, the system still yields a lot of information to work with in determining its underlying functions. That information could easily be captured by sampling behavior of some (possibly all) of the quantities in the system over a period of time.

The idea then is to approximate the actual unknown functions with polynomials using the techniques explained in Sections 2.1 and 2.2. When the polynomials used in approximating the functions are accurate enough, where accuracy is rated as the accuracy with respect to the data samples at hand, it is possible to use these polynomials

to search for qualitative relationships between the variables. The upcoming sections describe how a new technique is built on this idea. This new algorithm is named LYQUID, which arises from a combination of the word poLYnomial with QUalitative system IDentification.

3.2. Inputs, Outputs and Initialization

LYQUID requires observed numerical data that are sampled over time as input. For the purposes of the algorithm, it is neither mandatory to have equal number of samples for each variable nor necessary to have the samples equally spaced over time. The samples, together with the timepoints of measurements, are input to the algorithm. The unit of each observed variable and a multiplication table of the specified units are also provided to the algorithm as inputs.

The output of LYQUID is a qualitative model expressed in QSIM's well-known vocabulary [3]. Considering the availability of numerical data for processing, LYQUID enhances its output by suggesting landmark values for constants and functions for monotonic relationships. In other words, it is very much possible to use LYQUID's output with semi-quantitative extensions of QSIM like Q2 [3] and Q3 [8].

Assume that n variables, which are denoted by v_i ($i = 1, \dots, n$), are observed over a time interval $[a, b]$ and supplied as inputs to LYQUID. Then, during the initialization (preprocessing) stage, LYQUID uses Algorithm 2 to find a piecewise polynomial representation for every observed variable, i.e. $v_i(t) \approx pp_i(t)$. Here $pp_i(t)$ represents a piecewise polynomial on the time interval $[a, b]$ and it divides this interval into non-overlapping sub-intervals so that $v_i(t)$ is approximated with a different polynomial in each sub-interval. Functions which are denoted by $v_i(t)$ are the real functions that were stated to be unknown in the description of the problem.

3.3. Units

Units of variables are important for LYQUID during the process of model discovery. Units, on the other hand, are not known by LYQUID in their physical meanings or notations. They are identified by *group.order* integer pairs. Units with the same *group* field are time derivatives of one another; *order* field determines the order in time. For example, the units m , m/s and m/s^2 could be identified by 1.1, 1.2 and 1.3 respectively. The *group* field indicates a group of units and its value is not important as long as it is unique for every different group. In this representation of units, two units are equal only when both of their *group* and *order* fields are equal.

The unit of every observed variable needs to be provided as input to LYQUID. These units are decided by the runner of the algorithm; the user is supposed to group related units together if possible and assign proper orders of time inside these groups. If the user has no information about the units of the observations, every variable could be assigned a unit of a different group but then LYQUID might be unable to discover even some trivial relationships. Although it is dependent on the input initially, the algorithm then becomes capable of determining units of automatically generated variables, which are created from the observed variables.

When LYQUID takes the derivative of an existing variable to obtain a new variable, the unit of the new variable is easily found by increasing time order of the unit of the original variable. This relative ease is in fact why such a representation scheme for units was selected in the first place. When addition or subtraction operations are used in creating a new variable, the new variable definitely has the same unit with the operands. The only tricky operation in this respect is the multiplication operation. Recall that LYQUID needs a multiplication table of units as an input. This multiplication table is basically a list of tuples, where every tuple is made of two operand units and a resulting unit. For example, if 1.1 corresponds to kg , 2.3 corresponds to m/s^2 and 3.1 corresponds to N as specified by the user in the input, the user ought to include the tuple (1.1, 2.3, 3.1) in the multiplication table. When the multiplication table is more informative, LYQUID is able to generate more new variables from observed variables

according to these multiplication definitions.

3.4. Piecewise Polynomial Operations

Performing operations on polynomials, as described in Section 2.3, is very easy. The polynomial representations in LYQUID, however, are possibly piecewise polynomials. This is not actually a problem for the unary operations like differentiation. The solution for the unary piecewise polynomial operations is very simple; it is to apply the required operation to every sub-polynomial without changing the time intervals they are defined on.

On the other hand, the solution for the binary operations is a little more tricky. An example illustrates the solution:

$$\begin{aligned}
 p_1(t) &= \begin{cases} t + 2 & 0 \leq t < 2 \\ t^2 & 2 \leq t < 5 \end{cases} \\
 p_2(t) &= \begin{cases} t^2 - 1 & 0 \leq t < 3 \\ t + 5 & 3 \leq t < 5 \end{cases} \\
 d(t) = p_1(t) - p_2(t) &= \begin{cases} -t^2 + t + 3 & 0 \leq t < 2 \\ 1 & 2 \leq t < 3 \\ t^2 - t - 5 & 3 \leq t < 5 \end{cases}
 \end{aligned}$$

Apparent from the example, the idea is to first find out all the transition timepoints in the piecewise polynomials that are being operated on. The resulting piecewise polynomial must have transitions from one polynomial to another at all of those timepoints. Once the sub-intervals are determined using the transition points, calculating coefficients of the resulting polynomial is as simple as doing the ordinary operation.

The error criterion in Equation (2.8) is defined for two ordinary polynomials p and q . In fact, since polynomial operations are extended to span multiple time intervals, this error criterion can also be extended for comparing two piecewise polynomials. Recall that calculation of e , which is the square error per unit time, requires calculation of a

difference of two polynomials, followed by a squaring operation on the difference and finally an integration of the squared polynomial. When p and q in the definition are extended to be piecewise polynomials, their difference is a new piecewise polynomial defined on multiple time intervals. Difference of p and q must be calculated as it was done in the above example while subtracting $p_2(t)$ from $p_1(t)$. Once this difference is calculated, since squaring and integration are unary operations (they have only one polynomial as their operand), calculating e is as simple as doing the remaining operations. For $p_1(t)$ and $p_2(t)$ in the example above, e on the time interval $[0, 5)$ turns out to be:

$$\begin{aligned}
 d(t) &= p_1(t) - p_2(t) \quad (\text{previously calculated}) \\
 [d(t)]^2 &= \begin{cases} t^4 - 2t^3 - 5t^2 + 6t + 9 & 0 \leq t < 2 \\ 1 & 2 \leq t < 3 \\ t^4 - 2t^3 - 9t^2 + 10t + 25 & 3 \leq t < 5 \end{cases} \\
 \int_a^b [d(t)]^2 dt &= \begin{cases} \frac{1}{5}t^5 - \frac{1}{2}t^4 - \frac{5}{3}t^3 + 3t^2 + 9t & 0 \leq t < 2 \\ t & 2 \leq t < 3 \\ \frac{1}{5}t^5 - \frac{1}{2}t^4 - 3t^3 + 5t^2 + 25t & 3 \leq t < 5 \end{cases} \\
 S.E. &= \int_0^5 [d(t)]^2 dt = \int_0^2 [d(t)]^2 dt + \int_2^3 [d(t)]^2 dt + \int_3^5 [d(t)]^2 dt \\
 &= 15.0\bar{6} + 0 + 140.4 = 155.4\bar{6} \\
 e &= \frac{S.E.}{5} = 31.09\bar{3}
 \end{aligned}$$

Looking at this value of e , $p_1(t)$ and $p_2(t)$ are not even close to being equal on $[0, 5)$. These two piecewise polynomials are plotted in Figure 3.1.

3.5. Generation of Hidden Variables

Most of the real world qualitative models involve relationships between observable variables and some hidden variables. Discovering those variables and such relationships is the essence of learning qualitative models. This section explains how LYQUID generates hidden variables from polynomial representations of observed variables. Note that, among the generated variables, only the ones which are involved in the discovered

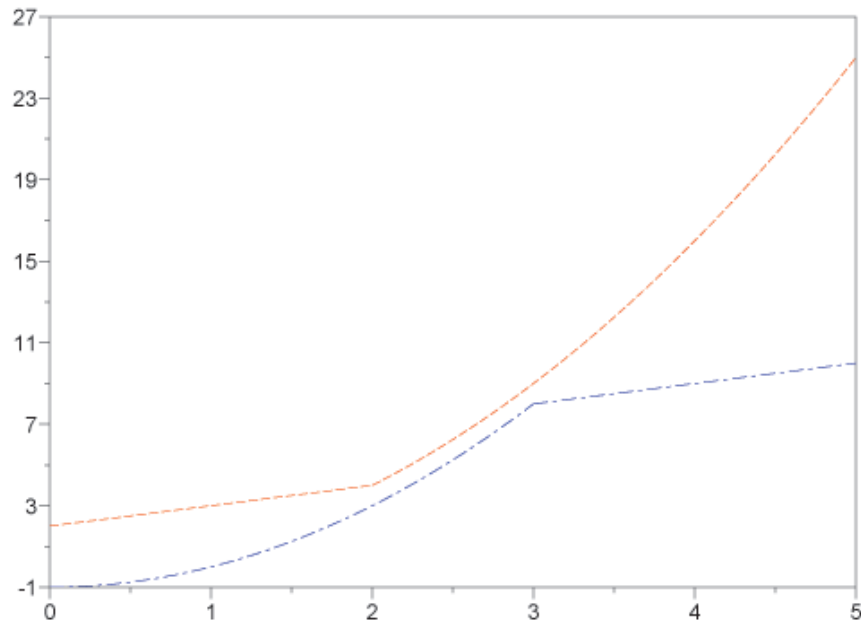


Figure 3.1. $p_1(t)$ (dash) and $p_2(t)$ (dash dot)

constraints are included in the output model.

First of all, LYQUID generates derivatives of the observed variables by differentiating their corresponding piecewise polynomials. The algorithm has a derivative depth parameter, which is denoted by o , and this parameter determines the depth of the highest order derivative that will be introduced as a new variable. To make the parameter meaningful, $o \leq d$ is expected to hold; remember that the degree of the polynomials used in approximations is set to be at most d and the $(d + 1)^{st}$ order derivative of such a polynomial is always zero. After this first step of hidden variable generation, a chain of derivatives for every observed variable is added to LYQUID's list of variables. The length of these chains is at most o but they could also be shorter. That is because whenever a constant piecewise polynomial is found in the derivative chain, LYQUID breaks the chain and stops seeking into higher order derivatives.

Composites are then generated as products of sums. LYQUID allows only a single appearance of an observed variable or one of its derivatives in a composite variable. Moreover, another parameter $c \leq n$ is used to limit the size of composite variables, where n is the number of observed variables. Setting $c > n$ is equivalent to

setting $c = n$ because of the single appearance limitation. When generating sums and differences, all possible combinations of variables with matching units are taken into consideration. When generating products as composites, only the ones whose units could be determined from the multiplication table are created.

Assume that, as an example, a , b , x and y are observed variables with units 1.1, 1.1, 1.2 and 2.1 respectively. Assume also that the multiplication table has a single entry, which is (1.1, 1.1, 2.1). In that case, after generation of hidden variables, the list of variables maintained by LYQUID will look like Table 3.1 under the final assumption that $o = 1$ and $c = 3$. Even though they are not shown in the table, automatically

Table 3.1. An example list of variables maintained by LYQUID

| Variable | Unit | Variable | Unit |
|----------|------|-------------------|------|
| a | 1.1 | $a \pm b$ | 1.1 |
| b | 1.1 | $x \pm da$ | 1.2 |
| x | 1.2 | $x \pm db$ | 1.2 |
| y | 2.1 | $da \pm db$ | 1.2 |
| da | 1.2 | $x \pm da \pm db$ | 1.2 |
| db | 1.2 | $a \cdot b$ | 2.1 |
| dx | 1.3 | | |
| dy | 2.2 | | |

generated composites are assigned names starting with “auto”. The \pm signs indicate that both the sum and the difference exists in the variable list; that is, for example, $x \pm da \pm db$ represents four different variables. These four variables, by the way, would have been left out of the variable list if c was set to 2. Note that $a \cdot (a + b)$ and $b \cdot (a + b)$ are two other possible composite variables but they are ruled out because of the single appearance limitation. Finally, although $y \pm a \cdot b$ is another variable that could be allowed according to the given information, it cannot be generated by LYQUID when product of sums approach is used as the generation method.

Using products of sums approach or allowing only a single appearance of base

variables in composites variables are design decisions. They are based on the assumption that most of the hidden variables in real world models could be generated with the described methodology. There are similar algorithms in literature that prefer other methodologies; an algorithm (LAGRANGE) even generates new variables by applying sin and cos functions to observed variables [9]. Since the size of the hidden variable space is very large, it is necessary to make some assumptions at some point in designing the algorithm which will rule out some possibilities of hidden variables. LYQUID's design decision about hidden variable generation, for example, misses out certain hidden variables in real world models. Among them are variables of mechanical energy. Recall that kinetic energy is proportional with the square of a moving body's velocity. Even though the total mechanical energy in a dynamics system is constant, LYQUID would not be able to make that discovery as it is not capable of creating square terms due to the single appearance limitation. Even this example shows that LYQUID cannot claim to discover all hidden variables in a given system; it only claims to discover models that follow some particular pattern.

At the time of constraint discovery, the size of the set of variables plays an important role in the complexity of the procedure. The problem here is to determine the number of variables after hidden variable generation when starting with n observed variables. After generation of the derivatives, it is apparent that the number of variables is $(o + 1)n$. In the worst case, all of the observed variables have a common unit and $c = n$. In this case, the variables with the same units create a total of $(o + 1)$ groups of size n . Since $c = n$, all possible combinations of sums of variables among these groups are allowed. So, in the worst case, the number of variables after generation of sums is

$$\begin{aligned}
 V_{sums} &= (o + 1) \sum_{k=1}^n \binom{n}{k} 2^{k-1} \\
 &= \frac{1}{2}(o + 1) \sum_{k=1}^n \binom{n}{k} 2^k \\
 &= \frac{1}{2}(o + 1) \left(\left[\sum_{k=0}^n \binom{n}{k} 2^k \right] - 1 \right) = \frac{1}{2}(o + 1) [(1 + 2)^n - 1] \\
 &= \frac{1}{2}(o + 1) (3^n - 1).
 \end{aligned} \tag{3.1}$$

The first line in Equation (3.1) constructs k sized sums among the $(o + 1)$ groups, varying k from 1 to n . The term 2^{k-1} ensures that all possible sign combinations between the k variables are counted; there are $(k - 1)$ signs to be placed in between k variables and the sign is either a plus or a minus. All of the remaining computation is simple arithmetic, including the replacement of the summation with a simpler exponential; note that the replaced summation is the binomial expansion of $(1 + 2)^n$.

When $c < n$, the upper bound of the summation in Equation (3.1) is c rather than n and the terms making up the sum are polynomials of n whose degrees range from 1 to c . In other words, when $c < n$, the number of variables could be written with a tighter polynomial bound: $O(n^c(o + 1))$. Note that c is constant and the exponential inside the summation disappears from complexity because $O(2^c)$ is $O(1)$.

It is not as easy to put an estimate on the number of variables after the final stage, which is the generation of products. That is because the multiplication table of units determines this number and it is not possible to make general worst case assumptions to come up with an exact value. However, assuming that the multiplication table allows multiplication of a sequence of at most m units, it is definite that the size of the final set of variables is $O((n^c(o + 1))^m)$. Note that when $c = n$, the size of the set of variables is inevitably exponential in n .

3.6. Discovering Qualitative Constraints

Once the hidden variables have been generated as explained in the previous section and the list of variables has been completed, LYQUID is supposed to discover qualitative relationships, which are expressed in the QSIM vocabulary, within this set of variables. This discovery process is nothing more than an equality check on two piecewise polynomials, based on the extended error criterion described in Section 3.4. Each type of qualitative constraint requires a different preparation prior to this equality check:

eq This constraint type is actually not used in QSIM formalism. It is a binary con-

straint, which is used by LYQUID to state that two variables are equal. It is very much possible for any two variables in LYQUID's variable list to be equal to each other within the tolerated error margin. Therefore, rather than trying to find a workaround with other QSIM constraint types to express such equalities, LYQUID uses eq constraints. Whenever it needs to check equality of two variables, LYQUID directly compares their piecewise polynomials according to the square error criterion for polynomials. Equality of units is the prerequisite for performing the equality check.

minus This constraint type is almost the same with eq constraint. The only difference is that when testing existency of a minus constraint, LYQUID creates an intermediate piecewise polynomial by multiplying one of the variables' piecewise polynomial by -1 . The equality check of piecewise polynomials is then carried out between the other variable's piecewise polynomial and the intermediate one. Units of the involved variables must match.

const Each of the sub-polynomials of a variable has a constant term. When checking if a variable is constant or not, LYQUID first creates an intermediate piecewise polynomial, which makes transitions at the same timepoints with the variable's piecewise polynomial, using these constant terms. The decision is then made by comparing the original piecewise polynomial with the intermediate piecewise polynomial. An example is given below:

$$p_{orig}(t) = \begin{cases} t + 2 & 0 \leq t < 2 \\ t^2 - t + 1 & 2 \leq t < 4 \end{cases}$$

$$p_{int}(t) = \begin{cases} 2 & 0 \leq t < 2 \\ 1 & 2 \leq t < 4 \end{cases}$$

If LYQUID decides that the original and the intermediate piecewise polynomials are equal according to its error criterion, it assigns the mean of all the constant terms as the landmark value for the newly discovered constant constraint.

d/dt When checking existence of a constraint like $d/dt(a \ b)$, LYQUID generates the intermediate piecewise polynomial by differentiating a 's piecewise polynomial with respect to time. The problem then reduces to checking whether this intermedi-

ate piecewise polynomial is equal to b 's or not. Units of b and the intermediate variable must also be the same.

add In order to conclude that an $\text{add}(a\ b\ c)$ constraint exists, LYQUID (*i*) requires that units of the three variables be the same and (*ii*) an intermediate piecewise polynomial, which is created by adding a and b , is equal—within tolerated limits—to c 's piecewise polynomial.

mult In order to conclude that a $\text{mult}(a\ b\ c)$ constraint exists, LYQUID (*i*) requires that the unit of a times b can be determined from the multiplication table, (*ii*) the determined unit is equal to c 's unit and (*iii*) an intermediate piecewise polynomial, which is created by multiplying a by b , is equal to c 's piecewise polynomial according to the square error criterion.

M⁺, M⁻ First of all, any two variables can be involved in a monotonic function constraint regardless of their units. Checking existence of an $M^{+,-}(x\ y)$ consists of two steps. The first step ensures that x and y have the same or inverse directions of change and the second step ensures that there exists a function f such that either $x = f(y)$ or $y = f(x)$. During the first step, LYQUID computes time derivatives of both x and y and then calculates an intermediate piecewise polynomial which is the product of $\frac{dx}{dt}$ and $\frac{dy}{dt}$. When x and y have the same directions of change for all of the observation interval, i.e. $M^+(x\ y)$, this intermediate polynomial has to be positive at all timepoints. For the case of an $M^-(x\ y)$, the intermediate piecewise polynomial has to be negative at all timepoints. It is possible to check if a piecewise polynomial is positive/negative on an arbitrary time interval with the technique described in Section 2.5: for every sub-polynomial that make up the piecewise polynomial, the sub-polynomial must be positive/negative at the endpoints of the interval on which it is defined and it must not have any zeroes inside that interval. LYQUID allows the user to make the definition of the words positive/negative through a non-negative parameter called *mzero*. During the first step, positive for LYQUID is the interval $(-mzero, +\infty)$ and negative is the interval $(-\infty, mzero)$. LYQUID becomes less strict with the direction of change requirement of the $M^{+,-}$ constraints when greater values of *mzero* are used. *mzero* = 0 makes LYQUID perfectly strict.

If the first stage results successfully, LYQUID moves onto the second stage. The

second phase requires discovery of a function f such that either $x = f(y)$ or $y = f(x)$. This unknown function f could also be approximated by polynomials just like x and y . Similar to the d parameter, which determines the maximum degree of polynomials used in approximating observed variables, there is an S parameter of LYQUID that puts an upper bound on the degrees of the polynomials used in monotonic function approximations. Assuming a transformation from $x(t)$ to $y(t)$ is sought for, the problem is simple: find the polynomial $f(t)$, whose degree is S , such that $E(t) = [y(t) - f(x(t))]^2$ is minimized. When $f(t)$ is expressed as $f(t) = \sum_{i=0}^S c_i t^i$, where c_i are the unknown coefficients, E becomes

$$E(t) = \left[y(t) - \sum_{j=0}^S c_j [x(t)]^j \right]^2$$

Taking the derivative of both sides of the equation with respect to c_i ($i = 0, 1, \dots, S$) and equating to zero solves the optimization problem:

$$\frac{\partial E(t)}{\partial c_i} = 2 [x(t)]^i \left[y(t) - \sum_{j=0}^S c_j [x(t)]^j \right] = 0$$

However, note that $\frac{\partial E(t)}{\partial c_i}$ is dependent on time and therefore it has to be integrated over the observation time interval $[a, b]$ so that the normal equations are free of the independent time variable t :

$$\begin{aligned} \int_a^b \frac{\partial E(t)}{\partial c_i} dt &= \int_a^b [x(t)]^i y(t) dt - \sum_{j=0}^S c_j \int_a^b [x(t)]^{i+j} dt = 0 \\ \int_a^b [x(t)]^i y(t) dt &= \sum_{j=0}^S c_j \int_a^b [x(t)]^{i+j} dt \quad (i = 0, 1, \dots, S) \end{aligned}$$

As the piecewise polynomials $x(t)$, $y(t)$ and the timepoints a , b are known, the final equation produces a linear system of equations with $S + 1$ equations and $S + 1$ unknowns. It is possible to find out the coefficients of the hypothesized monotonic function f by solving this system of equations. The overall procedure of computing an f with this approach takes $O(Sd^2 + S^2)$ time. $O(Sd^2)$ results

from calculating values of a total of $3S + 2$ definite integrals, which require performing polynomial operations of $O(d^2)$ complexity. $O(S^2)$ results from solving the linear system of $S + 1$ equations with $S + 1$ unknowns.

The first intuition might be to set S equal to d . However, the degree of $y(t)$ is d while the degree of $f(x(t))$ is $(S \cdot d)$. As it is explained in the next paragraph, these two polynomials are compared for equality before a monotonic function constraint is added to the output model. When degrees of the compared polynomials are very different and one of the polynomials is very dominant with relatively high order terms, the comparison procedure becomes numerically unstable. For example, when $S = d = 5$, LYQUID would have to compare two polynomials whose degrees are 5 and 25; square of their difference would be another polynomial whose degree is 50 and the definite integral of this polynomial over a finite interval would definitely exceed tolerated error, resulting in the rejection of the monotonic function constraint. Therefore, experience tells that it is necessary to fix S as constant at 1, 2 or at most 3 to get good results. This also keeps computational complexity free of S .

The final step after computing f is to compute the piecewise polynomial $f(x(t))$ to obtain an intermediate piecewise polynomial and to compare it with $y(t)$ using the usual error criterion. If this comparison tells that $y(t)$ and $f(x(t))$ are equal piecewise polynomials within the tolerated error margin, then LYQUID concludes that an $M^{+,-}(x, y)$ exists, with the $+$ or $-$ being decided at the first phase. LYQUID provides the polynomial $f(t)$ as the hypothesized monotonic function in its output.

It is possible to disable LYQUID's second phase control over potential monotonic function constraints. In that case, it decides about monotonic function constraints based only on the direction of change control.

3.7. The Algorithm in Pseudo-Code

This section makes a brief summary of all the discussion about LYQUID by specifying the algorithm in pseudo-code as Algorithm 3. Table 3.2 is a list of symbols that appear in Algorithm 3 and their meanings. Table 3.3 is a list of symbols that

implicitly exist within Algorithm 3 without being mentioned explicitly.

Table 3.2. Symbols appearing in Algorithm 3

| | |
|--|---|
| n | Number of observed variables |
| $u^{(k)}$ | Unit of the k^{th} observed variable |
| $N^{(k)}$ | Number of samples of the k^{th} observed variable |
| $\left\{t_i^{(k)} x_i^{(k)}\right\}_{i=1}^{N^{(k)}}$ | Timepoint–value pairs belonging to the k^{th} observed variable |
| o | Order of the highest order derivative to generated by LYQUID |
| c | Maximum number of variables allowed in a composite variable |
| M | Multiplication table of units |

Table 3.3. Symbols that exist implicitly inside Algorithm 3

| | |
|-------------------------|--|
| d | Maximum degree of polynomials used in approximations |
| <i>Tolerance</i> | Square error of the approximation that is tolerated at maximum, the point at which no further split is required |
| <i>Improvement</i> | Change in total square error that is considered as improvement after splitting a time interval |
| <i>SplitImprovement</i> | Change in total square error in a sub–interval that is considered as improvement after splitting a time interval |
| <i>iTolerance</i> | Square error per unit time that is considered to make two piecewise polynomials equal when comparing them |
| <i>mzero</i> | Positive real number that is used to define the words positive and negative in monotonic function checks |
| S | Maximum degree of polynomials used in suggesting functions for monotonic relationships |

Algorithm 3 makes references to subroutines whose names start with *Generate*. They are related to the generation of hidden variables, which is discussed in Section 3.5. There are also references to subroutines whose names start with *Check*. The way how these subroutines work is explained in Section 3.6 in detail. They return a set of constraints if a qualitative relationship is found between the input variables and an empty set otherwise. The discovered qualitative relationships are appended to a list

of constraints, which contains the qualitative model of the observed system once the algorithm stops.

When a constraint is newly discovered by one of the *Check* subroutines and the new constraint involves an automatically generated hidden variable, LYQUID appends the qualitative constraints, which are necessary to create the automatic variable, to the list of constraints. This makes sure that the qualitative constraints that result in creation of a new automatic variable do not appear in the output qualitative model when the new variable is not bound to the model with a different constraint. LYQUID also makes sure not to append artificial discoveries to the list of constraints. For example, if the variable *auto1* is generated as the sum of two observed variables x and y , *CheckSum*($x, y, auto1$) would return a valid qualitative relationship without LYQUID's internal redundancy checking.

3.8. Overall Complexity

Since variables are approximated by piecewise polynomials, LYQUID has to test existence of constraints on multiple time intervals. Assuming that piecewise polynomials consist of k sub-polynomials at maximum, k has to appear as a linear factor in the complexity of the constraint discovery stage. Therefore, it is important to put an upper bound on k . In the general case, it is impossible to estimate k precisely; however, it is possible to put a bound on k using the maximum number of extrema taken on by any one of the observed variables.

Assume that one of the observed variables attains at most x local extrema during the observation interval $[a, b]$. Then, it is possible to say that every observed variable consists of at most $x + 1$ monotonically increasing or decreasing sections, which connect consecutive local extrema. Fitting polynomials on such increasing or decreasing sections generally yields very good approximations. Therefore it is reasonable to conclude that k is $O(x)$ if not exactly $x + 1$.

Algorithm 2 runs at the initialization stage for every observed variable. It is

Algorithm 3 LYQUID

 $V \leftarrow \emptyset$ (set of variables)

for $k = 1$ to n **do**
 $ppoly \leftarrow$ new piecewise polynomial fitted to $\{t_i^{(k)}, x_i^{(k)}\}_{i=1}^{N^{(k)}}$ (using Algorithm 2)

 $var \leftarrow$ new variable, whose unit is $u^{(k)}$, represented by $ppoly$
 $V \leftarrow V \cup var$
end for
 $V \leftarrow V \cup \text{GenerateDerivatives}(V, o)$
 $V \leftarrow V \cup \text{GenerateSums}(V, c)$
 $V \leftarrow V \cup \text{GenerateProducts}(V, c, M)$
 $C \leftarrow \emptyset$ (list of constraints)

for all $(v_1, v_2) \in V$ **do**
 $C_{temp} \leftarrow \text{CheckEquality}(v_1, v_2)$
if $C_{temp} \neq \emptyset$ **then**
 $V_{prune} \leftarrow v_2 \cup \{v \mid v \text{ is derived from } v_2 \text{ or involves } v_2\}$
 $V \leftarrow V - V_{prune}$
 $C \leftarrow C \cup C_{temp}$
end if
end for
for all $v \in V$ **do**
 $C \leftarrow C \cup \text{CheckConstant}(v)$
end for
for all $(v_1, v_2) \in V$ **do**
 $C \leftarrow C \cup \text{CheckDerivative}(v_1, v_2)$
 $C \leftarrow C \cup \text{CheckMonotonic}(v_1, v_2)$
end for
for all $(v_1, v_2, v_3) \in V$ **do**
 $C \leftarrow C \cup \text{CheckSum}(v_1, v_2, v_3)$
 $C \leftarrow C \cup \text{CheckProduct}(v_1, v_2, v_3)$
end for

this algorithm which calculates piecewise polynomials for all of the observed variables. Since $O(x)$ is put as an upper bound for the number of sub-polynomials that make up piecewise polynomials, the outer-most while loop of Algorithm 2 will run $O(x)$ times. Algorithm 1, which is the polynomial fitting algorithm, is used as a subroutine inside this while loop and it takes $O(N d^2)$ time. Then, overall complexity of the initialization stage (the first loop in Algorithm 3) becomes $O(n x N d^2)$.

Recall that the size of the set of variables after generation of hidden variables is $O((n^c(o+1))^m)$. Since hidden variables are generated by operating on piecewise polynomials, generation of each hidden variable takes $O(x d^2)$ time. Therefore, complexity of this stage of the algorithm is $O((n^c(o+1))^m x d^2)$.

Complexity of the model discovery stage depends on the size of the set of variables. If v represents the number of variables prior to constraint discovery, then there could be at most $O(v)$ unary, $O(v^2)$ ternary and $O(v^3)$ ternary constraints that need to be tested. Existence of constraints are tested by operating on piecewise polynomials, which take $O(x d^2)$ time even for the monotonic function constraints. Therefore, even though the dominant term arises from the binary constraints and actual complexity is much lower because of unit checking, it is safe to express complexity of the discovery stage as $O(v^3 x d^2)$. As a result, the overall complexity of LYQUID is

$$O(n x N d^2 + (n^c(o+1))^{3m} x d^2).$$

4. EXPERIMENTAL RESULTS

LYQUID was tested on the u-tube, cascaded tanks and coupled tanks models which are used as benchmark systems in related literature [3, 10]. Figure 4.1 shows these physical systems and the figures are taken from Coghill et. al. [11]. The numerical data required for the experiments were obtained by simulating physically realistic systems of the three kinds. Apart from the clean data, noisy datasets were obtained by adding Gaussian noise over the original samples and LYQUID has been tested on both clean and noisy data samples. Other types of experiments involve datasets in which, (i) samples are obtained with different or irregular sampling frequencies, (ii) there are time intervals where some variables are not sampled at all.

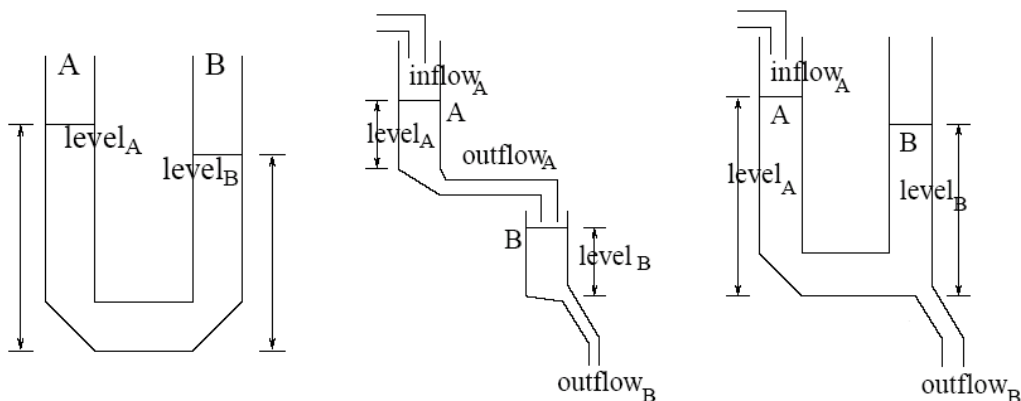


Figure 4.1. Benchmark models: u-tube, cascaded tanks and coupled tanks

4.1. Experiments on Clean Data

4.1.1. U-tube

U-tube data used in the experiments belong to a cylindrical u-tube with 1.25 and 0.8 units of radii. Initial heights of the liquid columns in the tubes were 2.5 and 0.5 units respectively and the levels were sampled over time with 0.01 second intervals for a total of 3 seconds. Recall at this point that all u-tubes obey the following qualitative model:

$d/dt(level_A, rise_A)$
 $d/dt(level_B, rise_B)$
 $add(level_B, levelDiff, level_A)$
 $M^-(levelDiff, rise_A) (0, 0)$
 $M^-(rise_A, rise_B) (0, 0)$.

Table 4.1. Discoveries from the clean u-tube dataset

| | | |
|-----------------------|-------------------|--------------------|
| $d/dt(A \ dA)$ | | |
| $d/dt(B \ dB)$ | | |
| $add(A \ B \ auto1)$ | | |
| $minus(B \ -B)$ | | |
| $add(A \ -B \ auto2)$ | | |
| $M^-(A \ B)$ | (2.7048 0.0000) | (0.0000 6.6033) |
| $M^-(A \ dA)$ | (1.9177 0.0000) | (0.0000 97.3223) |
| $M^-(A \ auto1)$ | (4.5813 0.0000) | (0.0000 6.6035) |
| $M^+(A \ auto2)$ | (1.9188 0.0000) | (0.0000 -6.6034) |
| $M^+(B \ dA)$ | (1.9217 0.0000) | (0.0000 -2.4451) |
| $M^+(B \ auto1)$ | (-4.5811 0.0000) | (0.0000 2.7048) |
| $M^-(B \ auto2)$ | (1.9188 0.0000) | (0.0000 2.7048) |
| $M^-(dA \ dB)$ | (0.0000 0.0000) | |
| $M^+(dA \ auto1)$ | (-99.6365 0.0000) | (0.0000 3.8394) |
| $M^-(dA \ auto2)$ | (-0.0121 0.0000) | (0.0000 -0.0040) |
| $M^-(dB \ auto1)$ | (243.2531 0.0000) | (0.0000 3.8394) |
| $M^+(dB \ auto2)$ | (0.0297 0.0000) | (0.0000 -0.0040) |
| $M^-(auto1 \ auto2)$ | (3.8377 0.0000) | (0.0000 9.1624) |
| $M^+(A \ dB)$ | (1.9177 0.0000) | (0.0000 -237.5864) |
| $M^-(B \ dB)$ | (1.9217 0.0000) | (0.0000 5.9695) |

Using fifth degree polynomials for approximations and 6×10^{-5} as *iTolerance*, LYQUID discovered the constraints given in Table 4.1. Although the listed constraints are numerous when compared to the general u-tube model, all of the listed monotonic function constraints are correct for the specific numerical dataset. The functional relationships suggested for the monotonic function constraints are not provided but the landmark pairs given in the table are the intercepts of those unlisted functions.

Note that the hidden variables $rise_A$, $rise_B$ and $levelDiff$ are discovered as dA , dB and $auto2$ respectively; the two monotonic function constraints which bind these hidden variables are also discovered. One of the most important things about this output model is the discovery of the landmark value pairs, which are located at the origin, for the constraints $M-(dA\ dB)$ and $M-(dA\ auto2)$. Those two landmark value pairs are necessary to end qualitative simulation of the output model correctly at an equilibrium point where the levels in the tubes become equal. Being able to capture this very detailed and correct qualitative model from only two observed variables is remarkable.

4.1.2. Cascaded Tanks

In a cascaded tanks system, there are two separate tanks with holes at their bases. There is an inflow of liquid into the first tank and outflow from this first tank pours into the second tank. The most general qualitative model for a cascaded tanks system is the following:

$$\begin{aligned}
 & d/dt(amount_A, netflow_A) \\
 & d/dt(amount_B, netflow_B) \\
 & add(outflow_A, netflow_A, inflow) \\
 & add(outflow_B, netflow_B, outflow_A) \\
 & M^+(amount_A, outflow_A) (0, 0) \\
 & M^+(amount_B, outflow_B) (0, 0)
 \end{aligned}$$

The dataset for the cascaded tanks experiment was created by simulating a cascaded tanks system with two cylindrical tanks whose radii are 1 and 0.75 units. The initial levels of liquid in the tanks were 1 units each and $amounts$ of liquid in the tanks were sampled over time with 0.01 second intervals for a total of 4 seconds. The Inflow of liquid into the first tank, which is the exogenous variable, was taken to be $e - e^{-t}$. The unit of inflow was specified as the derivative of the unit of amounts. The sampled amounts and inflow were supplied as input to the algorithm.

LYQUID was tested on the cascaded tanks dataset with fifth and eleventh degree polynomials separately. In both cases $iTolerance$ was set to 7×10^{-5} . The results that are listed in Tables 4.2 and 4.3 show that the hidden variables $netflow_A$, $netflow_B$,

$outflow_A$ and $outflow_B$ are discovered as `dA`, `dB`, `auto4` and `auto16` respectively in both cases. The monotonic function relationships between the amounts and outflows are also discovered. Considering nature of the exponential function that is used as inflow, the monotonic function constraint that relates inflow and its derivative is also correct. Inspection of the simulation data revealed that the three other monotonic function constraints are also correct discoveries. It is again remarkable to see in Table 4.3 the landmark value pairs that are located at the origin for the constraints `M+(A auto4)` and `M+(B auto16)`. Similar to the ones in the u-tube case, discovery of these value pairs are important because they basically tell the qualitative simulator that outflow of liquid from an empty tank is not possible.

An important observation when going from Table 4.2 to Table 4.3 is the increase in the accuracy of landmark pairs when eleventh degree polynomials are used. Not only does the accuracy of LYQUID's numerical suggestions improve but also LYQUID discovers another true monotonic function constraint which was left unrevealed otherwise. This improvement is obviously caused by the increase of the accuracy of the polynomial approximations. Some monotonic function constraints given in Tables 4.2 and 4.3 as well as Table 4.1 are redundant in the presence of some other ones. Even though they are redundant, LYQUID retains them in its output because extra landmark pairs provided with them might be necessary to eliminate possible spurious behaviors from the qualitative simulation of the output model.

It should also be noticed that one of the monotonic function constraints lacks landmark value pairs in both Tables 4.2 and 4.3; this is because the intercepts of the discovered function are inconsistent with the M^- . Such an inconsistency for an M^- occurs when two value pairs like $(+, 0)$ and $(0, -)$, or $(-, 0)$ and $(0, +)$ are discovered for the function. It is not possible for an M^- function to possess such intercepts. The same inconsistency might apply to M^+ functions as well, if the discovered intercepts are $(+, 0)$ and $(0, +)$, or $(-, 0)$ and $(0, -)$. The only way to cope with such inconsistency is either to remove the monotonic function discovery altogether or to remove the landmark value pairs leaving the function alone. LYQUID prefers the latter.

Table 4.2. Discoveries from the clean cascaded tanks dataset using 5th degree polynomials

| | | |
|-----------------------|------------------|-----------------|
| d/dt(A dA) | | |
| d/dt(B dB) | | |
| d/dt(I dI) | | |
| add(A B auto1) | | |
| add(I dA auto3) | | |
| minus(dA -dA) | | |
| add(I -dA auto4) | | |
| minus(dB -dB) | | |
| add(I -dA -dB auto16) | | |
| M+(A auto4) | (-0.1082 0.0000) | (0.0000 1.2634) |
| M-(B auto3) | (1.7135 0.0000) | (0.0000 2.1635) |
| M+(B auto16) | (-0.0625 0.0000) | (0.0000 1.2885) |
| M-(I dI) | (2.7181 0.0000) | (0.0000 2.6350) |
| M-(dA auto1) | (1.0462 0.0000) | (0.0000 1.8080) |
| M-(auto3 auto16) | | |

Table 4.3. Discoveries from the clean cascaded tanks dataset using 11th degree polynomials

| | | |
|-----------------------|-----------------|-----------------|
| d/dt(A dA) | | |
| d/dt(B dB) | | |
| d/dt(I dI) | | |
| add(A B auto1) | | |
| minus(B -B) | | |
| add(A -B auto2) | | |
| add(I dA auto3) | | |
| minus(dA -dA) | | |
| add(I -dA auto4) | | |
| minus(dB -dB) | | |
| add(I -dB auto6) | | |
| add(I -dA -dB auto16) | | |
| M+(A auto4) | (0.0001 0.0000) | |
| M-(B auto3) | (1.7133 0.0000) | (0.0000 2.1838) |
| M+(B auto16) | (0.0000 0.0000) | |
| M-(I dI) | (2.7183 0.0000) | (0.0000 2.7183) |
| M-(dA auto1) | (1.0606 0.0000) | (0.0000 1.8085) |
| M-(auto2 auto6) | (3.6566 0.0000) | (0.0000 3.3998) |
| M-(auto3 auto16) | | |

4.1.3. Coupled Tanks

In a coupled tanks system, two tanks are interconnected with a horizontal pipe; there is an inflow of liquid to one of the tanks and there is an outflow of liquid from the other tank. The system is very similar to the u-tube and the only difference between them is that one of the tubes in the coupled tanks has a drain at the bottom. The exogenous inflow is necessary to bring the system to an equilibrium point where there is still some water inside the tubes. A general qualitative model for the described coupled tanks system is the following one:

$d/dt(\text{amount}_A, \text{netflow}_A)$
 $d/dt(\text{amount}_B, \text{netflow}_B)$
 $\text{add}(\text{flow}_{AB}, \text{netflow}_A, \text{inflow})$
 $\text{add}(\text{outflow}_B, \text{netflow}_B, \text{flow}_{AB})$
 $\text{add}(\text{level}_B, \text{levelDiff}, \text{level}_A)$
 $M^+(\text{amount}_B, \text{outflow}_B) (0, 0)$
 $M^+(\text{amount}_A, \text{level}_A) (0, 0)$
 $M^+(\text{amount}_B, \text{level}_B) (0, 0)$
 $M^+(\text{levelDiff}, \text{flow}_{AB}) (0, 0)$

It is also possible to write the same model using pressures as the hidden variables instead of levels. Notice that, be it levels or pressures, these variables are introduced to the model through monotonic function constraints. Since LYQUID introduces hidden variables only through differentiation, addition or multiplication, without observing either the levels or pressures, it is not possible to make LYQUID extract this general model. This is why both levels and amounts of liquid in the tanks were sampled during the generation of the coupled tanks dataset.

The dataset was created by simulating a coupled tanks system with two cylindrical tanks whose radii are 1 and 0.8 units. The tanks were initially empty. Amounts and levels of water in the tanks were sampled over time with 0.01 second intervals for a total of 10 seconds. The exogenous inflow variable was taken to be constant. Unit of inflow was specified as the derivative of the unit of amounts and unit of levels was specified as a different unit. Amount, level and inflow samples were input to LYQUID.

Using ninth degree polynomials for approximations and 5×10^{-5} as $iTolerance$, LYQUID discovered a total of 276 monotonic function constraints involving 24 different variables. Four of these 24 variables are the observed amounts and levels; the remaining 20 variables are the hidden variables generated by LYQUID. Those hidden variables include netflow_A , netflow_B , flow_{AB} , outflow_B and levelDiff . In other words, all of the hidden variables belonging to the general model have been discovered. These 24 variables are listed in Table 4.4 together with the inflow, which was successfully discovered to be constant. The reference column in this table shows how each variable is referenced in the next table, Table 4.5, which is a matrix of the monotonic function discoveries.

Table 4.4. List of variables generated in the coupled tanks experiment

| Ref. | Name | Creation | Corresponds to |
|------|--------|--|----------------------|
| 0 | I | inflow | (<i>Observed</i>) |
| 1 | amtA | amount _A | (<i>Observed</i>) |
| 2 | levA | level _A | (<i>Observed</i>) |
| 3 | amtB | amount _B | (<i>Observed</i>) |
| 4 | levB | level _B | (<i>Observed</i>) |
| 5 | damtA | $\frac{d}{dt}$ amount _A | netflow _A |
| 6 | dlevA | $\frac{d}{dt}$ level _A | (rise _A) |
| 7 | damtB | $\frac{d}{dt}$ amount _B | netflow _B |
| 8 | dlevB | $\frac{d}{dt}$ level _B | (rise _B) |
| 9 | auto1 | amount _A + amount _B | — |
| 10 | auto2 | amount _A − amount _B | — |
| 11 | auto3 | level _A + level _B | — |
| 12 | auto4 | level _A − level _B | levelDiff |
| 13 | auto5 | inflow + $\frac{d}{dt}$ amount _A | — |
| 14 | auto6 | inflow − $\frac{d}{dt}$ amount _A | flow _{AB} |
| 15 | auto7 | inflow + $\frac{d}{dt}$ amount _B | — |
| 16 | auto8 | inflow − $\frac{d}{dt}$ amount _B | — |
| 17 | auto9 | $\frac{d}{dt}$ amount _A + $\frac{d}{dt}$ amount _B | — |
| 18 | auto10 | $\frac{d}{dt}$ amount _A − $\frac{d}{dt}$ amount _B | — |
| 19 | auto11 | $\frac{d}{dt}$ level _A + $\frac{d}{dt}$ level _B | — |
| 20 | auto12 | $\frac{d}{dt}$ level _A − $\frac{d}{dt}$ level _B | — |
| 21 | auto14 | inflow + $\frac{d}{dt}$ amount _A + $\frac{d}{dt}$ amount _B | — |
| 22 | auto16 | inflow + $\frac{d}{dt}$ amount _A − $\frac{d}{dt}$ amount _B | — |
| 23 | auto18 | inflow − $\frac{d}{dt}$ amount _A + $\frac{d}{dt}$ amount _B | — |
| 24 | auto20 | inflow − $\frac{d}{dt}$ amount _A − $\frac{d}{dt}$ amount _B | outflow _B |

With the constant inflow variable not taking part in any of the monotonic function constraints, using the other 24 variables, there are a total of 276 ($\frac{24*23}{2}$) possible variable pairs which can be involved in a monotonic function relationship. It is interesting to observe in Table 4.5 the fact that all of these pairs are either involved in an M^+ or M^- according to LYQUID. The relationships that pass through or very close to the origin are written in bold typeface in the table. Unfortunately not all of these discoveries are correct.

Checking all 276 constraints one by one, by means of plotting each relationship, it was seen that only 196 of them were correct. The correct and incorrect monotonic function discoveries are displayed in Table 4.6. It is clear that all of the incorrect discoveries involve one of either 7th, 8th, 15th or 16th variables of Table 4.4. All of the relationships which are interpreted by LYQUID incorrectly as monotonic functions have something in common. For a long period of the observation interval, these relationships are in fact either monotonically increasing or decreasing as suggested by LYQUID; however, for the remaining and rather short period, the relationship is again monotonic but reversed in character. LYQUID has not been able to treat these relationships as needed.

LYQUID's output model in this coupled tanks experiment might look too over-constrained and this is exactly true when it is compared to the most general model. However, an important proportion of the discoveries are correct for the specific numerical behavior. See Section 6.3 about improving LYQUID's overconstrained outputs in such a way that the output involves more general constraints that would apply to numerical behaviors obtained from several observations.

4.2. Experiments on Noisy Data

4.2.1. U-tube

A noisy dataset for the u-tube was generated by adding $N(0, 1)$ Gaussian noise over the clean samples that were used for the first experiment. Figure 4.2 is the plot of the noisy dataset together with the original clean samples. Table 4.7 lists the constraints which are discovered by LYQUID on this noisy u-tube dataset. The only difference in LYQUID's settings between the two u-tube experiments is the value of $iTolerance$; $iTolerance$ was set to 6×10^{-2} for the experiment on the noisy dataset.

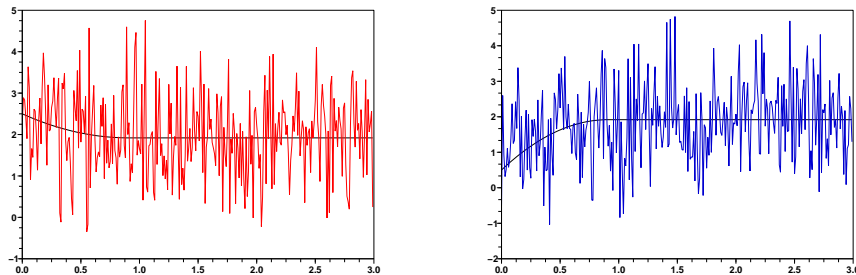


Figure 4.2. U-tube samples with $N(0, 1)$ Gaussian noise: $level_A$ (left), $level_B$ (right)

Obviously, Tables 4.1 and 4.7 differ only in their landmark value pairs. Many constraints in the second experiment lack landmark pairs because intercepts of the discovered relationships were not consistent with the M^+ or M^- . The M^+ between dB and $auto2$, namely $rise_B$ and $levelDiff$, originally passes through the origin and this landmark is a critical one for elimination of unrealistic behavior. LYQUID was able to estimate landmark pairs close to the origin, which is satisfactory considering the amount of noise. The M^- between $rise_A$ and $rise_B$ also has a critical landmark pair at the origin; this pair is implicit, even though not perfectly, from the value pairs of the two monotonic function constraints between $(dA\ auto1)$ and $(dB\ auto1)$. Shortly, the discoveries listed in Table 4.7 are a strong evidence of how tolerant LYQUID is against noise. It is apparent from Figure 4.2 that there are no obvious trends but only fluctuations in the time-series data even after equilibrium is reached; however, LYQUID was still able to capture the nature of the underlying system.

Table 4.7. Discoveries from the noisy u-tube dataset

| | | |
|-----------------|-------------------|------------------|
| d/dt(A dA) | | |
| d/dt(B dB) | | |
| add(A B auto1) | | |
| minus(B -B) | | |
| add(A -B auto2) | | |
| M-(A B) | | |
| M-(A auto1) | | |
| M+(A auto2) | | |
| M+(B auto1) | (-22.8004 0.0000) | (0.0000 4.0153) |
| M-(B auto2) | (1.9367 0.0000) | (0.0000 4.0153) |
| M-(dA dB) | | |
| M+(dA auto1) | (-2.2032 0.0000) | (0.0000 3.8270) |
| M-(dA auto2) | | |
| M-(dB auto1) | (120.3892 0.0000) | (0.0000 3.8643) |
| M+(dB auto2) | (-0.0397 0.0000) | (0.0000 0.0122) |
| M-(auto1 auto2) | (3.8735 0.0000) | (0.0000 45.6008) |
| M-(A dA) | (1.9604 0.0000) | (0.0000 49.1951) |
| M+(A dB) | | |
| M+(B dA) | | |
| M-(B dB) | | |

4.2.2. Cascaded Tanks

A noisy cascaded tanks dataset was created by adding $N(0, 0.5)$ noise over the clean samples which were used in the first cascaded tanks experiment. The clean and noisy datasets are plotted together in Figure 4.3. Seventh degree polynomials were used in this experiment and *iTolerance* was increased to 8×10^{-3} to allow for more error in comparisons.

In this experiment, LYQUID discovered 55 monotonic function constraints which involve 3 observed and 9 hidden variables. All of the constraints listed in Tables 4.2 and 4.3 are successfully discovered among these 55, even though the landmark pairs are not so accurate as in the u-tube case. In fact, all of the discovered monotonic function constraints are correct when the observed numerical behavior is considered; the reason why these constraints were not discovered in the clean dataset experiment is related to the value of *iTolerance*. In the clean dataset experiment, *iTolerance* was set to a very low value, which in turn resulted in rejection of actually correct relationships. The constraints in Table 4.2 and 4.3 are the ones which were able to survive under a very low margin of error. As a result, although it discovered too many constraints for this noisy cascaded tanks dataset, LYQUID was still able to capture the necessary constraints which determine the behavior of the system under a high level of noise.

4.3. Other Experiments

4.3.1. Arbitrarily Sampled Data

In this experiment, LYQUID was tested on numerical u-tube data again. This time, however, the data samples were taken arbitrarily from the whole 3 second observation interval. 300 samples were taken from levels of water in the tubes; even though a specific sampling frequency was not used, the samples were still well distributed on the observation period. The data samples were left clean without artificial noise.

Using the same tolerance with the first u-tube experiment, LYQUID discovered

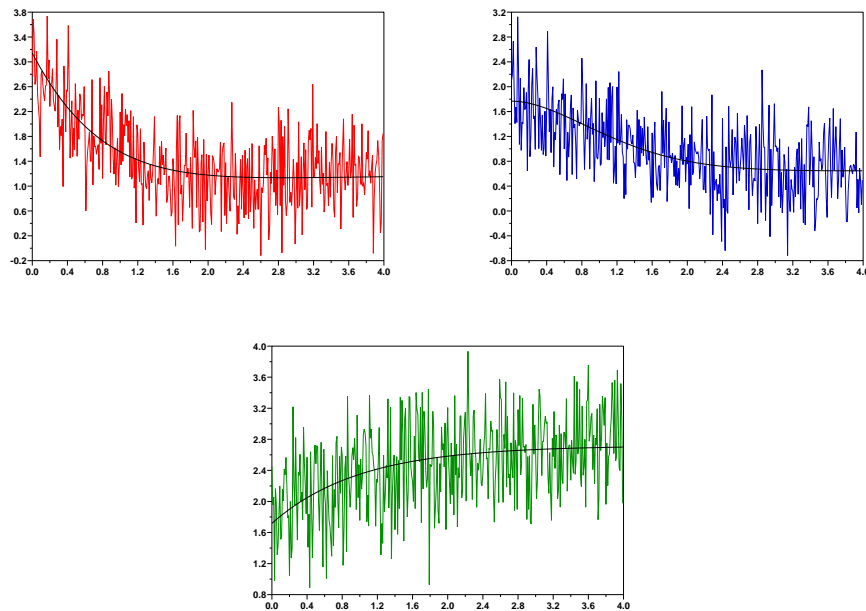


Figure 4.3. Cascaded tanks samples with $N(0, 0.5)$ Gaussian noise: $amount_A$ (top left), $amount_B$ (top right), $inflow$ (bottom)

all of the constraints listed in Table 4.8. The discovered constraints are identical to the discoveries on u-tube datasets with equi-distant samples. This is a clear evidence of how capable LYQUID is when working with arbitrarily sampled data. It shows that LYQUID neither requires data samples to be equally spaced in time nor sampling timepoints of different variables have to coincide.

4.3.2. Missing Data

For the final type of experiment, numerical u-tube simulation was done with the usual settings; however, this time the samples were taken from only the $[0, 0.5]s$ and $[1.5, 2.5]s$ time intervals. There were a total of 150 samples which were equally spaced in the specified time intervals. Note that the timepoint of reaching the equilibrium lies inside the $[0.5, 1.5]s$ time interval, which was left unobserved. Increasing LYQUID's tolerance slightly to 8×10^{-5} , it was possible to end up with the discoveries which are also listed in Table 4.9. Note that, this time also, although some data samples were missing right in between the observation interval, LYQUID was again able to capture all of the necessary constraints with appropriate landmark value pairs.

Table 4.8. Discoveries from the arbitrarily sampled dataset

| | | |
|-----------------|------------------|-------------------|
| d/dt(A dA) | | |
| d/dt(B dB) | | |
| add(A B auto1) | | |
| minus(B -B) | | |
| add(A -B auto2) | | |
| M-(A B) | (2.6920 0.0000) | (0.0000 6.4480) |
| M-(A dA) | (1.9173 0.0000) | (0.0000 18.0090) |
| M-(A auto1) | (4.3603 0.0000) | (0.0000 6.4480) |
| M+(A auto2) | (1.9192 0.0000) | (0.0000 -6.4480) |
| M+(B dA) | (1.9239 0.0000) | (0.0000 -1.5053) |
| M+(B auto1) | (-4.3603 0.0000) | (0.0000 2.6920) |
| M-(B auto2) | (1.9192 0.0000) | (0.0000 2.6920) |
| M-(dA dB) | (-0.0004 0.0000) | (0.0000 -0.0009) |
| M+(dA auto1) | | |
| M-(dA auto2) | (-0.0162 0.0000) | (0.0000 -0.0067) |
| M-(dB auto1) | | |
| M+(dB auto2) | (0.0386 0.0000) | (0.0000 -0.0066) |
| M-(auto1 auto2) | (3.8384 0.0000) | (0.0000 8.7207) |
| M+(A dB) | (1.9173 0.0000) | (0.0000 -33.6867) |
| M-(B dB) | (1.9238 0.0000) | (0.0000 3.7529) |

4.4. Interpreting the Results

This chapter explained results of the experiments performed to test LYQUID's success on benchmark models. Results on these benchmark models may not look significant because the real world models are generally more complex. However, these benchmark models act as the starting point for testing and understanding properties of similar algorithms in almost all of the related literature.

It is obvious from the experimental results that LYQUID is capable of discov-

Table 4.9. Discoveries from the dataset with missing data

| | | |
|-----------------|------------------|-------------------|
| d/dt(A dA) | | |
| d/dt(B dB) | | |
| add(A B auto1) | | |
| minus(B -B) | | |
| add(A -B auto2) | | |
| M-(A B) | (2.7047 0.0000) | (0.0000 6.5978) |
| M-(A dA) | (1.9136 0.0000) | (0.0000 20.3705) |
| M-(A auto1) | (4.5771 0.0000) | (0.0000 6.5978) |
| M+(A auto2) | (1.9188 0.0000) | (0.0000 -6.5978) |
| M+(B dA) | (1.9239 0.0000) | (0.0000 -1.1418) |
| M+(B auto1) | (-4.5771 0.0000) | (0.0000 2.7047) |
| M-(B auto2) | (1.9188 0.0000) | (0.0000 2.7047) |
| M-(dA dB) | (0.0000 0.0000) | |
| M+(dA auto1) | | |
| M-(dA auto2) | (-0.0355 0.0000) | (0.0000 -0.0178) |
| M-(dB auto1) | | |
| M+(dB auto2) | (0.0865 0.0000) | (0.0000 -0.0178) |
| M-(auto1 auto2) | (3.8376 0.0000) | (0.0000 9.1542) |
| M+(A dB) | (1.9136 0.0000) | (0.0000 -49.7620) |
| M-(B dB) | (1.9314 0.0000) | (0.0000 2.7876) |

ering hidden variables. The hidden variable may be a time derivative of an observed variable or it may be a composite variable formed by using addition, subtraction and multiplication as the operations where observed variables and their time derivatives are the operands. The experimented models did not have a hidden variable which was introduced through multiplication; however, the results should suggest that if there was such a variable and if its creation was allowed by the user's multiplication table of units, then it would have been discovered. There is a specific pattern of hidden variable creation in LYQUID and if the variable follows that pattern, then it will be discovered. However, for example, if the hidden variable requires a square (i.e. an energy variable) or if the hidden variable is introduced by applying an arbitrary function to the observed variables, or if the hidden variable is a sum of products rather than a product of sums, then it will not be discovered because it will not be created at the hidden variable generation stage of the algorithm.

It is again obvious from the results that LYQUID is capable of discovering monotonic function constraints correctly and it is also capable of providing useful landmark value pairs for those constraints. Discovery of landmark value pairs is very important because it is generally those value pairs that make a qualitative simulator work correctly. The experimental results show that LYQUID is also able to work under high levels of noise. This is thanks to the design decision of LYQUID about not operating on individual data samples. Even if the data samples are noisy, a least squares approximation is able to come close to the underlying function that generated the samples; this is why LYQUID is successful even when plots of the time-series data do not tell anything. It should also be noted that not only the qualitative models but also the landmark value pairs that are discovered even from the noisy datasets are within acceptable error margins.

Finally, results show that LYQUID is able to perform well on arbitrarily sampled data or even when data samples are missing for certain periods. A test on arbitrarily sampled data was done to point to LYQUID's capability; this capability of working with such data is a feature that is specific to LYQUID. LYQUID does not need the data samples to coincide at specific timepoints nor it needs them to be equally spaced

in time. That experiment was specially designed to demonstrate this capability as it is not a common feature in the literature. Moreover, it was seen that LYQUID was able to tolerate absence of data, which is also not a common feature of other algorithms. LYQUID can tolerate absence of data because the time intervals in which variables are not sampled still get polynomial approximations which are somehow extended from neighboring intervals. Therefore, once model discovery starts, every variable is represented with polynomials on every interval even if the variable is not sampled for certain periods.

5. RELATED WORK

5.1. QMN

There are a wide variety of algorithms in the literature that address the problem of discovering qualitative models from observed qualitative or quantitative behavior. LYQUID is a new algorithm in this field and it creates a qualitative model as output from quantitative data. This section will compare LYQUID with existing algorithms or approaches.

Starting from the most relevant, LYQUID resembles the QMN algorithm [9] deeply. Like LYQUID, QMN takes quantitative data as input and produces a qualitative model as output using the well-known QSIM vocabulary. In other words, the two algorithms are —almost— the same when they are considered as two black boxes. The only difference is related to the sampling properties of the input data. QMN requires all observed variables to be sampled at the same timepoints, which also have to be equally spaced in time. LYQUID, on the other hand, allows observed variables to be sampled irregularly over time, as long as the samples are good enough to capture the behavior of the observed variable.¹ The reason why QMN is so strict in contrast to LYQUID will be clear in the following paragraphs.

QMN and LYQUID consist of two main stages: introduction of new variables and generating—and-testing qualitative constraints. However, there is a fundamental difference between LYQUID and QMN in the way they process the input data samples. QMN processes the samples directly, whereas, LYQUID goes through a preprocessing stage where piecewise polynomials are fitted to the input data samples. After this preprocessing stage, LYQUID never returns back to the original samples, whereas, QMN works on them during all of its execution.

¹As a very simple example, sampling a sine wave at timepoints $t = k\pi$ ($k \in \mathbb{Z}$) produces an observed variable which is constant at zero at all times. These samples do not reflect the nature of the variable and therefore are not good enough.

When introducing new variables, QMN creates the new variables by performing algebraic operations on individual data samples. For example, creating an automatic variable $X + Y$ for QMN is equivalent to adding individual samples belonging to X and Y at all timepoints. Recall that the same thing is done by adding Y 's piecewise polynomial to X 's piecewise polynomial in LYQUID. Time derivatives of observed variables are also calculated by numerical derivation in QMN, which requires a whole pass over the samples of a variable. LYQUID does the same thing by only differentiating a piecewise polynomial. Since QMN has to process individual data samples at specific timepoints, it requires that all observed variables are sampled at the same timepoints. Table 5.1 simply shows how variables are represented internally in QMN and LYQUID. If the variables X and Y in the table were sampled at different timepoints, QMN would have been unable to compute $X + Y$. LYQUID, however, would still be able to arrive at the same polynomials even if the variables were sampled at arbitrary timepoints.

Table 5.1. Variable representations of QMN and LYQUID

| | QMN | | | | | | LYQUID |
|---------|-----|---|---|----|----|----|----------|
| t | 1 | 2 | 3 | 4 | 5 | 6 | $[0, 6]$ |
| X | 1 | 2 | 3 | 4 | 5 | 6 | t |
| Y | 2 | 4 | 6 | 8 | 10 | 12 | $2t$ |
| $X + Y$ | 3 | 6 | 9 | 12 | 15 | 18 | $3t$ |

QMN and LYQUID also have similar parameters. Two of those parameters limit the depth of time derivatives and size of composite variables. Both algorithms also have two tolerance parameters which are used to determine tolerable errors during their execution. Similar to the error criterion (*equal*) in Equation (2.8), QMN uses a measure called *zero* in comparing equality of two variables. It also implements a subroutine called *Satisfied* which tests correctness of a proposed qualitative constraint; qualitative constraints are tested almost in the same way as explained in Section 3.6. One of the differences is to use *zero* during comparisons instead of *equal*. The other difference is with the monotonic function constraints; QMN does not require that the two variables in an $M^{+,-}$ be involved in a functional relationship. This is an advantage of LYQUID over QMN, in the sense that it complies strictly to the definition of

monotonic function constraints. Unlike LYQUID, use of dimensions (units) is optional in QMN; when turned on, it works in the same way with LYQUID.

It is also possible to compare the algorithms from a complexity theoretic point of view. The complexity of QMN is proposed to be $O(N^2 n^{2c} + N n^{3c})$ [9], where all the symbols are used in LYQUID’s notation to prevent confusion. The actual expression is simplified to make the comparison clear. LYQUID’s complexity, on the other hand, could be put simply as $O(n N d^2 + n^{3c} d^2)$. Note that N appears only as a linear factor here and it appears as a factor of only n rather than n^c . The dominating factor in the complexity arises from the model discovery stage in LYQUID and number of processed samples is not a factor in that dominant term. The degree of the polynomials used in the piecewise polynomial approximations plays an important factor in LYQUID’s complexity but d is generally much smaller when compared to N . This is why LYQUID should be considered as the better algorithm from a computational perspective.

LYQUID’s better complexity is a result of the idea of using polynomials —instead of individual data samples— to represent variables. This idea not only results in an improvement in computational efficiency but it also makes the algorithm more robust against noise. Individual data samples may suffer from unexpectedly high amounts of noise. However, LYQUID relies on interpolations of multiple data samples, which decreases the overall deviation from the actual function that is caused by noise. Therefore, even if the data samples are very noisy, LYQUID has the capability of smoothing the contribution of noise to the samples. QMN does not have such a capability as it directly operates on individual data samples, even if they are noisy or not.

Finally, it is a good idea to look at how both algorithms perform on a common model. One of the benchmark models, on which QMN is tested, is the u–tube model [9]. Džeroski and Todorovski describe how they have created their numerical dataset and QMN’s output on that dataset in detail. LYQUID was tested on the very same dataset, which is created by numerical simulation of a realistic u–tube system. This u–tube system is made up two identical cylinders which are connected by a thin pipe at the base. Levels of waters in the tubes are sampled and initial levels are set to 10 and 210.

This system is less general than the u-tubes used for testing LYQUID; the M^- between the change of levels in the general u-tube model is replaced by a minus constraint in this less general system.

When first derivatives and composite variables up to size two are introduced, QMN is reported to discover 362 qualitative constraints that are consistent with the example behavior [9]. These constraints are described to include the critical monotonic function constraint $M^+(\text{levelDiff}, \text{rise}_A)$ that appear in the common u-tube model. The authors also report discovery of a group of redundant constraints. It is obvious that an output model with 362 constraints is too overconstrained for a simple u-tube. LYQUID, on the other hand, discovered a total of only 13 constraints on the same dataset. These constraints are listed in Table 5.2.

Table 5.2. LYQUID's discoveries on the comparison dataset

| | |
|-----------------|--------------|
| d/dt(A dA) | M-(A B) |
| d/dt(B dB) | M-(A dA) |
| add(A B auto1) | M+(A auto2) |
| minus(B -B) | M+(B dA) |
| add(A -B auto2) | M-(B auto2) |
| minus(dA dB) | M-(dA auto2) |
| const(auto1) | |

LYQUID's output model with a much smaller number of constraints is definitely less constrained than QMN's model. However, LYQUID was still capable of discovering the hidden level difference variable as `auto2` and also the fundamental constraints of the u-tube model. This should in fact be expected as LYQUID was able to perform equally well on more general u-tube datasets. There are two main reasons why LYQUID discovered a lower number of constraints: (i) LYQUID requires the existence of a function between two variables in order to include a monotonic function constraint in its output, (ii) LYQUID employs variable pruning, i.e. after it discovered that `dA` and `dB` are equal with opposite signs, it removed `dB` and composites involving `dB` from its variable list; this prevented redundant constraints from being discovered. Both of these

features are specific to LYQUID and are not a part of QMN.

Another advantage of LYQUID over QMN, which is also important for this specific case, is the discovery of landmark values. In LYQUID's output, even though they are not displayed in Table 5.2, constant and monotonic function constraints are accompanied with landmark values or value pairs. This is important because a landmark value pair located at the origin for $M+(A \text{ auto}2)$ is especially necessary to make a correct qualitative simulation of the u-tube. QMN is not reported to discover such a value pair.

QMN is the most similar algorithm to LYQUID in literature and this is the reason why a thorough comparison between the two algorithms has been performed. The fundamental difference of LYQUID from QMN is that it makes polynomial approximations for observed variables prior to looking for qualitative relationships. Taking advantage of such approximations is basically a design decision which is also backed by mathematics; it is apparent that this design decision produces good results.

5.2. LAGRANGE and LAGRANGE

Džeroski and Todorovski propose two other algorithms which are similar to both LYQUID and QMN. One of these algorithms is LAGRANGE [12, 9]. Like QMN and LYQUID, LAGRANGE also requires numerical data as input but in contrast it produces a set of ordinary differential equations rather than discovering a qualitative model as output. LAGRANGE is identical to QMN while it introduces time derivatives of observed variables. It then introduces new variables, either through multiplication or by applying sin and cos functions to variables that are observed in radians. Finally, in the discovery stage it generates quantitative equations rather than qualitative constraints and tests correctness of those equations by using linear regression. This feature of LAGRANGE is reminiscent of LYQUID's existence checks for qualitative constraints; LYQUID uses polynomial regression instead of linear regression. As outputs of LAGRANGE and LYQUID are fundamentally different, it is not possible to compare the algorithms in the same way that is done for QMN.

LAGRANGE generates an ODE as output as opposed to LYQUID’s output in the form of a QDE. In other words, when LAGRANGE discovers an equation, instead of a qualitative relationship, this equation binds one of the variables to a linear combination of several other variables, i.e. $x = y - 2u + 0.4v$. Unless the coefficients in such an equation are 1 or -1 , i.e. $x = y - u + v$, it is left undiscovered by LYQUID. On the other hand, a quantitative equation such as $x = 4y' - 1$ which binds a variable to only another variable is discoverable. That is because there will be an $M^+(x y')$ constraint in the qualitative model for this specific case and LYQUID will suggest a function f such that $x = f(y') \approx 4y' - 1$ along with the M^+ . This is only true because there exists an M^+ in the output model for the quantitative equation $x = 4y' - 1$; normally, if a quantitative equation does not produce an M^+ or M^- in the qualitative output, the equation remains unrevealed by LYQUID. LAGRANGE, however, discovers such equations whatever the circumstances are. As a final remark about LAGRANGE, it should be told that the algorithm is capable of discovering hidden variables in the way LYQUID does and it is capable of discovering ordinary differential equations which are consistent with observed behavior. LAGRANGE is the first equation discovery system to extend the scope of the problem to discovery of ordinary differential equations.

LAGRANGE’s successor is LAGRAMGE [13, 14]. It allows the user to narrow down the search space of possible equations by using a context free grammar. Polynomials are again used here as this context free grammar. LAGRAMGE also makes multi-dimensional polynomial approximations prior to calculating partial derivatives of variables. This helps lower the large errors caused by numerical differentiation in LAGRANGE. However, unlike LYQUID, these polynomial approximations bind variables between themselves, rather than expressing them as functions of time. Finally, LAGRAMGE is equipped with additional search methods and heuristics to perform more efficiently and to favor less complex equation systems. The LAGRANGE family of methods are used especially in discovering quantitative models for biological systems [13, 14]. Unlike conventional system identification algorithms, which work under the assumption of some specific structure, they explore a space of possible equation structures without making any assumptions.

5.3. SQUID

The algorithms discussed up to this point operate on numerical data samples at all times to produce their output. SQUID [15, 16], on the other hand, also starts with numerical data as input but moves in a different direction. Instead of performing operations on quantitative data, SQUID makes a conversion to qualitative data by extracting trajectories, events and envelopes from the numerical input samples. Trajectories are monotonically increasing or decreasing sections of input data and events correspond to extrema within the data; envelopes are defined as upper and lower bounds for the trajectories. The discovery process of a qualitative model is then based on these extracted qualitative data. Model discovery in SQUID is done via a simulator and an inverse simulator. Starting with a big model space, SQUID simulates the model to match simulation results with observations; matching parts of predictions and observations are then mapped back to the model space by inverse simulation so that portions of the model space which are inconsistent with the observed behavior are eliminated. These forward and backward iterations continue until refinement of the model is complete.

What QMN, LAGRANGE and LAGRAMGE do is called raw data matching in SQUID terminology, whereas, SQUID claims to perform trend matching [15]. Kay correctly argues that trend matching is advantageous over raw data matching. This is because qualitative abstractions like trajectories and events describe sets of qualitative models and as a result of this, multiple model possibilities could be ruled out at once very quickly. Moreover, it is much easier to match and compare qualitative properties and even envelopes, when compared with raw data matching. Finally, the same set of qualitative properties match numerical traces of numerous different observations; therefore, the same discovery process could be used to explain and understand several observations.

Looking from this perspective, LYQUID performs neither raw data matching nor trend matching. It does not do raw data matching because it does not operate on individual data samples but rather tries to represent data samples with polynomials in a preprocessing stage. It does not do trend matching because the polynomial descrip-

tions of observed variables are not qualitative descriptions and they are specific only to one observed behavior. Being an approach that is in the middle, LYQUID benefits from advantages of both raw data matching and trend matching. Trend matching is computationally cheaper than raw data matching and LYQUID benefits from this; it eliminates a dominant N^2 term from the complexity of QMN, which employs raw data matching, and it even frees complexity of its model discovery from the number of processed samples. Raw data matching allows more precise mathematical deductions (landmarks) from data samples and allows introduction of hidden variables from quantitative data (which is good because once variables are mapped into the qualitative domain, results of algebraic operations that are used in generating hidden variables become imprecise, i.e. $(+) - (+) = (?)$). Moreover, raw data matching is very suitable for a generate-and-test based approach. LYQUID benefits from all of these advantages.

5.4. Q_{OPH}

Q_{OPH} is a fundamentally different approach that is used in learning qualitative models and it is based on Inductive Logic Programming (ILP) [10, 11]. ILP was in fact posed much earlier than Q_{OPH} as a generic learning methodology which can be applied to various learning problems [17]. Basically, given the observations, which is denoted by E of the word evidence, and background knowledge denoted by B , the ambition in ILP is to find a set of hypotheses H which explains the observations given the background knowledge: $B \wedge H \models E$. The background knowledge for this specific application of ILP is the QSIM vocabulary. The main problem here is to restrict H to the suitable solutions given that there is more than one possible solution for H including the trivial solutions E and $B \rightarrow E$. As discussed in [10], there are several different approaches to solve this general problem but the designers of Q_{OPH} have chosen to use ALEPH ILP system [18] for their Q_{OPH} implementation.

Q_{OPH} can run on both quantitative and qualitative data as input. When supplied with quantitative data as input, however, it does some preprocessing to convert the input data into the qualitative domain. In other words, unlike LYQUID or QMN but similar to SQUID, the model learning phase of Q_{OPH} runs on qualitative data. The

search space in which Q_{OPH} looks for solutions is organized like a tree structure where each node in the tree is a possible qualitative model; depth of each node is determined by the number of qualitative constraints in the qualitative model of that node. When a new level of nodes is to be added to the tree, every node in the previous level branches into a number of child nodes by adding all possible clauses which can be generated from observed variables and the intermediate variables of the node being extended. This branching factor is at least quadratic in the number of observed variables and overall size of the search space is exponential in that same number. Nodes at the same level are ordered according to their cost, which is the number of intermediate variables in the qualitative model of that node.

Q_{OPH} performs a best-first search inside this search space, which means that it first looks for a solution inside a level of nodes going from lower cost to higher cost; if qualitative models in this level of nodes do not pose a solution to the problem, the algorithm extends the tree for another level and continues searching. This means that Q_{OPH} looks for the simplest qualitative model, where simplicity is in terms of the number of qualitative constraints and intermediate variables, that explains the observed behavior. The algorithm relies on pruning heuristics to prevent size of the search space from going huge. Among these heuristics are dimensional analysis which is similar to the usage of units in LYQUID and elimination of redundant constraints. In order for a proposed model to be accepted as a solution, it has to bind all observed variables and it must not be disjoint (i.e. made up of two or more disjoint models).

Like LYQUID, Q_{OPH} was also tested on benchmark models [11]. The algorithm's learning success is judged by a real-valued precision measure in the range $[0, 1]$; this precision is the proportion of the models in the result which are equivalent to the correct model. In other words, higher values of precision represent better learning performance. For every benchmark model, subsets of the model's envisionment are used as training data; an envisionment graph of a qualitative model shows all legal qualitative states and legal transitions between those states. The aim of the authors here is to analyze the change in precision with varying sizes of subsets of the envisionment that is used in training. Results listed in [11] show that precision increases with the number of

qualitative states used as training data. When the number of states used for training is numerous enough, i.e. several different qualitative behaviors from the envisionment are used in training, the precision approaches 1. Results also show that the presence of noise on numerical data lower the value of precision while number of qualitative states used in training is kept constant; however, even in the presence of noise, precision still approaches 1 when the number of qualitative states used in training is increased. Therefore, Q_{OPH} could be called a successful qualitative learning algorithm.

5.5. Other Algorithms

GENMODEL [19] is one of the earliest algorithms in the field of learning qualitative models. It was extended later on to include dimensional analysis and fault tolerance [20]. The algorithm accepts qualitative descriptions of physical behavior in the form of qualitative states. The described architecture in [20], however, is also capable of processing a signal to convert it into a qualitative behavior, which is then input to GENMODEL. Once it starts, GENMODEL first searches the entire history of qualitative states for corresponding value pairs. After that, the algorithm creates its search space by generating all possible permutations of QSIM constraints that are dimensionally correct. Then it removes every constraint from this search space unless the constraint is consistent with the history of qualitative states. Finally, after removal of redundant constraints from the search space, the algorithm outputs the remaining constraints as the qualitative model of the system. The main idea behind GENMODEL is very similar to LYQUID's approach to the problem; GENMODEL, like LYQUID does, generates a list of all possible constraints and retains the ones which are consistent with the observations. Both algorithms also make sure that the remaining constraints are dimensionally correct. The most important difference between the algorithms is that GENMODEL does not even attempt to discover hidden variables. This is also one of the interesting features of the algorithm, which is not very common in literature.

GENMODEL has been tested on physiological signals to produce a qualitative model for a human's cardiovascular system. Given datasets that were obtained from six different patients undergoing bypass surgery, the algorithm was able to learn qualitative

relationships between heart rate, two types of blood pressure, stroke volume, cardiac output etc. Discovering such constraints is no easy task as it depends very much on the clinical conditions of the patient and the drugs that are circulating through the blood. The algorithm made both correct and spurious discoveries, where most of the spurious discoveries are caused by smooth waveforms that do not provide valuable information in the form of qualitative states. The correct discoveries, on the other hand, revealed valuable information about the cardiovascular system [20].

MISQ [21] is another algorithm which operates on qualitative input with a generate-and-test approach for discovering qualitative constraints. Dimensions are again taken into consideration in MISQ. MISQ is unique in its treatment of different forms of incomplete information. MISQ is capable of handling situations where behavioral information is incomplete or dimensions of variables are unknown or some variables are not observed at all. With incomplete qualitative values or variable dimensions, MISQ generates several possible output models. This is because such incomplete knowledge may cause incompatible constraints to be consistent with the observations. In such a case, a model with a pair of incompatible constraints is output as two possible models in the end, with each model containing only one of the constraints from the incompatible pair.

The more interesting case occurs when some variables are not observed at all. In such a case, the qualitative model learned from the observations may be disjoint. In order to connect the disjoint models into a whole, MISQ employs a method called relational pathfinding to discover an intermediate (hidden, unobserved) variable which is part of a constraint that joins two model fragments. For each model fragment, MISQ generates all possible constraints involving only one new variable; the constraint that introduces this new variable restricts the behavior of the new variable. MISQ then looks for intersections between these new variables, where intersections occur as a result of consistent restrictions on the behaviors of the variables. When an intersection is discovered, the new variable at the intersection point becomes registered into the model with the constraints that lead up to its creation. As a result, two model fragments are connected into a larger model. This is how hidden variables are discovered in MISQ. It

happens only when observed variables are not sufficient to create a complete qualitative model.

QSI [22] is another interesting algorithm. At the conceptual level, it is possible to say that QSI uses GENMODEL as a subroutine. Recall that GENMODEL starts with a history of qualitative states and tests all possible constraints for consistency with the observations. That is the point where GENMODEL stops. QSI, however, starts by doing the same thing but when it reaches this point it does not stop. Instead, QSI makes a qualitative simulation of the discovered model and finds out all possible behaviors of the generated model. These results must include the qualitative behavior that is input to the algorithm because the constraints are created to be in harmony with the input behavior in the first place. What matters actually is the additional qualitative behavior in the output of the simulation that is not input to QSI. Such behavior appear in the output of the model's simulation because the discovered qualitative model is not deep enough. Once QSI decides that the discovered model is loose, it creates some new variables from the derivatives, sums, squares etc. of the old ones. QSI then runs another GENMODEL, at least conceptually, to discover the consistent constraints for this extended model. Qualitative simulation and depth testing follows this and QSI iterates similarly until it decides that an output model of GENMODEL is tight enough to produce —only, if possible— the input behavior. Such an output model is likely to include the new variables which are introduced to tighten the qualitative simulation. In this respect, it is possible to call QSI an extension of GENMODEL with the ability of discovering unobserved variables.

6. CONCLUSION

6.1. Overview

Qualitative models are very successful in expressing the relationships between the variables in a physical system. They not only help us understand the structure of the underlying differential equation system but also they allow us to see how a change in one of the variables in the system will affect the others without having to worry about precise values of specific quantities. Above all, simulation of a qualitative model yields all paths of possible future behavior, allowing us to anticipate the possibilities that may happen in the future. The QSIM formalism [3] provides both the vocabulary needed in expressing qualitative relationships and the algorithm needed to simulate a qualitative model written using that vocabulary. The work carried out in this thesis is based on the QSIM formalism.

Converting an ODE into a QDE, i.e. a qualitative model, is an easy task but writing the qualitative model of a physical system from scratch is not easy at all. It requires intervention of an experienced intelligent expert with domain specific knowledge. Automating the process of writing qualitative models from observations by using computers is also hard. The problem of discovering qualitative relationships from observations automatically is a problem that falls into the category of machine learning and several algorithms have been proposed to deal with this problem as discussed in Chapter 5.

In this thesis, a new algorithm, which is called LYQUID, was proposed for learning qualitative models from observed numerical data. The algorithm is based on the generate-and-test principle like many of the other algorithms proposed for the same purpose. The uniqueness of the new algorithm, however, is in the way it processes the input data. Unlike other algorithms, LYQUID uses the time-series data of every observed variable to generate a piecewise polynomial approximation for the variable. These piecewise polynomials are used to represent the observed variables. Once the

piecewise polynomial representations are created, LYQUID never again processes the input samples but only operates on the fitted polynomials. As a matter of fact, there is no other algorithm in literature that treats observed variables as functions of time. This is interesting, because the variables are indeed functions of time, and it is those functions that create the qualitative relationships which are sought for. Finding polynomial approximations for those functions and operating on those polynomials is therefore an intuitive approach.

This intuitive approach proves to be advantageous as it relaxes many of the constraints related with the properties of the input samples. First of all, LYQUID does not require input samples to be equally spaced in time; samples may be distributed irregularly over the observation interval. Nor it requires samples for each observed variable to be taken at the same timepoints. These two properties are not common in similar algorithms in the literature. LYQUID's only requirement for the input data is that the samples should reflect the nature of the underlying functions which they are taken from. Such data are sufficient to fit good polynomial approximations to the actual functions and since LYQUID operates only on the fitted polynomials, this is the only requirement with the input data. Moreover, the idea of working on fitted polynomials applies nicely for tolerating absence of data samples. That is because even if samples are missing at some time intervals, the fitted piecewise polynomials also span those intervals and LYQUID is still able to perform constraint discovery.

LYQUID also has an advantage in terms of computational complexity. The pre-processing stage, where piecewise polynomials are fitted to observed data samples, takes linear time in the number of samples for each observed variable. The model discovery stage never again addresses the data samples and therefore the number of samples do not affect the complexity of that main part of the algorithm. This is important because the number of samples is generally high and such a large number turns out to be the most dominant factor in a product that represents computational complexity. Complexity of model discovery is actually the dominant factor in the overall complexity of the algorithm and it is an improvement to remove the number of samples from that dominant term. LYQUID replaces the complexity of operations performed

on individual data samples by the complexity of polynomial operations at the discovery stage. Polynomial operations are computationally cheap when compared to working on data samples one by one. That is why LYQUID's proposed complexity is lower than QMN, which is most similar to LYQUID in related literature. Recall from the previous analysis that QMN's complexity is quadratic in the number of samples.

The final advantage of LYQUID's approach to the problem becomes clear when it operates on input samples with noise. Even if individual data samples are very noisy, fitting polynomials on a group of samples reduces the overall affect of noise and produces a smoother interpolation curve. These smoother curves generally prove to be more accurate in terms of approximating the original functions. Operating on such curves rather than more noisy individual samples, LYQUID is able to make sound judgements while building the output qualitative model.

The experimental results that are given in Chapter 4 illustrate LYQUID's capabilities very well. LYQUID was able to discover hidden variables that exist in the benchmark models along with the monotonic function constraints they are involved in. LYQUID's success in discovering landmark value pairs for monotonic function constraints was also noticeable. Note that discovery of correct landmark value pairs is an important achievement because such landmark pairs are generally needed by the qualitative simulation algorithm to produce only the realistic and correct behavior in its own output. This success of discovering accurate landmark value pairs was also evident even in experiments with noisy datasets. This again shows LYQUID's capability of tolerating noise as explained in the previous paragraph. Experimental results also illustrate LYQUID's capability of working with irregularly sampled input data and with input data in which some variables are not sampled for certain time intervals.

LYQUID's problem seems to be in producing overconstrained qualitative models as it tends to discover too many constraints even in simple cases. This is partly because after creation of hidden variables the variable list becomes so crowded that numerous correct constraints actually exist in between those variables. It is also partly because the datasets used in experiments are not created from simulation of physical

systems that are more general, e.g. cylindrical u-tubes were simulated to create the datasets rather than using arbitrarily shaped tubes. LYQUID’s problem of creating overconstrained models is addressed in the final section of this chapter as future work.

6.2. LYQUID’s Contributions

LYQUID uses functions of time in the form of polynomials to represent observed variables. No other algorithm among the similar ones takes this approach. In similar algorithms, if a function is somehow used to represent a variable, the function represents the variable in terms of the other variables in the system but not in terms of time. Even if this difference cannot be judged as a major contribution, it currently makes LYQUID unique in its field. The approach is intuitive and proves to be promising because it lowers complexity and makes the algorithm more stable against high levels of noise. It is also possible to use other types of functions instead of polynomials to represent observed variables, such as sine and cosine functions of the Fourier expansion; LYQUID prefers polynomials only because of ease of computation. Additionally, analyzing the fitted functions for points of extrema looks like a good solution to the problem of discovering numerical landmark values even in noisy environments.

Another difference of LYQUID from similar algorithms is in the discovery of monotonic function constraints. Consider the time-series data given in Table 6.1 for two observed variables. In the example, x starts from 2 and increases upto 5 during the first three seconds; after that it decreases back to 2. In the meantime, y goes from 4 to 7 during the initial three seconds and ends with the value 1 after decreasing for the rest of the total six seconds. Note that the directions of change of x and y are the same for all of the six second observation interval. This may imply that there is an M^+ relationship between x and y . However, there does not exist a function f such that $y = f(x)$ because $y(0) \neq y(6)$ even though $x(0) = x(6)$. (Recall that x and y are functions of time themselves.) Since such a function does not exist, it is not possible to conclude that an M^+ between x and y exists.

Table 6.1. Time-series data for two observed variables

| | $t = 0$ | $t \in (0, 3)$ | $t = 3$ | $t \in (3, 6)$ | $t = 6$ |
|---|---------|-------------------|---------|-------------------|---------|
| x | 2 | $2 \rightarrow 5$ | 5 | $5 \rightarrow 2$ | 2 |
| y | 4 | $4 \rightarrow 7$ | 7 | $7 \rightarrow 1$ | 1 |

None of the algorithms in the literature take such a restriction into consideration even though the restriction is implicitly required by the definition of the monotonic function relationships. Although it is not so meaningful to implement such a control in algorithms that operate that on qualitative input data, algorithms that process numerical input data should implement such a control. This restriction is an important way of eliminating incorrect discoveries of monotonic function relationships. LYQUID implements such a control and it is considered to be a remarkable advantage.

6.3. Further Work

It has already been discussed that LYQUID produces qualitative models that are overconstrained. Even if all of the discovered constraints are correct discoveries, most of these discoveries are specific to the observation dataset. In most of the cases, the most general qualitative model that we are looking for do not contain many of such constraints. For example, assume that a dataset is created by observing levels of water in a u-tube with two identical columns. When this dataset is given to LYQUID as input, it will discover

$$\begin{aligned}
 &d/dt(level_A, rise_A) \\
 &d/dt(level_B, rise_B) \\
 &minus(rise_A, rise_B) \\
 &add(level_A, level_B, total) \\
 &const(total)
 \end{aligned}$$

as valid constraints among others. These constraints are correct for the specific case but they are not general enough. In the general case, the sum of the levels is not constant and the add and const constraints must be removed from the output. Again in the general case, the minus constraint must be replaced with an M^- . As future work, LYQUID could be modified in such a way that it removes or replaces such correct but

too specific discoveries.

The idea here is to make LYQUID merge outputs of multiple runs on several datasets obtained from several different systems of the same family, i.e. systems having the same qualitative model. Such a merge operation is supposed to remove a constraint from the final output if the constraint does not appear in the outputs of all of the runs. The merge operation must also be able to replace a minus constraint with an M^- , for example, when all of the outputs except one contains an $M^-(x, y)$ but only one of them has $\text{minus}(x, y)$. Same constraints with different landmark value pairs should also be removed from the final output because landmark value pairs of a general constraint must not change from one run to another; if the same constraint differs in its landmark value pairs, then it does not deserve to stay in the final generalized output. Such a merge operation would eliminate constraints that are specific to the scenario under which a dataset is created. On the contrary, it would protect constraints which are general enough to apply to all of the observations. Therefore, the final model obtained by merging different models would be less constrained and more general.

This idea is reminiscent of the use of the complete envisionment as input to Q_{OPH} [10]. That is because when Q_{OPH} receives the complete envisionment with all of the possible qualitative states as input, this input contains several different qualitative behaviors in it. When discovering a qualitative model for this input, Q_{OPH} has to look for a general model that would apply to all of these possible behaviours. Similarly, when LYQUID is run over several numerical traces of data and the outputs are merged into a final model, the final model has to be general enough to explain all of the input behaviors. It would even be better if the numerical traces that are input to LYQUID correspond to different paths (qualitative behaviors) on the envisionment graph.

One final addition is to postprocess LYQUID's output to convert the output to a QSIM input. Such a postprocessing could, for example, make discovery of new landmark value pairs possible in some cases. A good example exists in Table 4.7: according to the constraints $M^+(\text{dA auto1})$ and $M^-(\text{dB auto1})$ in this qualitative model, when auto1 is 3.8270, \mathbf{a} is 0 and when auto1 is 3.8643, this time \mathbf{b} is 0. A postprocessor

could evaluate easily that **a** and **b** become zero at almost the same time and that the actual difference in the values of **auto1** result from approximation errors. Then, the postprocessor can automatically fill in the missing (0, 0) value pair of the M-(**dA** **dB**) constraint; recall that this landmark pair has critical importance in the qualitative simulation of the u-tube model.

REFERENCES

1. Forbus, K. D., “Qualitative Process Theory”, *Artificial Intelligence*, Vol. 24, pp. 169–204, 1984.
2. Kuipers, B., “Qualitative Simulation”, *Artificial Intelligence*, Vol. 26, pp. 289–338, 1986, reprinted in *Qualitative Reasoning about Physical Systems*, ed. Daniel Weld and J. De Kleer, Morgan Kaufmann, 1990, pp. 236–260.
3. Kuipers, B., *Qualitative reasoning: modeling and simulation with incomplete knowledge*, MIT Press, Cambridge, MA, USA, 1994.
4. Travé-Massuyès, L., L. Ieroni, and P. Dague, “Mathematical Foundations of Qualitative Reasoning”, *AI Magazine*, Vol. 24, No. 4, pp. 91–106, 2004.
5. Bratko, I. and D. Šuc, “Learning Qualitative Models”, *AI Magazine*, Vol. 24, No. 4, pp. 107–119, 2004.
6. Shiavi, R., *Applied Statistical Signal Analysis*, Academic Press, London, UK, 2nd edition, 1999.
7. Ralston, A. and P. Rabinowitz, *A First Course in Numerical Analysis*, Dover Publications, Mineola, New York, 2nd edition, 2001.
8. Berleant, D. and B. J. Kuipers, “Qualitative and quantitative simulation: bridging the gap”, *Artificial Intelligence*, Vol. 95, No. 2, pp. 215–255, 1997.
9. Džeroski, S. and L. Todorovski, “Discovering Dynamics: From Inductive Logic Programming to Machine Discovery”, *Journal of Intelligent Information Systems*, Vol. 4, No. 1, pp. 89–108, 1995.
10. Coghill, G. M., S. M. Garrett, and R. D. King, “Learning Qualitative Models in the Presence of Noise”, *Proceedings of the 16th International Workshop on Qualitative*

- Reasoning: QR'02*, pp. 27–35, Sitges, Spain, 2002.
11. Coghill, G. M., S. M. Garrett, A. Srinivasan, and R. D. King, “Qualitative System Identification from Imperfect Data”, Technical report, University of Aberdeen, 2005.
 12. Džeroski, S. and L. Todorovski, “Discovering Dynamics”, *Proceedings of the 10th International Conference on Machine Learning*, pp. 97–103, Morgan Kaufmann, Amherst, MA, USA, 1993.
 13. Todorovski, L., S. Džeroski, A. Srinivasan, J. Whiteley, and D. Gavaghan, “Discovering the Structure of Partial Differential Equations from Example Behavior”, *Proceedings of the 17th International Conf. on Machine Learning*, pp. 991–998, Morgan Kaufmann, San Francisco, CA, 2000.
 14. Džeroski, S. and L. Todorovski, “Learning Population Dynamics Models from Data and Domain Knowledge”, *Ecological Modelling*, Vol. 170, No. 2, pp. 129–140, 2003.
 15. Kay, H., “Robust identification using semiquantitative methods”, In IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, 1997.
 16. Kay, H., B. Rinner, and B. Kuipers, “Semi-quantitative system identification”, *Artificial Intelligence*, Vol. 119, No. 1-2, pp. 103–140, 2000.
 17. Plotkin, G., *Automatic Methods of Inductive Inference*, Ph.D. thesis, Edinburgh University, 1971.
 18. Srinivasan, A., “Aleph Web Site”, http://web.comlab.ox.ac.uk/oucl/research/areas/mach-learn/Aleph/aleph_toc.html, 2000.
 19. Coiera, E. W., “Learning qualitative models from example behaviours”, *Proceedings of the 3rd Workshop on Qualitative Physics*, pp. 45–51, Stanford, 1989.
 20. Hau, D. T. and E. W. Coiera, “Learning qualitative models of dynamic systems”,

Machine Learning, Vol. 26, pp. 177–211, 1993.

21. Richards, B. L., I. Kraan, and B. Kuipers, “Automatic Abduction of Qualitative Models”, *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 723–728, The AAAI Press, Menlo Park, California, San Jose, California, 1992.
22. Say, A. C. C. and S. Kuru, “Qualitative system identification: deriving structure from behavior”, *Artificial Intelligence*, Vol. 83, No. 1, pp. 75–141, 1996.